N O T I C E

8.0 - 1 0.1 1 5

AgRISTARS

A Joint Program for
Agriculture and
Resources Inventory
Surveys Through
Aerospace
Remote Sensing

Supporting Research

November 1979

Final Report

Vol. III
Processing Techniques Development
Part 2: Data Preprocessing and
Information Extraction Techniques

by **P. H. Swain, P. E. Anuta, D. A. Landgrebe and H. J. Siegel**

Laboratory for Applications of Remote Sensing
Purdue University
West Lafayette, Indiana   47907

NASA

FINAL REPORT

VOL. III   PROCESSING TECHNIQUES DEVELOPMENT

PART 2:   DATA PREPROCESSING AND INFORMATION EXTRACTION TECHNIQUES

BY

P.H. Swain, P.E. Anuta, D.A. Landgrebe, H.J. Siegel

Purdue University
Laboratory for Applications of Remote Sensing
West Lafayette, Indiana   47906

November 1979

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

2C. MULTISPECTRAL DATA ANALYSIS RESEARCH

This task consists of three subtasks involving research into advanced methods for classifying multispectral remote sensing data. The first two are multiyear investigations resulting from proposals submitted to NASA in response to the 1978 Applications Notice, OSTA-78-A (April 19, 1978)*. The first year of work on both of these subtasks is reported here.

The third subtask resulted from a proposal submitted to NASA Johnson Space Center during the contract year.$ The work was funded quite late in the year and will be continued. A background discussion is contained in this report.

---

* Proposals entitled "Design and Applications of Multistage Classifiers for Earth Resources Data Analysis" and "Analysis of Multispectral Earth Resources Data Using Context." Principal Investigator on both proposals was Philip H. Swain.

$ Proposal entitled "An Addendum to Research in Remote Sensing of Agriculture, Earth Resources and the Environment." Principal Investigator: D. A. Landgrebe.

2C1. MULTISTAGE CLASSIFICATION

D. A. Landgrebe, M. Muasher, P. H. Swain

1. Introduction

A number of different types of classifiers are now in routine use in remote sensing. With the emergence of more complex data sets, however, the need has been recognized for more sophisticated classifiers providing higher performance and lower cost. The objectives of subtask 2C1 are to continue progress toward the development of such advanced classification techniques. More specifically, the objectives for the current year have been (1) to test known multistage procedures and (2) to begin the development of optimal design procedures for such classifiers.

This task is a continuing task and the work is still in preliminary stages. Therefore this report is in the nature of a progress report, containing no final results.

2. Literature Survey

In order to assess earlier work in this area, and to gain better understanding of the problem, a literature survey was conducted and a bibliography assembled (see Appendix 2C1 for complete survey). The survey lists the main approaches taken to deal with the problem, citing both their advantages and drawbacks.

3. Resources

3.1 Available Software

Software from previous studies (Ref. [2] in Appendix 2C1) was available. Certain problems with the software were corrected and the software was then tested.

3.2 Data Sets

The assembling of data sets for use in design and test tasks proved to be a troublesome process. Flightline 210 from the 1971 Corn Blight Watch Experiment was chosen. Some information about the data set appears below:

Run Number	No. of Channels	No. of Classes	Date Data Collected
71023500	9	6	June 28, 1971

Reasons for initially choosing this data set were:

1. The large variety of classes represented in the set, containing water, forestry, pasture, corn, soybeans and wheat.

2. The large number of channels available to work with (9) than Landsat sets would offer.

3. The difficulty in statistically separating certain classes (pasture and wheat, pasture and corn, corn and wheat). This was thought to serve as a test as to whether any new methods could improve on the accuracy.

4. Available ground truth.

4. Procedures Used

4.1 Transformation of Data

To aid the process of feature selection, and to obtain a nearly uncorrelated set of features, a principle-components transformation was applied to the data. All 9 channels were used in the transformation, and then the first 6 transformed channels were used in the analyses because they carried 99% of the information.

4.2 Class-Conditional and Aggregate Clustering

Two methods for deriving class statistics were used. In the class-conditional clustering method, training fields corresponding to the same informational classes were clustered together. The resulting clusters were identified as subclasses of that informational class. Statistics for each cluster were calculated and used as a basis for classification (after some refinement).

In the aggregate clustering approach, all training fields were clustered together. The individual clusters were then each identified with the appropriate informational classes, and statistics calculated to serve as a basis for classification.

4.3 Comparison of Conventional and Multistage Classification

Using the statistics from 4.2 above, classification was performed using the conventional single-stage classifier and the multistage classifier available from previous studies. The results appear in the following section.

4.4 Hierarchical Clustering

In this method, the training data set was divided into two clusters. These in turn were subdivided each into two clusters, etc., creating a binary tree. The terminal clusters were then identified with informational classes. Statistics at each node were calculated to determine the optimal subset of features to use at that node. Classification was then performed using the available layered classification. Results appear in the next section.

5. Results

Figure 2C1 shows the binary tree obtained from the hierarchical clustering method. The terminal clusters are identified with informational classes and labeled as such. The different methods used in training and classifying were:
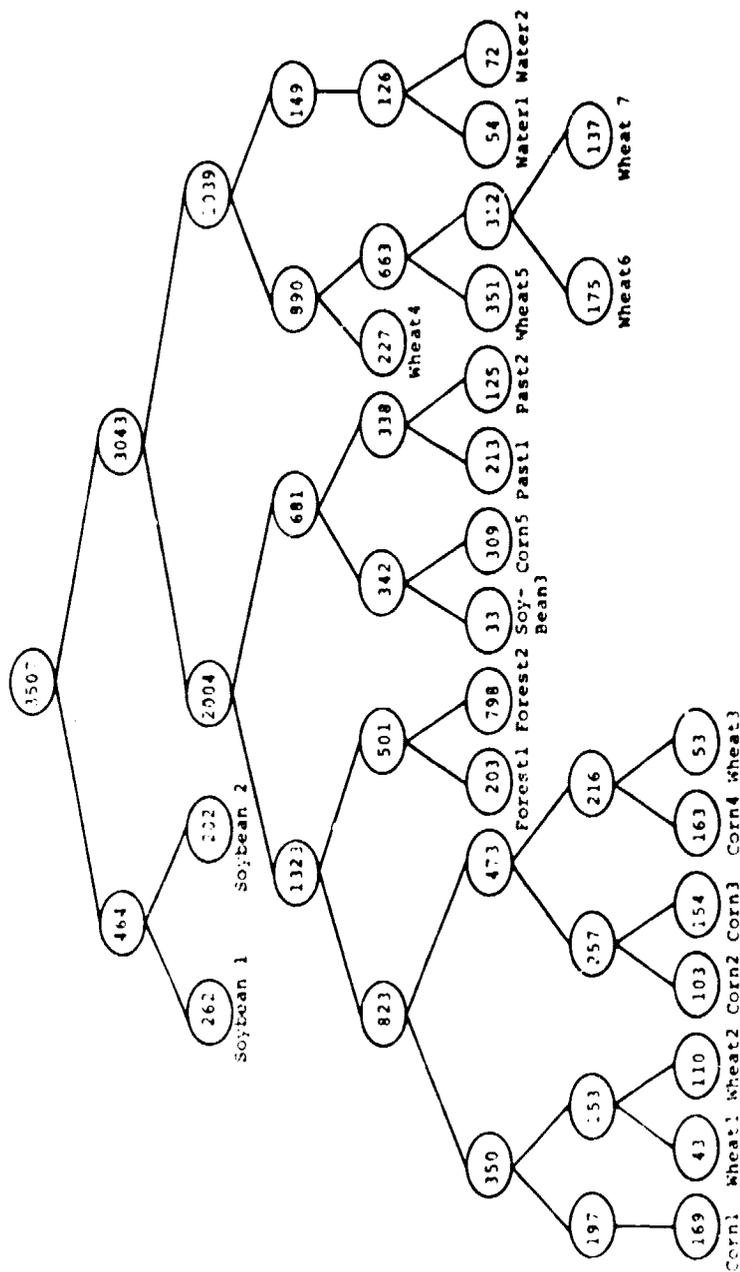
Figure 2C1. Binary tree obtained from hierarchical clustering method. The nodes show the number of data points at each node. The terminal nodes are labeled by their respective classes.

1.   The hierarchical clustering method.

2.   Class-conditional clustering, untransformed data, single-stage classification.

3.   Aggregate clustering, untransformed data, single-stage classification.

4.   Class-conditional clustering, untransformed data, layered classification.

5.   Class-conditional clustering, transformed data, layered classification.

6.   Class-conditional clustering, transformed data, minimum-distance (linear) classifier.

6.   Discussion

The overall accuracy did not change appreciably with the change in training or classification methods. There were two exceptions to this. The minimum-distance classifier did not perform as well as the others; it is assumed that this is because the classifier is linear and is suboptional from the Bayes point of view. The hierarchical clustering method relied heavily on the clustering algorithm, and on the distribution of data in high-dimensional space. Since it is not known or guaranteed that data in n-dimensioral space tend to cluster according to classes, the method did not prove effective in being able to separate clusters into representative informational classes.

Although much has been learned from the different methods used, the results do not seem to conform to theory. No appreciable gain was achieved by using different methods. Based upon a review of the histograms and statistics of the classes, it appears that the subclasses were unimodal, but there was much overlap among certain classes (pasture - wheat - corn). Attempts at improving the results have not been successful.

Wheat did appreciably better in the hierarchical clustering approach than in the other approaches, but pasture did much poorer in the same approach. Corn did best in the transformed, class-conditional clustering, single-stage classifier approach. The results are inconclusive, however, and further work will be done before any final results can be reported. It was observed that the present layered classifer is very effective in reducing the time needed for carrying out a classification.

The transformation of data looks promising at it produces uncorrelated features. Further, it imposes an approximate ordering in terms of the "importance" of the features, i.e., the first feature is likely to be more important than the second, etc. A possible disadvantage of using it is that features will have a larger variance, thus suggesting the need for a larger number of training samples to adequately estimate the distributions.

These are examples of several factors which could be contributing to the trends observed in the results, and these factors must be investigated. However, before doing so, an additional data set will be chosen and similar tests conducted on it.

APPENDIX 2C1
THE LAYERED CLASSIFIER: REVIEW OF LITERATURE

Most of the classification algorithms that have been used in remote sensing for information extraction using pattern recognition techniques can be regarded as "single-stage" classifiers, whereby an "unknown" pattern is tested against all classes using one feature subset, and then the pattern is assigned to one of the present classes in a single-stage decision procedure.

In recent years, as classification of multispectral data found a larger number of users and wider range of applications, the need has been felt for alternate, more powerful techniques than the conventional classifiers, where more information could be extracted more accurately and/or efficiently from the scene. Some of the reasons that warranted this need include:

1. The emergence of more complex data sets with the launching of Landsat-D with its Thematic Mapper sensor, and the need both to handle the data acquired efficiently and the ability to extract more information from the data.

2. As pattern recognition methods developed they found a larger number of users with a wider range of applications. The feedback from these different and versatile uses indicated problems and needs not initially present.

3. There are some applications where conventional classifiers have proved to be marginal at best. Some of these are listed in Swain et al.[1] and include multi-image analysis and the use of mixed feature types.

4. The conventional classifiers use only one particular feature subset and are somewhat inefficient as they must compare an unknown pattern against all possible classes before assigning that pattern to a particular class.

Because of these and other factors, there has been some research in recent years directed toward developing multistage classifiers, whereby the decision procedures go through several stages before finally assigning a pattern to a class.

There has been some earlier work aimed at grouping together the methods of designing multistage classifiers already reported in the literature [2,3]. In general, one can group the earlier work into two main categories:

1. Sequential classification methods. These can be found in several papers and books [4,5,6] in this area. Basically, the method consists of observations made on feature measurements, one at a time. After an observation is made, the classifier either reaches a final decision and the process is terminated, or it makes another observation until a final decision is reached.

2. Hierarchical classification methods. Examples of work in this area can be found in a review paper by Kanal [7], in papers by Mattson et al.[8], Meisel et al.[9], Nadler [10] and Wu [2].

As pointed out by Kulkarni [3], hierarchical methods differ from sequential ones in certain aspects. While in any sequential schemes any class can be accepted at any stage of the measurement process, in hierarchical schemes classes are rejected from consideration at each stage. Also, sequential methods impose a linear ordering on the features. In hierarchical methods, features used along a decision path can be different from those used along another path.

Several heuristic methods of constructing tree designs are proposed in the literature. There have been some studies done in using optimization methods to automate the classifier design procedure, but these are still at an early stage. Meisel et al.[9] presented a two-stage partitioning algorithm for the design of an optimal tree. In the first stage, a suboptimal sufficient partition is obtained. The second stage optimizes the result of the first stage through a dynamic programming approach. The

method uses a binary decision tree, but only linear discriminant functions are allowed to partition the space.

Dynamic programming and branch-and-bound methodologies were used by Kulkarni et al.[3] in design of hierarchical classifiers. The criterion of optimality they use is a weighted sum of the probability of error and the average measurement cost incurred in classifying a random sample. Also, the design of the "optimal tree" assumes a very low error rate for the tree. Further, the authors use only one best feature at each tree node. Although the authors present some methods of reducing the complexity of their design algorithms, the examples they and previous papers have used involve only a small number of classes and features.

In 1974, Wu [2] published a thesis on a decision tree approach with direct application to multispectral data analysis. He proposed several design procedures, one of which is manual, with special emphasis on a heuristic, machine-implemented approach. The optimality criterion he used is again a weighted sum of computation cost and accuracy. He presented results which show superiority in efficiency and/or accuracy over the conventional classifier. The method involves many approximations, is heuristic in many respects, and is certainly suboptimal. However, it may serve as a starting point for the design of an optimal approach, especially since it was used successfully for some remote sensing applications [11,12].

7. References

1. Swain, P. H. and Hauska, H., "The Decision Tree Classifier: Design and Potential," IEEE Trans. Geoscience Electronics, Vol. GE-15, No. 3, July 1977.

2. Wu, C. L., "The Decision Tree Approach to Classification," Ph.D. dissertation, TR-EE75-17, Purdue University, West Lafayette, IN, 1974.

3. Kulkarni, A. V. and Kanal, L. N., "An Optimization Approach to Hierarchical Classifier Design," Proc. Third Int. Joint Conf. on Pattern Recognition (Coronado, CA, Nov. 1976) IEEE Cat. No. 76CH/140-3C.

4. Fu, K. S., Sequential Methods in Pattern Recognition and Machine Learning, Academic Press, 1968.

5. Fu, K. S., Chien, Y. T. and Cardillo, G. P., "A Dynamic Programming Approach to Sequential Pattern Recognition," IEEE Trans. Computers, Dec. 1967.

6. Wald, A., Sequential Analysis, Wiley, New York, 1947.

7. Kanal, L., "Patterns in Pattern Recognition, 1968-1974," IEEE Trans. Info. Theory, Vol. IT-20, No. 6, Nov. 1974.

8. Mattson, R. L. and Dammann, J. E., "A Technique for Determining and Coding Subclass in Pattern Recognition Problems," IBM Journal, July 1965.

9. Meisel, W. S. and Michalopoulous, D. A., "A Partitioning Algorithm With Application in Pattern Classification and the Optimization of Decision Trees," IEEE Trans. Computers, Vol. C-22, pp. 93-103, Jan. 1973.

10. Nadler, M., "Error and Reject Rates in a Hierarchical Pattern Recognition," IEEE Trans. Computers, Vol. C-20, Dec. 1971.

11. Swain, P. H., Wu, C. L., Landgrebe, D. A. and Hauska, H., "Layered Classification Techniques for Remote Sensing Applications," LARS Information Note 061275, Laboratory for Applications of Remote Sensing (LARS), Purdue University, West Lafayette, IN 47907, 1975.

12. Bartolucci, L. A., Swain, P. H. and Wu, C. L.. "Selective Radiant Temperature Mapping Using a Layered Classifier," IEEE Trans. Geoscience Electronics, Vol. GE-14, pp. 101-106, Apr. 1976.

2C2. CONTEXTUAL CLASSIFICATION
Philip H. Swain and Howard Jay Siegel*

1. Introduction

Multispectral image data collected by remote sensing devices aboard
aircraft and spacecraft are relatively complex data entities. Both the
spatial attributes and spectral attributes of these data are known to be
information bearing [1], but to reduce the magnitude of the computations
involved, most analysis efforts have focused on one or the other. Only
within the last few years have serious efforts been made to utilize them
jointly. For example, one approach uses the spectral homogeneity of
"objects," such as agricultural fields, to segment the scene and then
uses sample classification to assign each object as a whole, rather than
its individual pixels (picture elements), to an appropriate ground cover
class [2]. Another approach involves extraction of features based on
gray-tone spatial-dependency matrices from which texture-like character-
istics are developed [3].

In this project we are developing a more general way to exploit the
spatial/spectral context of a pixel to achieve accurate classification.
Just as in written English one can expect to find certain letters occur-
ring regularly in particular arrangements with other letters (qu, ee, est,
tion), so certain classes of ground cover are likely to occur in the "con-
text" of others. The former phenomenon has been used to improve charac-
ter recognition accuracy in text-reading machines. We have demonstrated
that the latter can be used to improve accuracy in classifying remote
sensing data. Intuitively this should not be surprising since one can
easily think of ground cover classes more likely to occur in some con-
texts than in others. One does not expect to find wheat growing in the

midst of a housing subdivision, for example. A close-grown, lush vegetative cover in such a location is more likely the turf of a lawn.

This report contains the theoretical foundations of a contextual classifier, experimental results from applying the contextual classifier to a variety of very different sets of data, and an extensive discussion of multiprocessor implementation of the classifier algorithm.

2. The Contextual Classifier Model

Consistent with the general characteristics of imaging systems for remote sensing, we assume a two-dimensional array of $N = N_1 x N_2$ pixels of fixed but unknown classification, as shown in Figure 2C2.1.

Associated with the pixel having image coordinates (i,j) is its true state or true classification $\theta_{ij} \epsilon \Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$, and a random measurement vector (observation) $X_{ij} \epsilon R^n$ having class-conditional density $p(X_{ij}|\theta_{ij})$. We note that $\{p(X|\omega_i), i=1,2,\ldots,m\}$ is the set of class-conditional probability density functions associating the multispectral measurement vector X with the classes.

Let $\underline{X}$ denote a vector whose components are the ordered pixel measurement vectors:

$$\underline{X} = [X_{ij}|i=1,2,\ldots,N_1; j = 1,2,\ldots,N_2]^T .$$

Similarly, let $\underline{\theta}$ be the vector of states:

$$\underline{\theta} = [\theta_{ij}|i=1,2,\ldots,N_1; j = 1,2,\ldots,N_2]^T .$$

The individual measurement vectors are assumed to be class-conditionally independent; that is, their joint density can be written as:

$$p(\underline{X}|\underline{\theta}) = \prod_{i,j} p(X_{ij}|\theta_{ij}). \tag{2.1}$$

Evidence that this is a reasonable assumption may be found in reference [4].

$$\begin{array}{cccc} \theta_{11} & \theta_{12} & \cdots & \theta_{1N_2} \\ \theta_{21} & \theta_{22} & \cdots & \theta_{2N_2} \\ \vdots & & & \\ \theta_{N_1 1} & & \cdots & \theta_{N_1 N_2} \end{array}$$
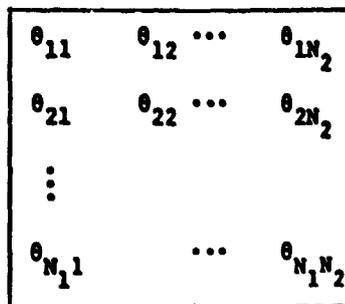
Figure 2C2.1. A two-dimensional array of $N = N_1 \times N_2$ pixels.
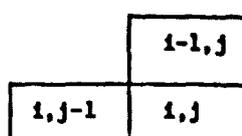
a p=3 choice
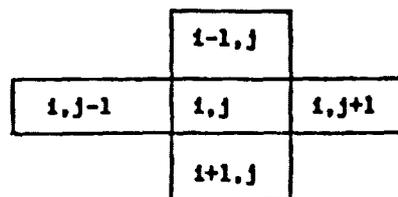
a p=5 choice

Figure 2C2.2. Examples of p-context arrays.

Let the action (classification) taken with respect to pixel (i,j) be denoted by $a_{ij} \epsilon \Omega$. The loss suffered by taking action a_{ij} when the true class is θ_{ij} is denoted by $L(\theta_{ij}, a_{ij})$ for some fixed non-negative function $L(.,.)$. Then the average loss suffered over the N classifications in the array is

$$L = \frac{1}{N} \sum_{i,j} L(\theta_{ij}, a_{ij}).$$

If we make the action a_{ij} a function of the observations, then for a given array $\underline{\theta}$ the expected average loss (or risk) is

$$R_{\underline{\theta}} = E\left[\frac{1}{N} \sum_{i,j} L(\theta_{ij}, a_{ij}(\underline{X}))\right] \qquad (2.2)$$

where the expectation is with respect to the distribution of the vector of observations.

Our objective may be stated as follows: We want to determine the dependence of the decision function $a_{ij}(\cdot)$ on $\underline{X}$ in such a way that for any given array $\underline{\theta}$, the risk, equation (2.2), will be minimum. One way to approach the problem of making R_{θ} small is to view $\underline{\theta}$ as a realization of a random process in two dimensions and to derive a decision rule which is Bayes versus this "prior distribution" for $\underline{\theta}$ (probably under some simplifying assumptions concerning the nature of this process). This is the approach of Welch and Salter [5] and Yu [6], who make assumptions on the random process sufficient to guarantee that the Bayes decision concerning pixel (i,j) depends on $\underline{X}$ only through X_{ij} and the four nearest neighbors of the pixel.

We will adopt an approach to controlling R_{θ} through $a_{ij}(\cdot)$ that is more closely related to the large body of statistical literature traceable to Robbins [7], and known as compound decision theory. See, for example, the works and references of VanRyzin [8,9], Cover and Shenhar [10], and Vardeman [11,12]. Rather than looking for a distribution for $\underline{\theta}$ whose associated Bayes rule is both simple and has small R_{θ} for most $\underline{\theta}$, we use the following argument. First, specify some arrangement of p pixel locations including a pixel to be classified. Call this arrangement the p-context array, several choices of which are shown in Figure 2C2.2.

Let $\underline{\theta}^P \epsilon \Omega^P$ and $\underline{X}^P \epsilon (R^n)^P$ stand respectively for p-vectors of classes and n-dimensional measurements; each component of $\underline{\theta}^P$ is a variable which can take on values in Ω; each component of $\underline{X}^P$ is a random n-dimensional vector which can take on values in the observation space. Correspondence of the components of $\underline{\theta}^P$ and $\underline{X}^P$ to the positions in the p-context array is fixed but arbitrary except that the pixel to be classified in the array will always correspond to the pth component. The notation $\underline{\theta}_{ij}$ and $\underline{X}_{ij}$ will refer to the particular instance of $\underline{\theta}^P$ and $\underline{X}^P$ associated with pixel (i,j).

Now, consider finding an optimal decision rule of the form

$$a_{ij}(\underline{X}) = d(\underline{X}_{ij}) \tag{2.3}$$

for a fixed function $d(\cdot)$ mapping p-vectors of observations to actions. The risk associated with any rule of this form is, from equation (2.2),

$$
\begin{aligned}
R_{\underline{\theta}} &= E\left[\frac{1}{N}\sum_{i,j} L\left(\theta_{ij}, d(\underline{X}_{ij})\right)\right] \\
&= \frac{1}{N}\sum_{i,j} E\left[L\left(\theta_{ij}, d(\underline{X}_{ij})\right)\right] \\
&= \sum_{\underline{\theta}^P \epsilon \Omega^P} G(\underline{\theta}^P) E\left[L\left(\theta_p, d(\underline{X}^P)\right)\right]
\end{aligned}
\tag{2.4}
$$

where $G(\underline{\theta}^P)$, the **context distribution**, is the relative frequency with which $\underline{\theta}^P$ occurs in the array $\underline{\theta}$ and θ_p is the pth component of $\underline{\theta}^P$. Notice that $R_{\underline{\theta}}$ depends on $\underline{\theta}$ only through $G(\underline{\theta}^P)$. Writing equation (2.4) in more detail and invoking the class-conditional independence assumption, equation (2.1), we have

$$
R_{\underline{\theta}} = \sum_{\underline{\theta}^P \epsilon \Omega^P} G(\underline{\theta}^P) \int L\left(\theta_p, d(\underline{X}^P)\right) \prod_{i=1}^{P} p(X_i | \theta_i) d\underline{X}^P
$$

$$
= \int \sum_{\underline{\theta}^P \epsilon \Omega^P} G(\underline{\theta}^P) L\left(\theta_p, d(\underline{X}^P)\right) \prod_{i=1}^{P} p(X_i | \theta_i) d\underline{X}^P \tag{2.5}
$$

where the product is over the components X_i of $\underline{X}^P$. For any frame $\underline{\theta}$, a

decision rule $d(\underline{X}^P)$ minimizing R_θ can be obtained by minimizing the integrand in equation (2.5) for each $\underline{X}^P$; thus for a specific $\underline{X}_{ij}$ (an instance of $\underline{X}^P$), an optimal action is:

$$d(\underline{X}_{ij}) = \text{the action (classification) a which minimizes}$$

$$\sum_{\underline{\theta}^P \in \Omega^P} G(\theta^P) L(\theta_p, a) \prod_{i=1}^{P} p(X_i | \theta_i).$$

This can be written in a slightly different form which makes more apparent the specific contribution due to context (the term in brackets below):

$$d(\underline{X}_{ij}) = \text{the action a which minimizes}$$

$$\sum_{\theta' \in \Omega} \left[\sum_{\substack{\underline{\theta}^P \in \Omega^P \\ \theta_p = \theta'}} G(\underline{\theta}^P) \prod_{i=1}^{p-1} p(X_i | \theta_i) \right] L(\theta', a) p(X_p | \theta'). \qquad (2.7)$$

In practice, a "0-1 loss function" is usually assumed, i.e.,

$$L(\theta, a) = \begin{cases} 0, & \text{if } \theta = a \\ 1, & \text{if } \theta \neq a \end{cases}$$

Then (2.7) simplifies and the decision rule becomes:

$$d(\underline{X}_{ij}) = \text{the action a which } \underline{\text{maximizes}}$$

$$\left[\sum_{\substack{\underline{\theta}^P \in \Omega^P \\ \theta_p = a}} G(\underline{\theta}^P) \prod_{i=1}^{p-1} p(X_i | \theta_i) \right] p(X_p | a) \qquad (2.8)$$

Thus (2.8) defines a set of discriminant functions for the classification problem.

The optimal choice of $d(\cdot)$ cannot actually be determined because it depends on $G(\underline{\theta}^P)$ which is unknown. We can, however, expect that, at least for large $N = N_1 \times N_2$, a decision rule in which $G(\underline{\theta}^P)$ is replaced

by an estimate of $\hat{G}(\underline{\theta}^P)$ based on the $\underline{X}_{ij}$ will have risk $\hat{R}_\theta$ approximating that of the optimal rule. (We call this the "bootstrap effect.") That this is the case when p = 1 (approximating an optimal pointwise classifier with estimated a priori probabilities) and suitable forms of estimation are used is a consequence of the work of VanRyzin [9].

The notion of attempting to approximate the risk of the best rule of the form equation (2.3) for p>1, given its first general treatment in Gilliland and Hannan [13], has not been as thoroughly studied as the p = 1 version. But related work for p > 1 in sequence versions of compound decision theory [14] suggests the validity of the generalization. Further, Vardeman [12] points out that if one is willing to separate the N locations into several groups G_1, G_2, ..., G_ℓ within each of which the $\underline{X}_{ij}$ are independent, the results for p = 1 by VanRyzin guarantee that, for p > 1, replacing the $G(\underline{\theta}^P)$ by estimates of the frequencies of $\underline{\theta}^P$ group-by-group produces a decision procedure having the risk of the optimal rule as an approximate upper bound on its risk. An illustration of this separation idea is shown in Figure 2C2.3.

In the interest of a practical solution to the problem of incorporating context into the classification procedure, estimates of $G(\underline{\theta}^P)$ were derived experimentally by simply counting the occurrences of each $\underline{\theta}^P$ obtained in a preliminary classification of the scene without the use of context. Although the use of this rather crude method of estimating $G(\underline{\theta}^P)$ has not been studied in the statistical literature, we will demonstrate in Section 3 its effectiveness for our application.

Before proceeding to a discussion of our experimental results, we make two further observations concerning this approach. First, seeking a criterion for the "context richness" of a scene, we have been able to reach only the following result. Suppose the frequencies $G(\underline{\theta}^P)$ are such that $G(\underline{\theta}^P)$ can be written in factored form, i.e.,

$$G(\underline{\theta}^P) = G_1(\underline{\theta}') \cdot G_2(\underline{\theta}'')$$

where $\underline{\theta}'$ and $\underline{\theta}''$ are, respectively, p - ℓ and ℓ vectors of classes. Then (2.6) can be written in the form
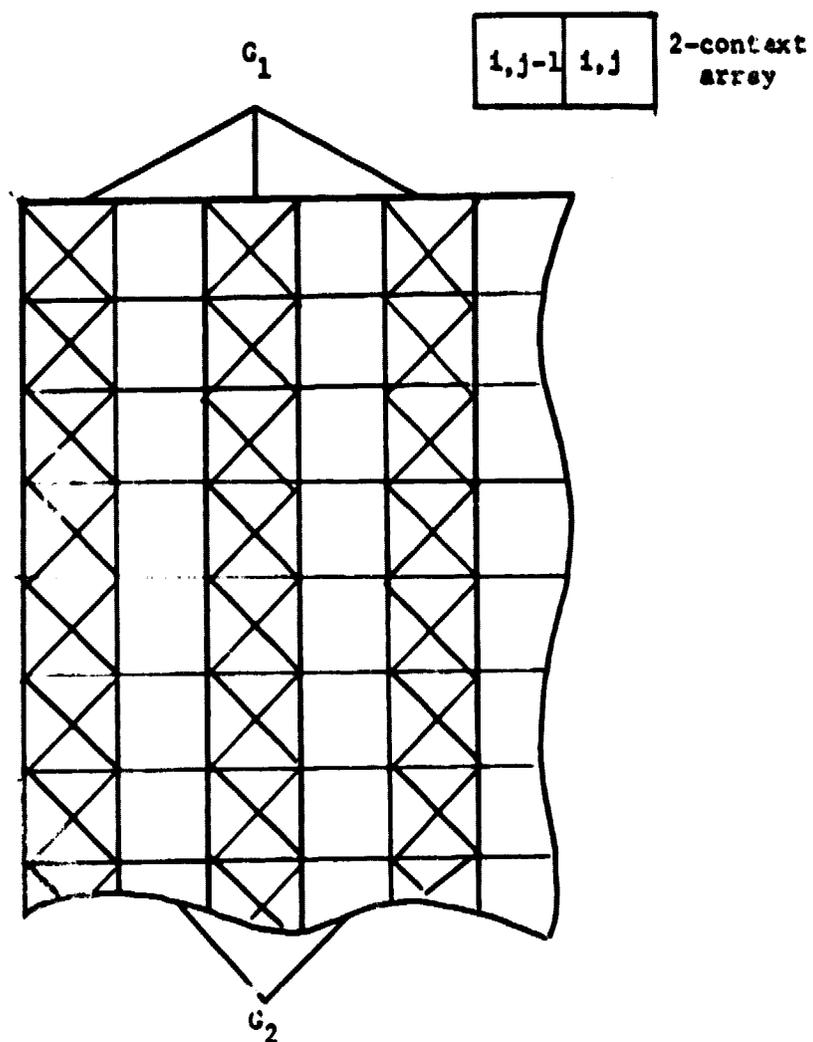
Figure 2C2.3. A 2-context array with separable pixel groups.

$$\sum_{\underline{\theta}''} L(\theta_p,a) \; \underset{\substack{i=\\p-\ell+1}}{\overset{p}{\pi}} \; p(X_i|\theta_i) G_2(\underline{\theta}'') \quad \cdot \quad \sum_{\underline{\theta}'} \overset{p-\ell}{\underset{i=1}{\pi}} p(X_i|\theta_i) G_1(\underline{\theta}').$$

But now the terms included in the second summation are independent of the conditions at the pixel to be classified and are therefore constant relative to the decision to be made. Thus, the decision depends only on ℓ components of the p-context array and is independent of the other $p - \ell$ locations. If it were possible to determine such factorability of the $G(\underline{\theta}^p)$, one could simplify the context classification computations by reducing the size of the context array.

Second, comparing (2.7) with the results of Welch and Salter [5] and reinterpreting the $G(\underline{\theta}^p)$ as the marginal of an _a priori_ distribution for $\underline{\theta}$, one may view (2.7) as a generalization of the Welch and Salter context classification rule. The advantages of the present formulation are that one need make no possibly unrealistic assumptions about the distribution for $\underline{\theta}$ and has complete freedom to choose both p and the form of the p-context array. There are situations (e.g., locating clouds and their associated shadows in a scene) in which context arrays other than those involving neighboring pixels would be useful, a possibility unique to this approach.

3. Experimental Results

Experiments were performed to explore the effectiveness of contextual classification as applied to the analysis of multispectral remote sensing data. First, simulated data were used to determine the degree to which contextual classification might improve the analysis results (as compared to no-context classification), given that the class-conditional densities and the context distribution for the scene were known. The simulated data were used again to investigate candidate methods for estimating the context distribution since, as noted in Section 2, it usually cannot be assumed that the context distribution is known _a priori_. Finally, contextual classification was applied to real data to determine the extent

to which the conclusions drawn from the simulated-data experiments could be extended to the more realistic case.

3.1 Simulated-Data Experiments

A no-context classification of multispectral remote sensing data was selected which had been judged to be very accurate (produced by careful analysis and refinement of multitemporal data). Such a classification could be expected to embody the contextual content of an actual ground scene. Using the classification map and the associated statistics of the classes (developed in performing the no-context classification) data vectors were produced by a Gaussian random number generator and composed into a new data set. Thus the new data set had the following characteristics:

1. Each pixel in the simulated data set represented the same class as in the "template" classification. The template could be considered the "ground truth" for the simulated data set.

2. All classes in the data set were known and represented.

3. All classes had multivariate Gaussian distributions with statistics typical of those found in real data.

4. All pixels were class-conditionally independent of adjacent pixels.

5. There were no mixture pixels.

Data simulated in this manner are somewhat of an idealization of real remote sensing data, but the spatial organization of the simulated data is consistent with a real world scene and the overall characteristics of the data are consistent with the contextual classifier model. In essence, then, the experimental results based on the simulated data demonstrate the effectiveness of the context classifier, given that the underlying assumptions are satisfied. Further experiments with real data are required to generalize the conclusions.

Three data sets were selected representing a variety of ground cover types and textures. Data set 1 was agricultural (Williston, North Dakota), with ground resolution and spectral bands approximating those of the projected Landsat-D Thematic Mapper. Data set 2a was Landsat-1 data from

an urban area (Grand Rapids, Michigan). Data set 2b was from the same Landsat frame as 2a, but from a locale having significantly different spatial organization. Each data set was square, 50 pixels on a side.

Figure 2C2.4 shows the classification results obtained. The "no-context" classification accuracy is plotted coincident with the vertical axis of each graph. Data set 1 was classified using successively 0, 2, 4, 6 and 8 neighboring pixels; data sets 2a and 2b were classified using 0, 2, 4 and 8 neighboring pixels. The accuracy improvement resulting from the use of contextual information was found to be quite significant.

To accomplish the context classification using this approach, it is necessary to have available the class-conditional density functions for the classes to be recognized, $p(X|\omega_i)$, and the context distribution (the frequency distribution associated with the p-vectors, $G(\underline{\theta}^P)$). In remote sensing applications, the class-conditional density functions are typically learned from training samples. For the experiments described above, the Gaussian class statistics on which the data simulations were based were used for the classification (these were originally the training statistics used to produce the "template" classification). An important question is how in practice to determine the context distribution. In the foregoing experiment, this distribution was simply tabulated from the "template" classification (actually, from an area somewhat larger than classified in this test). But in a real data situation, such a template is not available, else there would be no need to perform any further classification.

One can envision a number of ways in which the context distribution might be estimated for a given remote sensing application. For example, it could be extracted from a classification of data obtained previously from the same area. This would require that the area not have changed much in its class make-up since the earlier data were collected and that the earlier classification was reasonably accurate. Alternatively, the distribution might be obtained from a classification of any similarly constituted area. Still another possibility would be to estimate the context distribution for the data to be classified from a "conventional"
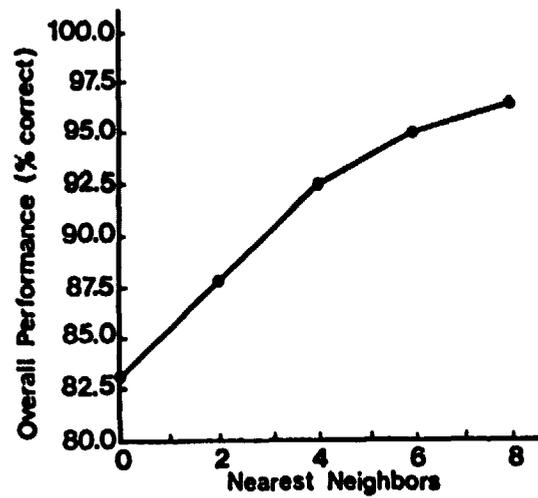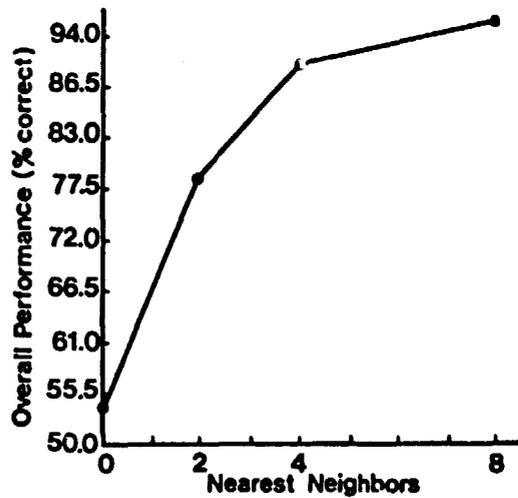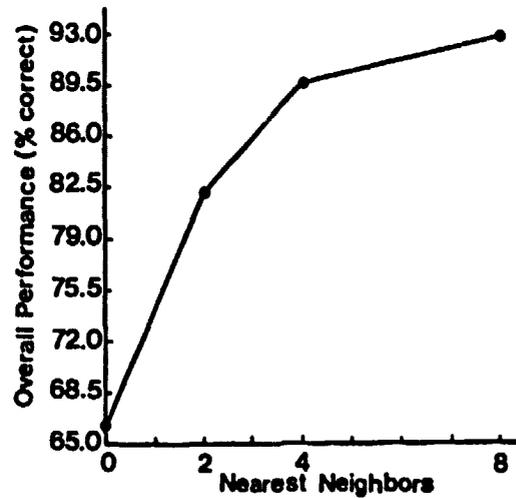
Figure 2C2.4. Contextual classification of simulated data.
(a) Data set 1. (b) Data set 2a. (c) Data set 2b.

classification of the same data determined to have "reasonably good" accuracy. Conceivably, one might then refine the contextual classification by making another estimate of the context distribution based on the resulting more accurate classification, and even iterate in this way until no further improvements in accuracy were obtained. All of these methods produce an estimate of the context distribution, and a crucial question on which hinges the utility of this contextual classification method is how sensitive the contextual algorithm is likely to be to the "goodness" of the estimate.

The iterative technique starting with a no-context classification seemed to be the most practical approach, since no classifications are needed from earlier data or from other areas of similar context. All that is needed is a good initial point-by-point classification of the area in question.

To test the potential of this "bootstrap" technique, it was first tried on the simulated data set 2a. Also, the classifications using the reference template were rerun using an estimate of the context distribution from just the 50-pixel-square area classified, rather than from the larger area (276 x 320) used to obtain the estimate for the results presented in Figure 2C2.4. This was done to provide a better comparison to what could be accomplished using the bootstrap technique.

Using this approach, seven iterations (classifications followed by re-estimation of the context distribution) produced an improvement of 36 percent in overall accuracy compared to the point classification using equal a priori probabilities (from 52 percent to over 88 percent). No significant change was observed in average-by-class accuracy (constant at 68 percent).* This compares with an increase of over 44 percent in over-

---

* Classification performance can be tabulated in two ways. Overall accuracy is simply the overall number of correct classifications divided by the total number attempted. Average-by-class accuracy is obtained by first computing the accuracy for each class and taking the arithmetic average of the class accuracies. The latter is significant when the classification results exhibit a tendency to discriminate in favor of or against a subset of the classes.

all accuracy (over 20 percent in average-by-class accuracy) obtained using the context distribution estimated from the template classification. These results are summarized in Figure 2C2.5.

As seen in Figure 2C2.5, a number of values of p were used in the iteration process. At each iteration, the best classification found by varying p, as judged by trading off overall accuracy against average-by-class accuracy, was used as the template for re-estimating the context distribution for the next iteration.

The best classification on the first iteration was obtained for p = 3 (two nearest neighbors), which was also the case for the second iteration. On the third iteration, the p = 5 (four nearest neighbors) choice was deemed best. Finally, by the seventh iteration, the p = 9 (eight nearest neighbors) choice was considered best. In this case, the overall accuracy was slightly less than for the p = 5 choice (88.2 percent versus 88.6 percent), but the average-by-class accuracy was better by a larger margin (68.1 percent versus 67.4 percent).

This implementation of the bootstrap technique involves a large number of classifications, usually three or more per iteration. A simpler approach would be to do just one classification per iteration and in-crease the number of nearest neighbors used for each iteration. As shown in Figure 2C2.6, for data set 2a the final result using this me-thod was virtually the same as for the more involved procedure.

It was wondered just how much of the accuracy improvement was due to a better estimate of the point-by-point prior probabilities. After five iterations doing 0-nearest-neighbor classification, the improvement in overall accuracy saturated at 80.3 percent, but the average perfor-mance by class had degraded to 46.9 percent. This compares closely to the 0-nearest-neighbor classification done using the context distribution determined from the reference template, which had an overall accuracy of 80.8 percent and an average performance by class of 48.3 percent. It appears from this result that the context serves to improve the overall performance compared to that of the 0-nearest-neighbor result while resisting degradation in average-by-class accuracy.
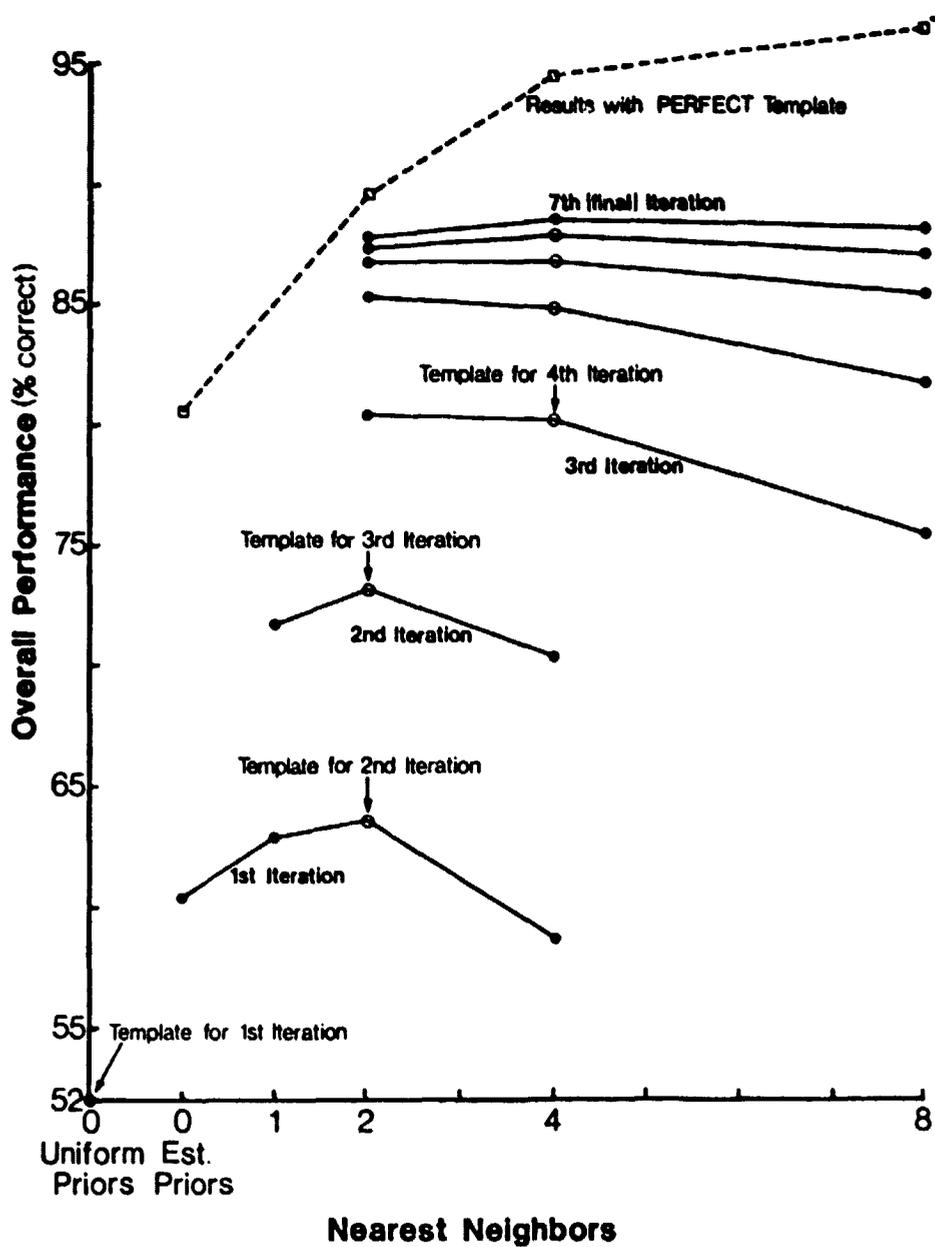
Figure 2C2.5. Results of contextual classification using iteratively estimated context distribution (simulated data set 2a).
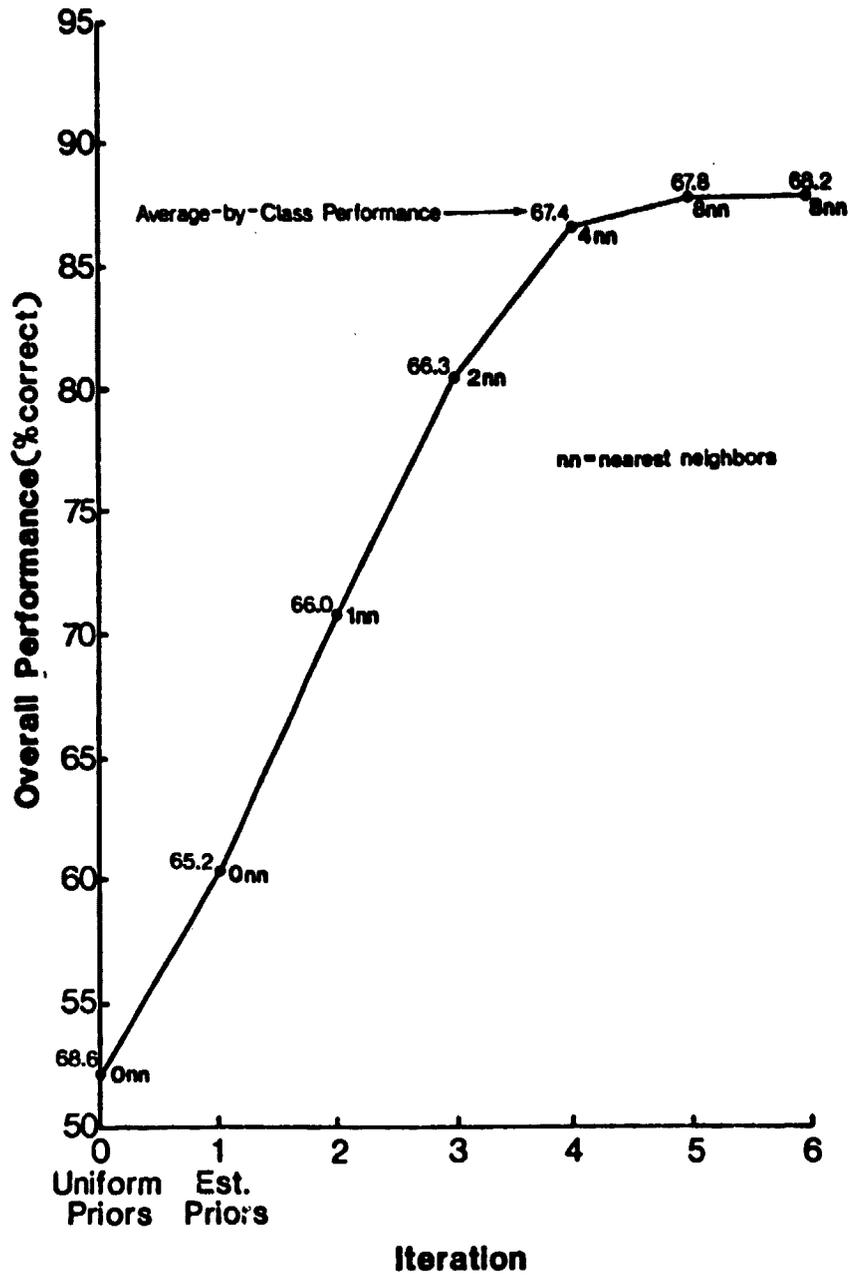
Figure 2C2.6. Contextual classification results based on simplified iterative technique (simulated data set 2a).

3.2 Real-Data Experiments

Having observed excellent performance of the context classifier on
simulated data, the next step was to see how well it would perform on
real data. A 50-pixel-square segment of Landsat data was chosen which
included approximately equal amounts of urban and agricultural area loca-
ted to the southeast of Bloomington, Indiana. Statistics for the spectral
classes were estimated using the 100-pixel-square area centered on the
50-pixel-square segment. A very careful classification using 14 spectral
classes was performed to delineate agricultural, urban and forested areas.
As there were too few forested pixels to delineate forest test areas re-
liably, the classification was tested only for accuracy in classifying
the agricultural and urban classes. Out of the 2500 pixels in the seg-
ment, a total of 867 pixels were manually interpreted as agricultural
and 450 pixels as urban. The identification was made by interpretation
of color infrared photography taken by aircraft on the same day as the
Landsat pass.

The results from using the full bootstrap technique on this data set
were not nearly as favorable as the results obtained from the simulated
data. See Figure 2C2.7.

The no-context classification using uniform prior probabilities had
an overall accuracy of 83.1 percent and an average-by-class accuracy of
82.7 percent. The best classification obtained using this result as a
template to estimate the context distribution was a p = 2 (one-nearest-
neighbor) classification based on the neighbor to the "north" (85.2 per-
cent overall, 84.7 percent average-by-class). Interestingly, the one-
nearest-neighbor result based on the neighbor to the "west" produced a
somewhat poorer classification (84.2 percent overall, 83.8 percent average
by class).* No apparent features in the scene would account for the dif-
ference (i.e., be seen by eye), raising a new issue yet to be pursued.

---

* In the figure, "25 window" refers to one-nearest-neighbor-to-the-north;
"45 window" refers to one-nearest-neighbor-to-the-west.
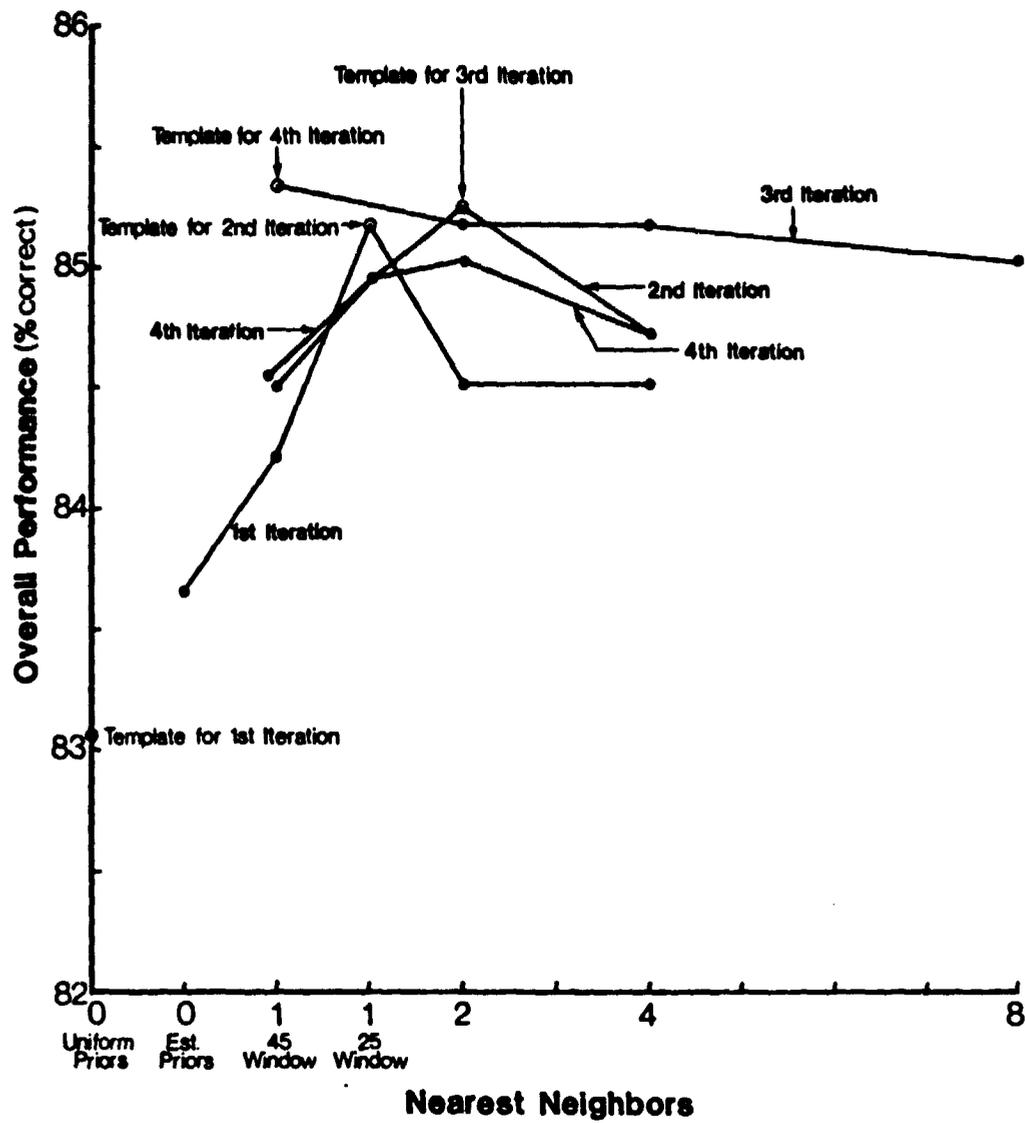
Figure 2C2.7. Contextual classification of Bloomington data using
the unmodified procedure for estimating the context distribution.

The second iteration was performed using the one-nearest-neighbor (north) classification from the first iteration for estimating the context distribution. Here the two-nearest-neighbor (neighbors to the "north" and "west") classification was the best with an overall accuracy of 85.2 percent and average-by-class accuracy of 84.7 percent). The best classification for the third iteration was again the one-nearest-neighbor (north) case with 85.3 percent overall accuracy and 84.8 percent average-by-class accuracy. The fourth iteration produced no improvement. The context classifier thus only yielded just over two percent improvement in both overall accuracy and average-by-class accuracy.

In order to assess the sensitivity of these results to the accuracy of the template used to estimate the context distribution, a manual "cleanup" of the original template was performed, as follows: Change the classification of all incorrectly classified points in the test areas in the original point-by-point uniform priors classification to the closest spectral class in the correct information class as observed by means of a cross-plot of Landsat bands 2 and 3. Where either of two spectral classes might have been the correct class, a coin was tossed to decide the assignment. The context distribution was then estimated from the entire modified classification including both test and non-test areas.

The first iteration using this modified classification as template produced excellent results (Figure 2C2.8). The $p = 9$ (eight-nearest-neighbor) classification produced an improvement of over 10 percent to 93.8 percent in overall accuracy and over 11 percent to 93.6 percent in average-by-class accuracy (compared to the conventional point classifier with uniform prior probabilities). A second iteration was performed using a context distribution estimate from a similarly modified eight-nearest-neighbors classification from the first iteration. No further improvement in accuracy was observed, suggesting that this iterative process "saturates" very quickly.

Finally, both of the techniques applied to the Landsat data from near Bloomington were tried on a 50-pixel-square area from the northwest corner of the Large Area Crop Inventory Experiment (LACIE), segment No.
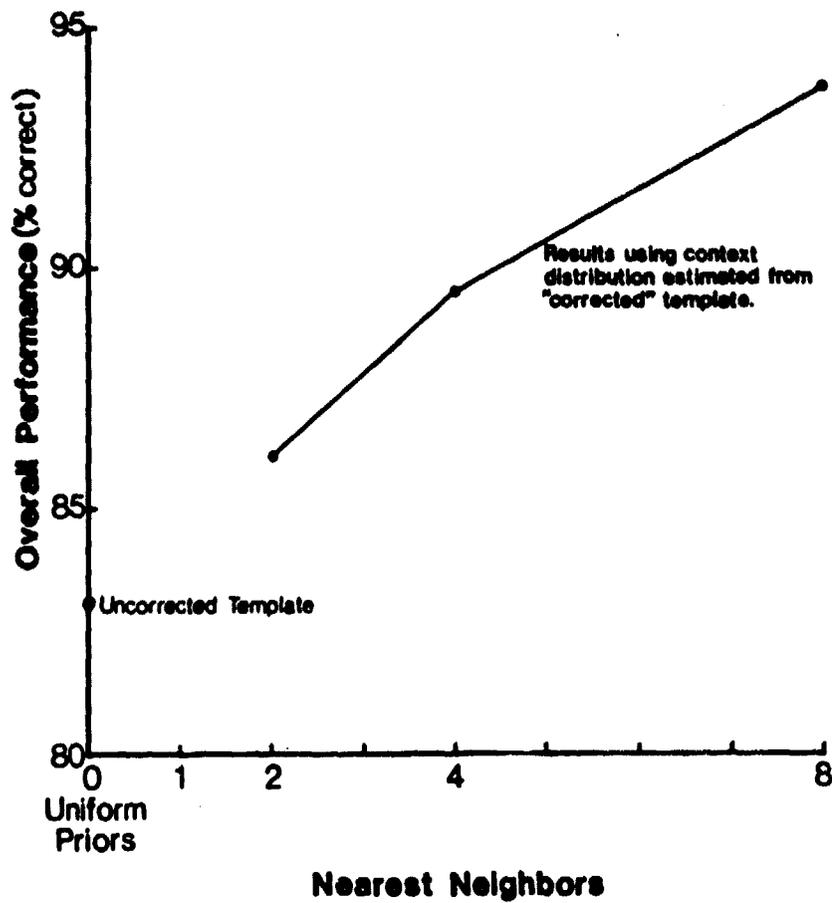
Figure 2C2.8. Performance using manual template correction
for estimating the context distribution (Bloomington data).

1860 in Hodgman County, Kansas. Statistics for the 16 spectral classes were generated from randomly located training fields scattered throughout the entire 117 by 194-pixel Landsat data frame. The coordinates of the training fields were chosen by selecting pixel coordinates from a random number table and surrounding the selected pixel by the largest homogeneous rectangle (up to field size 20 by 20). The classifications were tested for accuracy over five information classes (pasture, idle, wheat, corn and alfalfa) from "wall-to-wall" pixel-by-pixel ground truth.

The results from using the strightforward full bootstrap technique paralleled those from the Bloomington study. Here the no-context classification using uniform prior probabilities had an overall accuracy of 78.7 percent and an average-by-class accuracy of 72.0 percent. As shown in Figure 2C2.9, the best classification (after five iterations) was a $p = 9$ (eight-nearest-neighbors) classification with 80.5 percent overall accuracy and 73.0 average-by-class accuracy. The context classifier could only manage a 1.8 percent improvement in overall accuracy here and a 1.0 percent improvement in average-by-class accuracy.

A manual "clean-up" similar to that done on the Bloomington data was then performed on the first 25 lines of the original no-context uniform priors classification, and the p-vector distribution was estimated from just these 25 lines. Context classifications were performed, and the classification accuracies were evaluated over the remaining 25 lines.

Again, the results employing the manual "clean-up" technique were excellent (see Figure 2C2.10). The overall accuracy over the last 25 lines of the original no-context uniform priors classification was 78.0 percent, while the average-by-class accuracy was 75.6 percent. The $p = 9$ (eight-nearest-neighbor) classification improved the overall accuracy by 9.4 percent and improved the average-by-class accuracy by 6.1 percent.

The excellent results produced by using the context distribution estimated from the manually modified point classification suggest the following approach for classification using context:
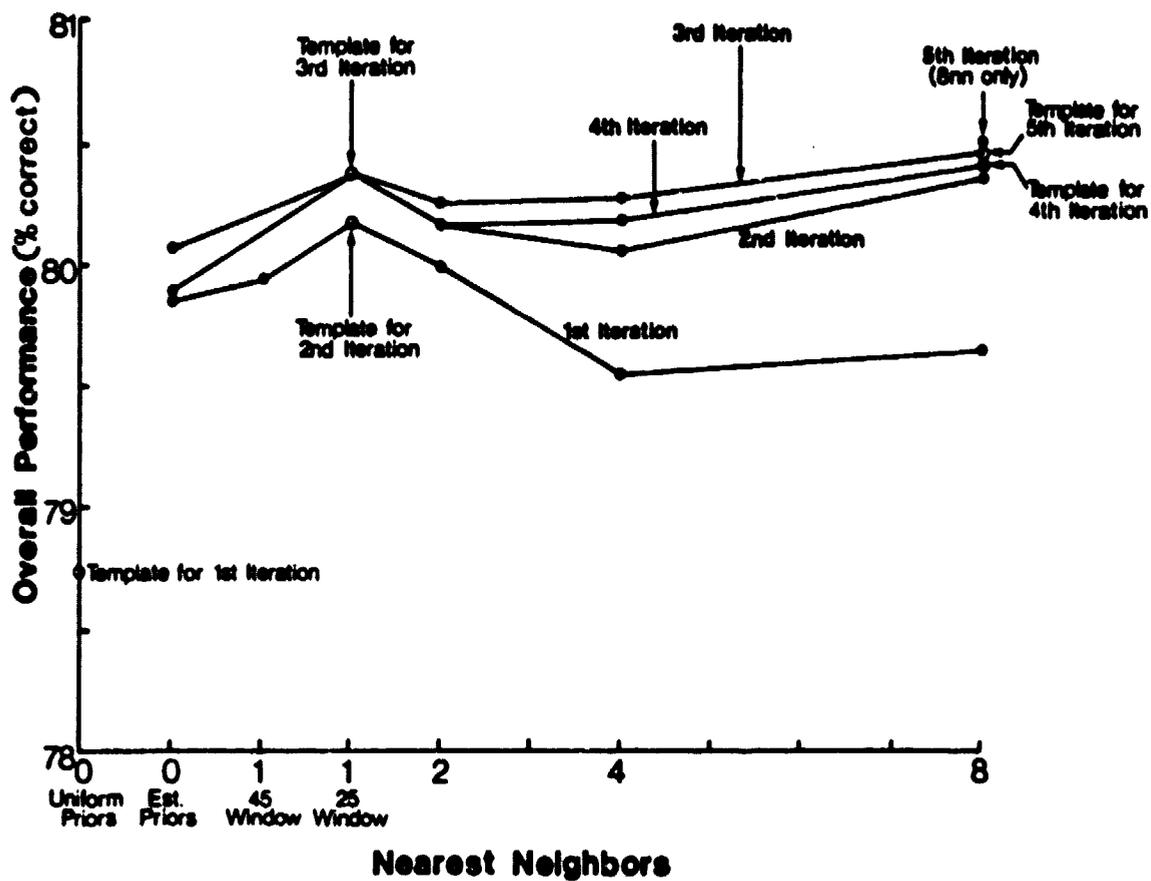
Figure 2C2.9. Contextual classification of LACIE segment using
the unmodified procedure for estimating the context distribution.
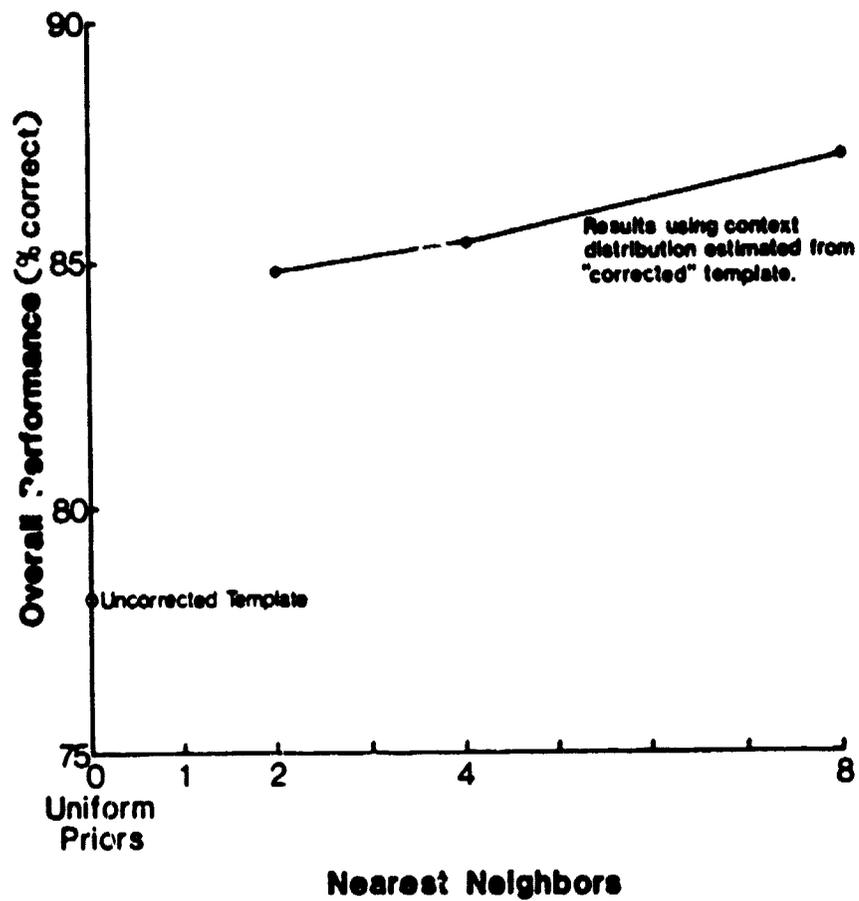
Figure 2C2.10. Performance using manual template correction
for estimating the context distribution (LACIE data).

1. Perform point-by-point classification using uniform prior probabilities on the training set as before, but with the following twist: When a pixel is known to be of a certain information class, allow the classifier to choose only between spectral classes associated with that information class. This will force a 100 percent accurate classification in the training areas and should permit an even better estimate of the context distribution than the manual modification method described above.

2. Estimate the context distribution from the resulting 100 percent accurate classification of the training fields.

3. Classify the entire scene with the statistical context classifier and evaluate the results over a test set disjoint from the training set.

4. Overview of the CDC Flexible Processor Array System

Classification algorithms such as the context classifier (and even much simpler algorithms used for remote sensing data analysis) typically require large amounts of computation time. One way to reduce the execution time of these tasks is through the use of parallelism. Various parallel processing systems that can be used for remote sensing have been built or proposed. These include pipelined processors [16], multimicrocomputer systems [17,18], and special purpose systems [19]. The Control Data Corporation Flexible Processor System [16,20,21] is a commercially available multiprocessor system which has been recommended for use in remote sensing [22].

The remainder of Section 4 consists of a skeleton description of many of the key features of the CDC Flexible Processor System. The description will convey to the potential user (at the programming level) a flavor of the task to be dealt with.

Section 5 contains a description of how the Flexible Processor System can be used to implement contextual classification. As a somewhat simpler problem to start with, implementation of the maximum likelihood classifier is first discussed.

Since our research is being pursued remote from a real Flexible Processor System, we have developed a simulator to facilitate code development and testing. The simulator is described in Section 6.

4.1 The Hardware

4.1.1 Introduction

Key elements of the Flexible Processor hardware are discussed first, focused on the Flexible Processor itself which is the basic building block of the Flexible Processor System.

4.1.2 The CDC Flexible Processor

The basic components of a Flexible Processor (FP) are shown in Figure 2C2.11. Each FP is microprogrammable, allowing parallelism at the instruction level. An example of the way in which N FPs may be configured into a system is shown in Figure 2C2.12. There can be up to 16 FPs linked together, providing much parallelism at the processor level. The clock cycle time of an FP is 125 nsec (nanoseconds). Since 16 FPs can be connected in a parallel and/or pipelined fashion, the effective throughput can be drastically increased, resulting in a potential effective cycle time of less than 10 nsec.

A central feature of the FP is its dual 16-bit internal bus structure, enabling the FP to manipulate either 16- or 32-bit operands. If 32-bit operands are used, the FP can be programmed to execute floating point routines (on its integer hardware) based on the floating point representation of such systems as the IBM 370 and the PDP 11/70. If the needed data width is 16 bits, the FP can be programmed to perform different operations on each of the 16-bit words simultaneously.

4.1.3 Register Files

In each FP, there are two files of registers, one called the temporary register file and the other the large register file. Both are divided into 16-bit addressable subunits. If the needed path width is 16 bits, the two files can act like four files, thus creating more addressable user space. A special feature of the temporary file is its two separate read and two separate write address registers. This can save much CPU time in many types of matrix operations. The large register file has its own two read/write address registers. It is possible to do either a read or write to either file and simultaneously increment (or decrement) the address register. The temporary file is 16 words, 32 bits each, while the large file is 4096 words, 32 bits each. All of the register files consist of 60-nsec random-access memory.

DATA PATHS IN A FLEXIBLE PROCESSOR
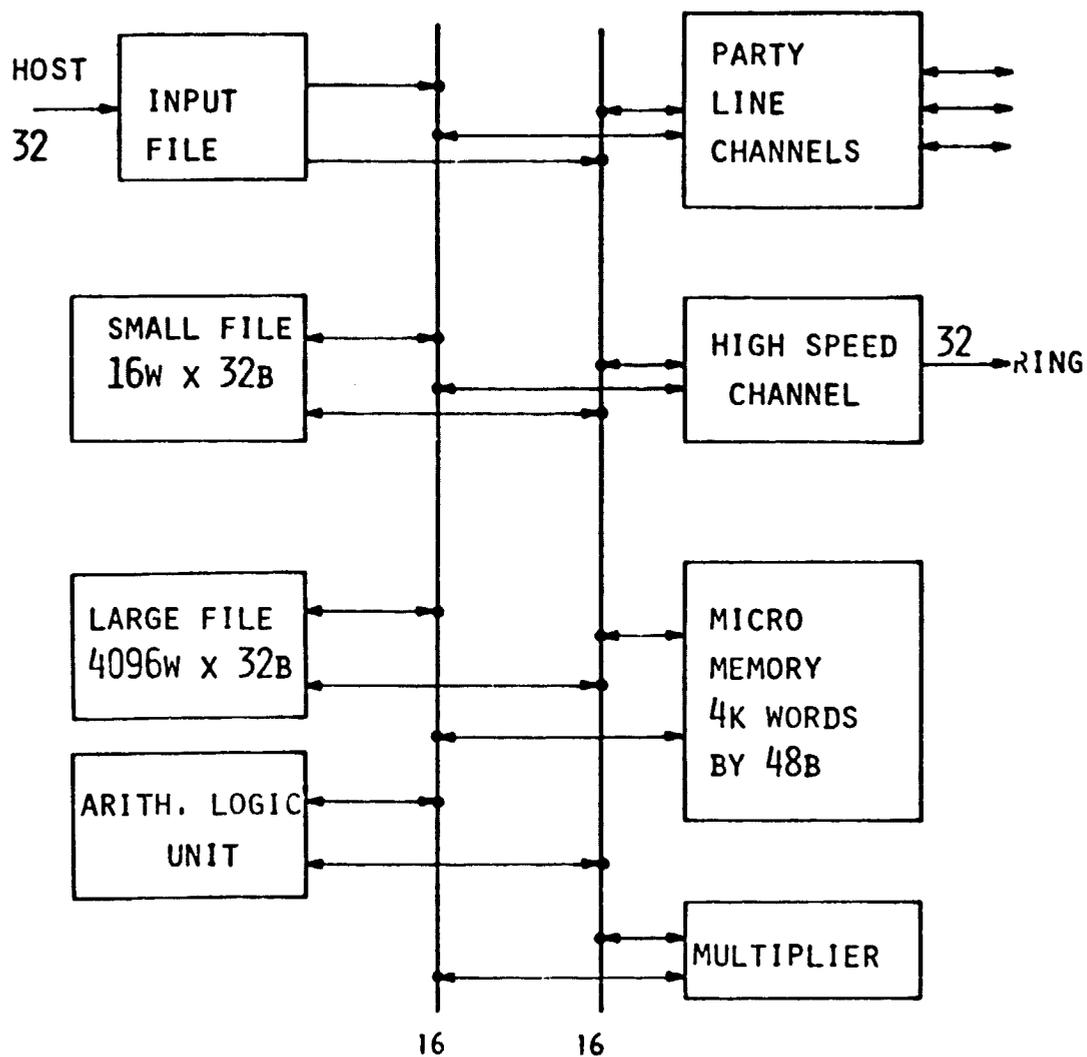
Fig. 2C2.11. Data path organization in the CDC Flexible Processor.

FLEXIBLE PROCESSOR (FP) ARRAY
TYPICAL CONFIGURATION

TO I/O BUFFER MEMORY

TO SYSTEMS
CONTROLLER

TO IMAGE
DISPLAY STATION
CONTROL CONSOLE

FP 0   FP 1   • • •   FP N-1

RING

TO ALL BANKS
OF IMAGE
DISPLAY MEMORY

BULK
MEMORY

BULK
MEMORY
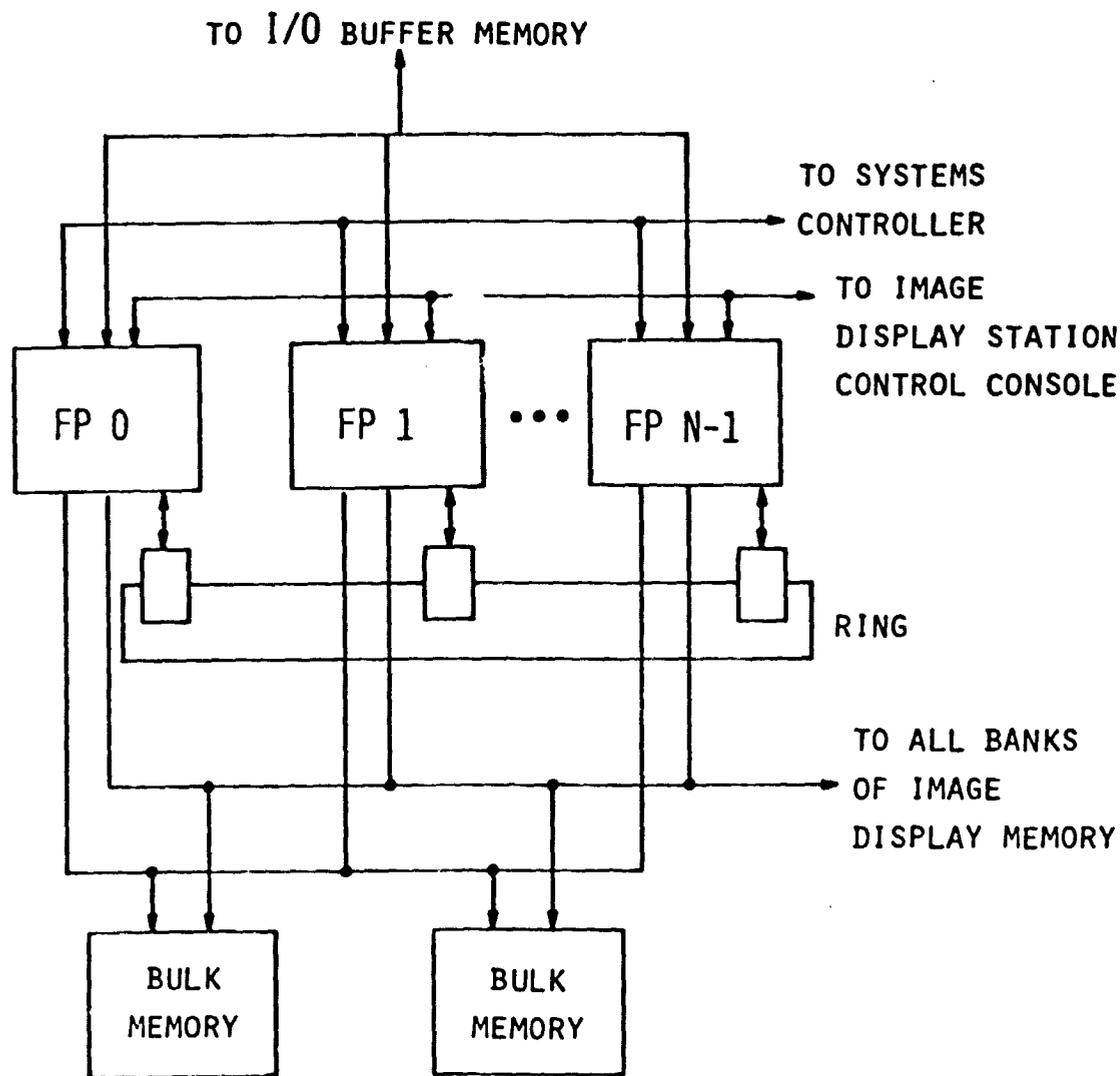
Fig. 2C2.12. Block diagram of typical Flexible Processor array.

4.1.4 Registers and Arithmetic Units

Details of the architecture of an FP are shown in Figure 2C2.13. There are three 32-bit general purpose registers called the E, F, and G registers. All of these registers are connected to the arithmetic logic unit (ALU), which can perform 32-bit additions in 125 nsec. The E and G registers are readable directly through the ALU. The general purpose registers can be shifted separately, or the E and F registers can be combined into a 64-bit shift register for double-length shifts. The output of the ALU is a 32-bit register, A, that is addressable by byte (8 bits). This makes a variety of byte manipulations possible. Separate from the ALU is a hardware integer multiplier, which takes two bytes and multiplies them to produce a 16-bit result in 250 nsec. The input registers are the P and Q registers, which are each 16 bits wide. The user can choose which two bytes are to be multiplied. The FP is equipped with four index registers and eight corresponding compare registers. The index registers can be used for looping and can be incremented or decremented during any statement not addressing those registers. The FP also contains a hardware jump stack, so it is capable of handling standard types of program calls such as subroutine jumps.

4.1.5 Micro-Memory and Input/Output

The micro-memory consists of 4k 48-bit words. It stores the micro-program. Each FP in a system can contain a different program.

Input/Output (I/O) for the FP depends on the overall system (i.e., the FP array and its host machine). An FP is capable of interrupting another FP for I/O. I/O among the FPs is done one of two ways. The first is a very high speed communication link, arranged in a ring configuration [20,21]. It operates at four mega-words (16 bits per word) per second. Each FP has a station on the ring, and each station on the ring is connected to two other stations. When an FP does a write to the ring, it gives 16 bits of data and the address of the destination.   If a station receives data for another address, it shifts the data to the next station. This is continued until the data reach the correct station. Special hard-

Figure 2C2.13. Details of the Architecture of a F1

idx0 | idx1 | idx2 | idx3 — index registers

icr0 | icr1 | icr2 | icr3 — index compare registers

mcr0 | mcr1 | mcr2 | mcr3 — maintenance compare registers

cmr0 | cmr1 | cmr2 | cmr3 — condition mask registers

brg1 | brg0

p | q

lf1a | lf0a

multiplier

mult

large file one | large file zero

4096

g1 | g0 | e1 | e0 | f1 | f0

alu

a1 | a0

b3 | b2 | b1 | b0
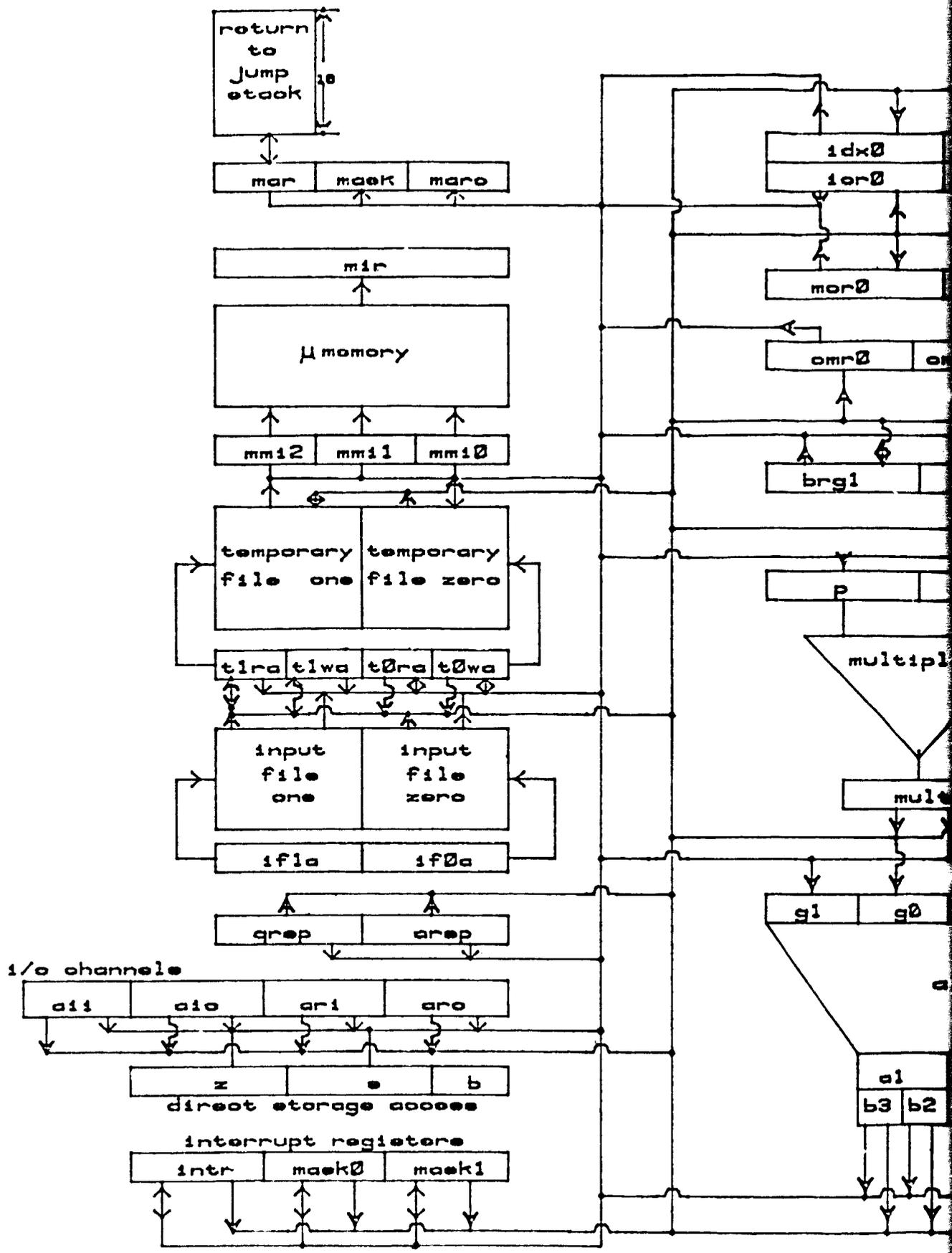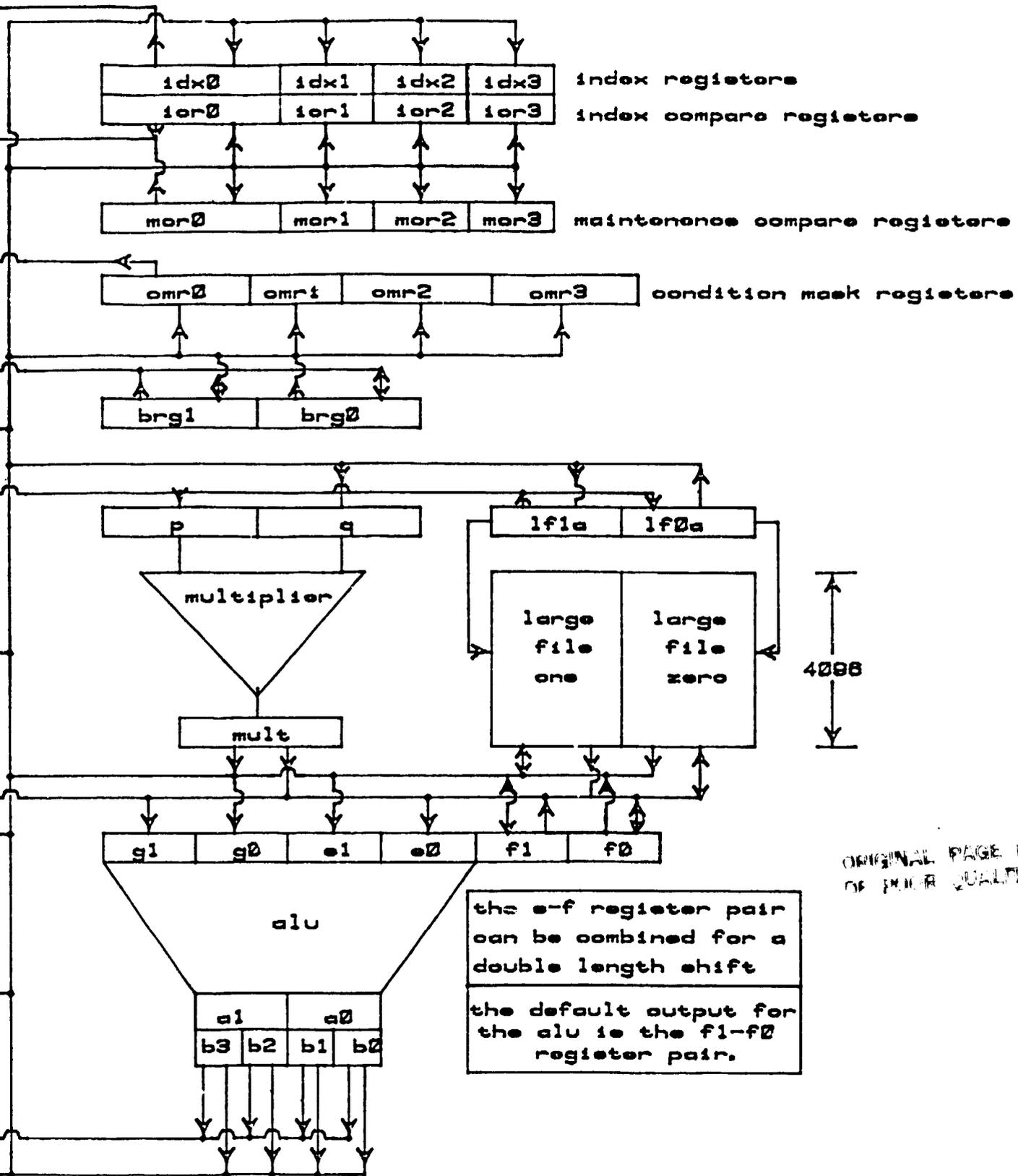
the e-f register pair
can be combined for a
double length shift

the default output for
the alu is the f1-f0
register pair.

of the Architecture of a Flexible Processor.

ware has been added to remove data from the ring in the event of a station failure. The data are loaded into the "input file." This 16 32-bit/word register file can be used as a small buffer. Another form of I/O is through up to 16 64k-byte banks of shared 160-nsec memory. This is not as fast as the previous method; however, for large data transfers, it frees the ring for other communications, as well as providing a buffer between FPs.

4.1.6 Microprogramming of the Flexible Processor

The FP is micro-programmed in "micro-assembly language," allowing parallelism at the instruction level, as indicated in the FP coding form shown in Figure 2C2.14. For example, it is possible to conditionally increment an index register, do a program jump, multiply two 8-bit integers, and add the E and G registers, all simultaneously. This type of operational overlap, in conjunction with the multiprocessing capability of the FPs, greatly increases the speed of the FP array.

4.1.7 A Flexible Processor Image Processing System

Figure 2C2.15 is a schematic block diagram of the system in operation at the CDC Digital Systems Display Laboratory in Minneapolis [22]. This figure is provided as an example of one possible FP array configuration. The setup of this system has many desirable features for picture processing. The parallel-pipelined architecture of the FPs enables the system to do rapid matrix multiplications. There are image displays attached, so it is possible to view the pictures. The two 800-bpi tape drives, along with the 50M disk unit, contain enough storage space for jobs that require large amounts of memory. In addition, the system can handle up to eight terminals on its resident operating system (called ICE). Batch jobs can also be run from its 300-card-per-minute reader.

Figure 2C2.14. Flexible Processor Coding Form [21].
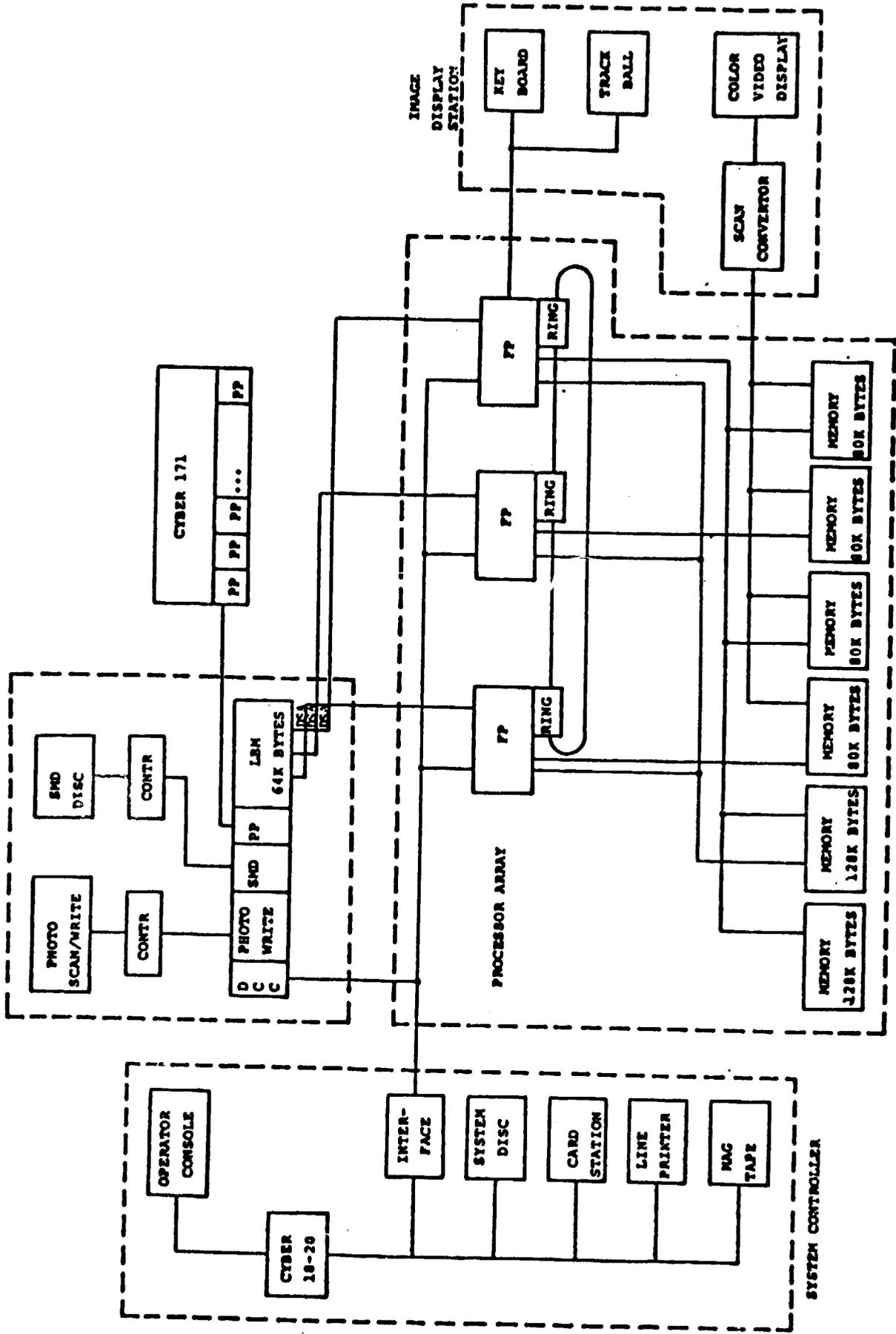
Figure 2C2.15.   A Flexible Processor Image Processing System [22].

4.2 The Software

4.2.1 Introduction

The host for the FP system is programmable in FORTRAN. FP programs written in assembly language can be called from the FORTRAN library, enabling the calling programs to be written in FORTRAN [22]. The average user, then, will not have any contact with the FP assembly language, making the use of the system much easier. Data analysis packages, such as parts of LARSYS, which are written in FORTRAN, can with very simple modifications run on the FP system. The rest of this section overviews how to program an FP at the micro-assembly language level.

4.2.2 Registers

The three general purpose registers (E, F, and G) are divided in halves because they are 32 bits long and the busses are only 16. The most significant bits of the registers are referred to as the "one" group and the least significant bits are referred to as the "zero" group. For example, the most significant bits of the E register are called E1, and the least significant bits of the E register are called E0.

The ability to address registers in groups of 16 bits allows one to address halves of two separate registers simultaneously. For example, if one wished to write into the upper 16 bits of the F register and the lower 16 bits of the G register, the pair would be referred to as F1G0 in the command. Both will get the same data, but they will get it in one machine cycle instead of two. This increases throughput when, for example, loading initial conditions.

4.2.3 The Transfer Constant Instruction

These registers can be loaded with a constant using the Transfer Constant (TC) instruction. Figure 2C2.14 shows the coding form. Line three gives the form of the TC instruction format. Omitting the AAAA and the comments, the basic form of the instruction is:

```
                 TC        $HHHH DSTO DST1
```

The $ tells the assembler that the four following digits are to be in
hexadecimal. This command places the constant on both data lines to enable
the loading of two registers simultaneously. The DST (destination) is
filled in by an appropriate register which can read off the corresponding
bus. Not all registers can provide data to ("source") or receive data
from ("destine") both busses. For example, F1 cannot read ("destine") bus 0,
the E and G registers can only be sourced into the arithmetic logic unit,
and the E1 and G0 registers can only read from bus 1 [21].

Some examples of correct TC instructions are:

```
                 TC        $FFA8 E0G1 F1G0,
                 TC        $0100 E0   G0  ,
                 TC        $0101 E0   NOP .
```

The first command in the example transfers the hexadecimal constant FFA8
to the 16-bit registers E0, F1, G0, and G1. The second command transfers
the hex constant 0100 to the E0 and G0 registers. In the third command,
the NOP indicates bus 1 is not used. Note that while it is not possible
to source two different registers at the same time, it is possible to
destine two registers off the same bus at the same time.

4.2.4 The Transfer Register Instruction

    Another way in which the registers can be used as a source of infor-
mation is in the Transfer Register (TR) instruction. This is the fourth
format shown in Figure 2C2.14. The basic format of the instruction is:

```
                 TR    SRC0      DST0      SRC1      DST1
```

This instruction tells the computer to source the register in the SRC0
field to bus 0 and to use the register(s) in the DST0 field as the
destination(s). In the event that the other bus is not to be used, a NOP
must be placed in both the SRC and DST fields corresponding to that bus.

4.2.5 Using the Temporary Files

    A special feature of the temporary register files, discussed in
Section 4.1.3, is that it has separate read and write indices. The in-

dices are TORA, TOWA, T1RA, AND T1WA, which stand, respectively, for Tempo-
rary file 0 Read Address, Temporary file 0 Write Address, Temporary file
1 Read Address, and Temporary file 1 Write Address. Each is four bits in
length. When using the temporary files, one usually initializes the index
value and then uses special instructions to increment, decrement, or clear
these registers while doing other operations. When storing information to
a temporary file, the mnemonic used is TFxf, where x is the file number
and f is the function to be performed. The following is a list of the
available functions:

> U   increment the corresponding index
>
> D   decrement the corresponding index
>
> C   zero the corresponding index
>
> N   perform no operation on the index

The machine will update the read or write address, depending on the con-
text used, i.e., if a temporary file is used as a source, the read address
will be assumed, and if it is used as a destination, the write address will
be assumed. Some examples are as follows:

> TC $0101   TF0U TF1D
>
> TC $0101   TF0N TF1C
>
> TC $0101   TF0C TF1C

In the examples, the hex constant 0101 is stored in the temporary file
while the write pointer is incremented, decremented, unchanged, and cleared.

4.2.6 Using the Large Files

The Large files, discussed in Section 4.1.3, have only one pointer
per file, but are accessed in the same manner as the temporary file. To
access a file, the format is LFxf, where x is the file number and f is
the function to be performed on the file. The functions performed are
the C, D, and N as defined in Section 4.2.5 and A which adds index
register 0 to the corresponding index and uses that location as the de-
sired address. The instruction

> TC $0101 LF0U LF1D

would store the hex constant 0101 in large files 0 and 1 while increment-
ing the pointer for large file 0 and decrementing the pointer for large
file 1. The length of the large file pointers is 10 bits. Large file

pointers are called LOA and L1A. Both the large file and the temporary
file pointers can be accessed in the same manner as standard general pur-
pose registers.

4.2.7 Programming the Arithmetic Logic Unit

In the TR instruction there is a field labeled ADD (see Figure 2C2.14).
This field controls the function of the ALU. Output from the ALU is avail-
able as the A (accumulator) register, which can be sourced in the same
manner as the F and G registers. In the event that the A register is not
sourced, the result is moved to the F0-F1 register pair. One feature of
the A register is different from the other general purpose registers in
that it is byte addressable. This ability makes it one of the most power-
ful registers on the machine. Figure 2C2.16 is a listing of the ALU mnemonics
and a brief interpretation of their meanings. It is important to remember
that this machine is micro-codable; thus there are many possibilities that
are not in the mnemonic set. This is the extent of the assembler mnemonics
for the ALU, but there are more commands. Figure 2C2.17 shows a listing of
the entire command set. To be able to use this list, first type either an A
or an L (for arithmetic or logical) and then a C or an N (for carry or
no carry). The A(L) determines the basic function type.   The N further
determines the type of function by determining the type of carry. With the
above, it is possible to use Figure 2C2.17 to determine the exact function
number desired. The only other entity necessary is the function number
(from 0 to F). Thus an ANF describes the arithmetic function in the no-carry
portion of the table that is in the fifteenth row. All three of the function
descriptors are placed in the column labeled ADD (see Figure 2C2.14).

As shown in Figure 2C2.13, the A register is divided into four bytes
numbered zero through three. If AO is sourced, bytes 0 and 1 will be ob-
tained. Likewise, sourcing A1 will yield bytes 2 and 3. If bytes 1 and 2
are needed together, adding an SW (which stands for SWap bytes) to the end
of AO will yield the desired result. If bytes 0 and 3 are needed, adding
an SW to the end of A1 will yield the desired result. Thus AOSW is the
correct way to address bytes 1 and 2.

Mnemonic:	Function:	Comments:
ADD	A=E+G	Twos complement add the E and G regs.
AND	A=EG	Logical AND the E and G registers.
E	A=E	This is the method for sourcing the E register, making it sourcable to both busses.
E+1	A=E+1	This makes it possible to increment,
E-1	A=E-1	decrement, and double the E register
E+E	A=E+E	without ever having to load a constant.
E-G	A=E-G	Twos complement subtract the E and G register pairs.
E=G	A=E-G-1	The Flexible processor has a branch if negative command. If the E register is less than or equal to the G register, this will branch.
EN	A=*E	Logically complement the E register (E NOT).
G	A=G	This makes the G register sourcable to both busses.
GN	A=*G	Logically complement the G register.
OR	A=E+G	Logically OR the E and G registers.
SB1	A=E-G	Ones Complement subtract the G register from the E register.
SET	A=E+*E	Set A to all ones.
XOR	A=E+G	EXCLUSIVE OR E and G registers.
ZRO	A=E*E	Load A register with all zeros.

Figure 2C2.16.   Flexible Processor Arithmetic Logic Unit Mnemonics.

Function Number	Logical Functions	Arithmetic Operations No Carry	With Carry
0	F = *E	F = E	F = E+1
1	F = *[E+G]	F = [E+G]	F = [E+G]+1
2	F = [*E G]	F = [E+*G]	F = [E+'G]+1
3	F = [*F F]	F = -1(2's comp)	F = 0
4	F = *[EG]	F = E+[E*G]	F = E+[E*G]+1
5	F = *[G]	F = [E+G]+[E*G]	F = [E+G+E*G]+1
6	F = [E*G+*EG]	F = E-G-1	F = E-G
7	F = [E*G]	F = [E*G]-1	F = [E*G]
8	F = [*E+G]	F = E+[EG]	F = E+[EG]+1
9	F = [*E'G+EG]	F = E+G	F = E+G+1
A	F = G	F = [E+*G]+EG	F = [+EG]+1
B	F = [EG]	F = [EG]-1	F = [EG]
C	F = [F+*F]	F = E+*E	F = E+E+1
D	F = [E+'G]	F = [E+G]+E	F = [E+G]+E+1
E	F = E+G	F = [E+*G]+E	F = [E+*G]+E+1
F	F = E	F = E-1	F = E

[] - contains only logical operations.

Figure 2C2.17. Entire Command Set of Flexible Processor
Arithmetic Logic Unit.

Another feature of the A0 and A1 registers is that they can do a right shift, preserving the signs of the registers. This is accomplished by catenating an RS (Right Shift) at the end of the desired register. It is possible to do a right shift in conjunction with a byte swap. The ALU has the ability to shift a byte of zeroes into either (or both) of the A0 and A1 registers. This is accomplished by shifting both accumulators right by one byte, and loading the upper byte of the pair with zeroes. The mnemonic for this is a RZ (Right shift Zero fill) catenated at the end of the byte pair desired. The following is a list of the possible combinations of the accumulator and the above operations [21]. The bus numbers are omitted because they can be sourced to either bus. Shift is done before swap. B0, B1, B2, and B3 indicate the four bytes of the A register.

Source A Field	Source B Field	Bus A			Bus B	
A0	A1	B1	B0		B3	B2
A0	A1RS			Illegal		
A0	A1RZ			Illegal		
A0	A1SW			Illegal		
A0RS	A1			Illegal		
A0RS	A1RS	LS	B1		US	B3
A0RS	A1RZ	Z	B1		US	B3
A0RS	A1SW	B2	B1		US	B3
A0RZ	A1			Illegal		
A0RZ	A1RS	LS	B1		Z	B3
A0RZ	A1RZ	Z	B1		Z	B3
A0RZ	A1SW	B2	B1		Z	B3
A0SW	A1			Illegal		
A0SW	A1RS	LS	B1		B0	B3
A0SW	A1RZ	Z	B1		B0	B3
A0SW	A1SW	B2	B1		B0	B3

Z — one byte of zeroes
LS — sign of lower two bytes
US — sign of upper two bytes

4.2.8 The Index Registers

In the diagram of the machine structure (Figure 2C2.13, there are four index registers, four index compare registers, and four compare mask registers. None of the registers can be sourced for their contents alone. Index register 0 and its corresponding compare register are 16 bits long, while all the others are only 8 bits long. The IDX field, shown in Figure 2C2.14, is the field that controls the operation of the indices and their compare registers. An INx command, where x is one of the index registers, will increment index register x. A DCx will decrement index register x by one, while a CLx will clear index register x. CLA will clear all registers.

4.2.9 Conditional Operations

The condition mask registers control the condition to be used. These registers do not have a one-to-one correspondence to the index registers. The following is a list of the functions used in the current software (a full listing appears in [21]. The lengths of the registers are shown in Figure 2C2.13.

Bit	Condition Mask Reg 0	Condition Mask Reg 3
0	E0 negative	Index Compare reg0 = index 0
1	E1 negative	Index Compare reg0 = index 0
2	F0 negative	Index Compare reg1 = index 1
3	F1 negative	Index Compare reg1 = index 1
4	G0 negative	Index Compare reg2 = index 2
5	G1 negative	Index Compare reg2 = index 2
6	ALU0 negative	Index Compare reg3 = index 3
7	ALU1 negative	Index Compare reg2 = index 3

It is possible to test for the conditions in Mask Register 0 by placing a TN in the CND (CoNDition) column. Figure 2C2.14 shows the location of the CND column in the coding form. To test for the logical "not" of the condition stored in Mask Register 0, an FN is placed in the CND column. To test for the condition in Mask Register 3, an AD is placed in the

CND column. Furthermore, the AD must be placed at least two instructions after an increment or decrement of the register in question. If the condition tested is true, the current instruction is executed.

The ability to conditionally execute a statement enables a conditional program jump. Recall that the basic form for a TC statement is:

TC $HHHH DST0 DST1

If DST0 is the MAR (memory address register), then after execution of the next statement, the FP will do a conditional jump to the value indicated by the hex constant, which can be a program label. The following is an example of a conditional jump which will jump to hex address 1234:

TC   AD $1234 MAR   NOP

To do an unconditional program jump, omit the AD. The following:

TC   NEXT MAR   NOP

will jump to the program label NEXT. Since the MAR and instruction fetch of the FP are buffered, it is impossible to do an immediate program jump. This adds little complication to the programming, except that the step to be executed before the jump is placed after the actual jump statement. It is very important, when reading source code for the machine, to remember that the order of execution is reversed.

4.2.10 Subroutine Calls, Program Jumps, and the Stack

As shown in Figure 2C2.13 there is a 16-by-12-bit stack called the return jump stack. This is a typical LIFO buffer which is used to hold return addresses as well as temporary data. As indicated in Figure 2C2.14, there is a field labeled RJ. This controls the return jump stack. There are three possible commands for the stack. SR (SubRoutine jump) will take the current value of the MAR (which is pointing to the next statement), increment it by one and store the result on the top of the stack. This will be the return address. JP (JumP return) takes the current top of stack and places it in the MAR. DF (Delete First item) will delete the top of the stack. The JP does not perform the delete function. Another feature of the SR, JP, and DF is that they all trap out interrupts. A typical subroutine jump looks like the following:

(Fields)	Type	RJ	$HHHH	DST0	DST1
Label	TC	SR	$1234	MAR	NOP
	TC		NOP	NOP	NOP

The above routine will store label+2 on the stack, execute the NOPs, and jump to the hexadecimal location 1234. A typical subroutine return looks like the following:

(Fields)	Type	RJ	$HHHH	DST0	DST1
	TC	JP	NOP	NOP	NOP
	TC	DF	NOP	NOP	NOP

This will take the top of stack, place it in the MAR, and then delete the top of stack. Since the CND field is valid on all types of instructions, it is possible to do a conditional subroutine jump just by placing the condition in the conditional field. The result looks like:

(Fields)	Type	CND	RJ	$HHHH	DST0	DST1
	TC	AD	SR	$1234	MAR	NOP

This will store the value of the return address, execute the next statement, and continue execution at location 1234. By placing a JP in the next statement, it is possible to do a jump, execute one statement and return.

4.2.11 The Hardware Multiply

The only remaining functional unit to be discussed is the hardware multiply. As shown in Figure 2C2.13, the inputs are the P and Q registers which are each 16 bits in length. The result of the multiply is a 16-bit product, which can be the result of the multiplication of any two bytes. This is the only case where the same byte can be sourced twice. The mnemonics for the addressing is L for the lower byte, and U for the upper byte. Thus, to multiply the lower byte of the P register by the upper byte of the Q register, a PLQU would be placed in the MULT field. Caution must be taken when a multiply is initiated. A multiply takes two machine cycles before the result can be sourced. If an interrupt is received before the result is ready, the result will be lost. To prevent such loss, it is necessary to trap out all interrupts. This is accomplished as

55

follows: Whenever a multiply is done, an SR is placed in the RJ column
of the first statement of the multiply, and a DF is placed in the RJ
column. The net result is to push a return address onto the stack and
then pop it off the stack. This will trap out interrupts as needed. Fur-
ther, caution must be taken in that the RJ stack is only 16 units long, so
overflow is possible. If overflow occurs, no error will be flagged. The
following is a routine to square the lower byte of the Q register.

(Fields)	TC	RJ	MULT	$HHHH		DST0		DST1
	TR	RJ	MULT		SRC0	DST0	SRC1	DST1
	TC	SR	QLQL	$0057		MAR		NOP
	TR	DF	QLQL		MULT	F0	MULT	F1

This not only does a multiply, but it also does a program jump and traps
interrupts all at the same time, showing how this machine obtains very
high processing speeds. (Consider that each program step takes .125 micro-
seconds). If more precision is desired, the following algebraic rule
can be used:

$$(a+b)*(c+d)=ac+ad+bc+bd.$$

This rule can be modified to the byte level, yielding the 32-bit result
in under three microseconds [26].

4.2.12 Bus Registers

The two registers in Figure 2C2.13 labeled BRG0 and BRG1 are the bus
registers. Normally these are used for breakpointing. It is possible
to use these registers for general purpose registers if no breakpointing
is needed. To write into these registers, BRG0 and BRG1 are put into the
respective columns, while to read from these registers, BSR0 and BSR1 are
put into their respective columns.

4.2.13 Shifting Data

The SH instruction field is used for shifting data as shown in
Figure 2C2.14, the OEINC, OFINC and OGINC fields all determine what type of
shift is to take place. The P field determines the Precision of the
shift. If the P field is set to S, all of the registers are treated as

separate registers; however, if the P field is set to D (Double Precision), the E and the F registers are tied together as one register for the shift. The following list shows all possible shift conditions and their specific operations. These conditions do not determine whether to execute the instruction, but on what version of the ALU's data [21].

CONDITION FIELD	ALU DATA USED FOR SHIFT	COMMENTS
UN	TRUE OF CURRENT DATA	(Unchanged Now)
TN	TRUE OF CURRENT DATA	(True Now)
FN	NOT OF CURRENT DATA	(False Now)
AD	NOT OF CURRENT DATA	(Conditional based on AD condition)
UP	TRUE OF PAST DATA	(Unconditional Past)
TP	TRUE OF PAST DATA	(True of Past data)
FP	NOT OF PAST DATA	(False of Past data)
IO	NOT OF PAST DATA	(Conditional based on IO condition)

These commands not only determine the data to be shifted, but they also control the conditions under which the shifts are done. When these mnemonics are placed in the CND field, they are used to check the conditions set in the condition field zero.

4.2.14 Input/Output to the FPs

Input/Output (I/O) is one of the most complicated parts of the entire CDC FP System. I/O must occur in one of the following forms:

1. FP to host
2. FP to FP
3. FP to MOS RAM (shared bulk memory)

For large amounts of data requiring FP-to-FP communication, FP to MOS RAM is the most reasonable means of data transfer. If the high-speed communication link, as described in Section 4.1.5, is used, there is only a buffer for 16 words of information. This requires very closely timed algorithms, as any error would result in the loss of data. Each FP is connected to four 16-bit channels, which are called Direct Storage Access (DSA) Channels.

Each of the channels is connected to four banks of 600 nsec MOS RAM. Each bank of MOS RAM is addressed by bank and channel. Different banks on various channels may be shared. For example, bank 1 on channel 3 may be the same as bank 2 on channel 1. The FP is capable of choosing a bank and address to which all the channels are linked through four S (Storage location) registers and B (Bank) registers. Since the RAM memory is much slower than the clock cycle, the read is done in two stages. The first stage sends the bank and address to the MAR and increments the data in the address register, initiating the read. Within the next four cycles, the data will appear in the Zx register, where x is the channel number (see Figure 2C2.13). The data will remain in the Zx register until the next read is initiated. In the event of a "memory bank busy," or "data not ready," the FP will automatically wait for two machine cycles, after which it will repeat the process. To do a write, the data is sourced directly to the MBR (Memory Buffer Register) of the memory bank corresponding to the bank register. (A write is a 1-stage process.) The FP is programmed to do I/O through the IO statement type. Figure 2C2.14 shows the form of the statement. The IO statement is similar to the TR statement in that arithmetic calculations can be done simultaneously with I/O. The following statements show how to initialize the S and B registers. (The S and B registers are linked together so that they can be loaded in one statement.)

IO	CND	IDX	RJ	MULT	ADD	SRC0	SRC1	IO	CH0	CH1	CH2	CH3
IO					ZRO	A0	A1	DS	LS	LS	LS	LS
IO						FO	FO	DS	LB	LB	LB	LB
IO			DF	PLQL		MULT	MULT	DS	LSB	LSB	LSB	LSB

1. Loads all four S registers with 0000.
2. Loads all four B registers with the contents of FO.
3. Loads all four S and B registers with the contents of the multiplier.

The DS stands for DSA I/O. The leading L in the channel column stands for load.

After initializing the S and B registers, the read needs to be initialized, which is done by placing an R in the channel field of the

channel to be read. Four cycles later, the data (or a wait) should appear in the Zx register. To do a write, a W is placed in the channel fields into which the data are to be written. The data to be sourced are in the source fields.

4.2.15 Interrupts

With I/O, interrupts are often needed. The FP has the ability to handle up to 16 different interrupts [20,21]. The FP can interrupt itself, the host and other FPs. While processing an interrupt routine, the FP sets a flip-flop indicating that an interrupt is being processed. This traps all lower priority interrupts. The interrupt flip-flops are reset when the program returns to processing the original routine, or until a zero is stored in the interrupt register.

4.2.16 Conclusions

This has been an introduction to the parts of the FP and the parts of the instruction set that will be used in the Bayes maximum likelihood classifier discussed in the next section. For further documentation, consult the CDC Flexible Processor Textbook [21].

The experience gained through the use of the simulator (see Section 6) has made evident the following advantages and disadvantages of the Flexible Processor system.

Advantages:
Multiple processors (up to 16)
User microprogrammable - parallelism at the instruction level
Connection ring for inter-Flexible Processor communications
Shared bulk memory units
Separate arithmetic logic unit and hardware multiply.

Disadvantages:
No floating point hardware
Micro-assembly language - difficult to program
Program memory limited to 4k microinstructions.

Based on the investigations to date, the advantages of this system appear to outweigh the disadvantages. However, alternative approaches, such as multimicroprocessor systems, should also be considered to determine the most cost-effective approach for implementing the contextual classifier and other computationally demanding image processing operations for remote sensing.

5. Parallel Implementations of Classification Algorithms

5.1 Introduction

To demonstrate the use of a Flexible Processor (FP) system on a task less complex than the contextual classifier, consider the analysis of Landsat data using a Bayes maximum likelihood classifier (MLC). Landsat measurements are taken from four spectral bands and received as a data vector. Based on decision theory akin to that developed in the section on the contextual classifier model, the vector is classifed by determining the probability that it belongs to each information class and assigning it to the class for which this probability is maximum.

The way in which an FP may be used in implementing a Bayes maximum likelihood classifier is demonstrated below. The techniques described are to be extended to the contextual classification algorithm.

In Section 5.2, methods for implementing the MLC on an FP array are presented. The ways in which the contextual classifer can be implemented on an FP array are presented in Section 5.3.

5.2 Implementation of the Maximum Likelihood Classifier on an FP Array

Two methods for implementing the maximum likelihood classifier (MLC) on an FP array are discussed. The first assigns to each FP a different set of classes, and each FP processes all pixels for its assigned classes. The second method assigns to each FP a different subimage, and each FP processes the pixels in its subimage for all classes. The basic matrix operations, described below, are the same for both methods.

The ability to do a fast matrix multiply is at the heart of efficiently implementing the Bayes maximum likelihood classifer. The form for the matrix multiplications is:

$$(X-U_i)^T (C_i^{-1}) (X-U_i),$$

where X is the data vector, U_i is the mean vector for the ith class, and C_i is the covariance matrix for the ith class.

Consider the use of the FP array to perform these classifications.
Assume there are m distinct classes and the computer system contains p
FPs. Each FP is assigned to process m/p classes. The large file in each
FP is initialized with the inverse of the covariance matrices and mean
vectors for each class it was assigned. The current data vector is
stored in each FP in the temporary file. When a new data vector is
loaded into an FP, it overwrites the previous one. For simplicity, but
without loss of generality, in the following assume that m = p. If m is
greater than p, then in each FP instead of applying just one inverse co-
variance matrix to the data set, several would be applied. This will, of
course, increase the execution time by a factor of approximately m/p.

In standard arithmetic, one would first multiply $(X-U_i)^T$ and C_i^{-1},
creating a new vector. This vector would then be multiplied by $(X-U_i)$
resulting in a scalar. In our implementation, the order has been some-
what altered. $(X-U_i)^T$ is multiplied by a column of C_i^{-1}, accumulating the
results in a variable called "sum." After this is done for column j of
C_i^{-1}, "sum" is multiplied by $(X-U_i)_j$ (the jth element of $(X-U_i)$), accumulat-
ing the result in a variable called "hold" and re-initializing "sum" to
0 [16]. The following is a "pidgeon ALGOL" description of the process
for one pixel:

```
hold = 0;
for j=1 to n do
     begin;
     sum=0;
     for k=1 to n do
          sum=sum+D[k]*C_i^{-1}[k,j];
hold=hold+sum*D[j];
end;
```

where: n = dimension of covariance matrix

   $D[k]$ = kth element of $(X-U_i)$, computed when X is loaded

   $C^{-1}[k,j]$ = element in the kth row and jth column of C_i^{-1}

At the end of the routine, the value contained in the "hold"
variable is the desired scalar. This algorithm requires fewer stores

and fetches than the standard algorithm, so it shortens the run time of the process. All pointers are kept in the index register, further simplifying the process. Finally, because only two accumulators are used, the three GPRs can be kept free for the floating-point operations, while the accumulators are stored elsewhere.

One way to perform this algorithm is to have the host initially send C_i^{-1} and U_i to FP i. The host then sends the current data vector X to FP 0, then FP 1, FP 2, etc. As soon as the FP receives the data vector, it begins the calculation of the value of the discriminant function. After the host gives all FPs the data for pixel (i, j), it waits until FP 0 has calculated the value for its discriminant function. The host then retrieves the value of the discriminant function and loads FP 0 with the data vector for the next pixel. The host executes this process for all the FPs. When the last FP has transmitted the result, the host does a compare and stores the class index corresponding to the maximum of the discriminant values computed for this pixel. Thus, the compares are done by the host while the FPs are computing the discriminant functions for the next pixel, minimizing delay.

An alternative method to perform the pointwise maximum likelihood classification of pixels using a Flexible Processor array is based upon having each FP perform the MLC for a different section of the image. Recall, the contextual classifier performs computations similar to those used by the maximum likelihood classifier, but is complicated by the involvement of "neighboring" pixels.

Consider performing a maximum likelihood classification on an A-by-B image with N Flexible Processors. One way to approach the problem is to divide the image into N subimages and have each Flexible Processor perform the maximum likelihood classification for all pixels in its subimage. This is shown in Figure 2C2.18. If all subimages have the same number of pixels, then the Flexible Processors will be fully utilized and the classification of the entire image will take approximately 1/N as much time as it would take a single Flexible Processor to perform the entire classificiation. Thus, maximum improvement, i.e., a factor of N, is obtained.
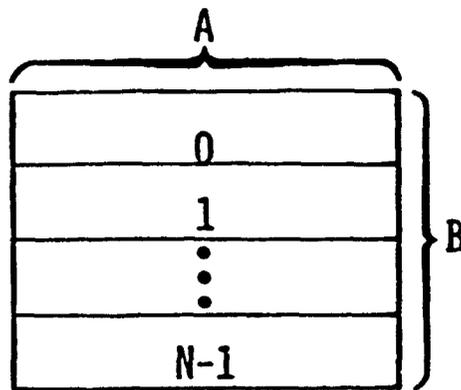
Fig. 2C2.18. An A by B image divided among N Flexible Processors.

Consider the case in which each subimage does not contain the same number of pixels, which will occur if (A*B)/N is not an integer. This will lead to underutilization of the Flexible Processors, but this underutilization will be negligible as will now be shown.

One way to approach this situation is as follows. To each of N-1 Flexible Processors, assign a subimage of size

$$\lceil (A * B)/N \rceil,$$

where $\lceil x \rceil$, the ceiling of x, is the smallest integer greater than or equal to x. To the remaining Flexible Processor assign a subimage of size

$$(A * B) - (\lceil (A * B)/N \rceil * (N-1)).$$

For example, if A = 117 and B = 196 (a typical LACIE image [25]), and N = 16, then

$$\lceil 22,932/16 \rceil = \lceil 1433.25 \rceil = 1434$$

pixels are in each subimage associated with 15 Flexible Processors. The remaining pixels, of which there are

$$22,932 - (15 * 1434) = 1422$$

are associated with one Flexible Processor. This sixteenth Flexible Processor will have fewer pixels to classify and thus will finish before the other Flexible Processors (assuming that, on the average, the time for the floating point calculations is approximately the same for all pixels), which implies some underutilization of this Flexible Processor. Ideally a factor of N = 16 performance improvement over a single Flexible Processor is desired, which, in this case, would require all 16 Flexible Processors to each classify 1434 pixels. To compute the utilization of the Flexible Processor array, divide the number of pixels actually classified by the maximum number that could be classified in the same amount of time if all 16 Flexible Processors were fully utilized. Thus, the utilization is

$$22,932/(16 * 1434) = 99+\%.$$

Therefore, a factor of 99+% of N improvement is obtained.

In general, using the above assignment of pixels to subimages, the utilization of the system is

$$\frac{A * B}{\lceil (A * B)/N \rceil * N}$$

The maximum value of the denominator is A*B+N-1 and occurs when A*B = k*N+1, where k is an arbitrary integer. Therefore,

$$min((A * B)/(\lceil (A * B)/N\rceil * N)) = (A * B)/(A * B + N - 1).$$

Based on typical sizes of remotely sensed images and assuming that the maximum size of a Flexible Processor array is 16,

$$A * B > 10 * N,$$

and

$$(A * B)/(A * B + N-1) > 99\%.$$

Thus, in general, the worst case performance is 99+% of the ideal factor of improvement over a single Flexible Processor.

The maximum likelihood classifier has been programmed on a simulator for a Flexible Processor array at the Laboratory for Applications of Remote Sensing (LARS). The simulator displays the contents of the main registers and provides a variety of tools for debugging Flexible Processor microcode. It is discussed in detail in Section 6. Preliminary simulation tests indicate that a single Flexible Processor will perform a maximum likelihood classification faster than a PDP-11/70. Exact comparisons of the Flexible Processor array performance with other systems are difficult without detailed information about factors such as pre- and/or post-processing of the data not included in the computation time, data precision used, memory load time, etc. However, to give a general idea of the effectiveness of this approach, consider a 256 x 256 classification of Landsat data (n=4) using 16 classes and a complete array of 16 FPs. The total processing time is approximately 10.7 sec. ESL [26] states that their array processor gives up to an increase of 25 times over the IBM 370/158. On the classification of four channels into eight classes, their time is 6.3 sec.

In Appendix 2C2, the MLC programs for the FP are described. Our current algorithm, which runs on the simulator described in Section 6, uses 3526 125-nsec steps to process one pixel (four floating-point component data vector) and two classes, including choosing the maximum value.

In the next subsection, the way in which a parallel processing system such as the Flexible Processor array can be used to perform context classification is examined.

5.3 Contextual Classification on a Flexible Processor System

Consider the implementation of a contextual classifier on an array of Flexible Processors. Assume the neighborhood is horizontally linear, as shown in Figure 2C2.19. Divide the image into subimages of B/N rows A pixels long, as shown in Figure 2C2.18. If B = kN, where k is an integer, there is 100% utilization of the Flexible Processors. Furthermore, there is no overhead for inter-Flexible Processor data transfers, since the entire neighborhood of each pixel is included in its subimage. Therefore, a factor of N improvement is attained.

If (A * B)/N is an integer, but B = kN + x, 0 < x < N, then Flexible Processors can be underutilized in order to keep neighborhoods within subimages, or Flexible Processors can be fully utilized, dividing neighborhoods between Flexible Processors, necessitating inter-Flexible Processor data transfers. This is shown for a simple example in Figure 2C2.20, where N = 2, A = 3, and B = 4. In Figure 2C2.20(a) no inter-Flexible Processor transfers are needed, but Flexible Processor 1 is not fully utilized. In Figure 2C2.20(b) both Flexible Processors are fully utilized, but, due to the horizontally linear neighborhood, at least pixel 11 will have to be sent to Flexible Processor 1 and at least pixel 12 will have to be sent to Flexible Processor 0.

If (A*B)/N is not an integer, some inter-Flexible Processor data transfers will be necessary. The number of transfers will be a function of the way in which the pixels are assigned to Flexible Processors, as in the previous paragraph. To determine the computationally fastest approach whenever B = kN+x, 0<x<N, requires knowledge of the actual image size, the actual number of Flexible Processors used, the exact time required to execute inter-Flexible Processor transfers, and the length of the neighborhood.

There are two other cases of linear neighborhoods. These are vertically linear and diagonally linear, as shown in Figures 2C2.21 and 2C2.22. The analysis for these two cases is similar to that for the horizontally linear case. The vertically linear case is just a 90° rotation of the horizontally linear case. The diagonally linear case can be simplified to a 45°

Fig. 2C2.19. Horizontally linear neighborhoods. Each box is one pixel.

(a)

(b)
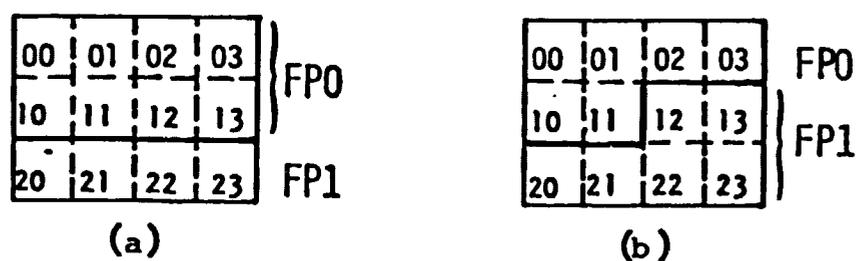
Fig. 2C2.20. Dividing an image N subimages for horizontally linear neighbor-
hoods, where N=2, A=4, and B=3.

(a) Underutilization, no inter-Flexible Processor data
transfers required.

(b) Inter-Flexible Processor data transfers required, full
utilization.
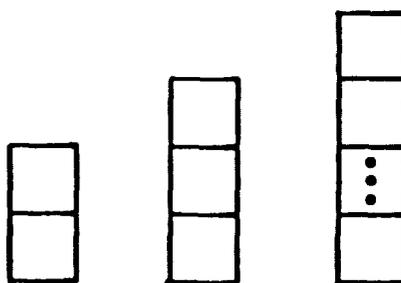
Fig. 2C2.21. Vertically linear neighborhoods. Each box is one pixel.
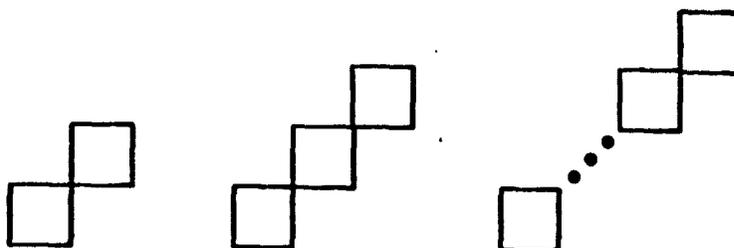
Fig. 2C2.22. Diagonally linear neighborhoods. Each box is one pixel.

A

0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7
5	6	7	8

B
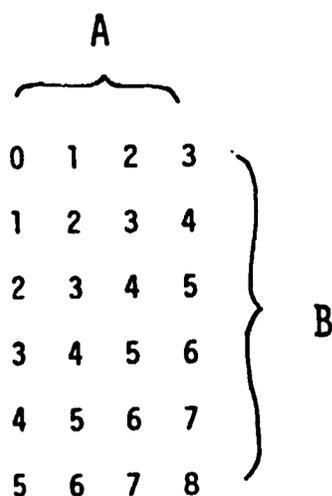
Fig. 2C2.23. The diagonals of an A by B image.
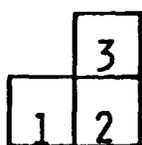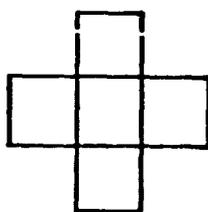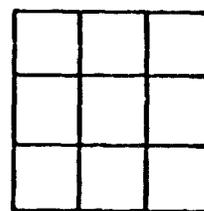
(a)   (b)   (c)

Fig. 2C2.24. Nonlinear neighborhoods. Each box is one pixel.

rotation of the horizontally linear case for B = kN by the proper assignment of pixels to Flexible Processors. Consider an A by B image, A $\leq$ B, and B = k. Label the diagonals from 0 to A+B-2, as shown in Figure 2C2.23 for A = 4 and B = 6. The pixels can then be grouped into B sets of A pixels as follows:

1. For each i, $0 \leq i \leq A-1$ the pixels in diagonals i and i+B form a group of size B,

2. For each j, $A-1 \leq j \leq B-1$, the pixels in diagonal j form a group of size A.

Using these rules, each Flexible Processor is assigned k groups. Thus, the problem has been reduced to the equivalent of the horizontally linear case, which has already been discussed. The case for B = kN+x, $0 < x < N$, is even more complex than for the analogous situation in the horizontally linear case, and requires a detailed tradeoff analysis based on the actual image size, the actual number of Flexible Processors used, the exact time required to execute inter-Flexible Processor data transfers, and the length of the neighborhood.

Now consider nonlinear neighborhoods, that is, neighborhoods which do not fit into one of the linear classes. For example, all of the neighborhoods in Figure 2C2.24 are nonlinear. Figure 2C2.24(a) and its rotations represent the simplest nonlinear neighborhood. It is included in all other nonlinear neighborhoods. Thus, that neighborhood is called the nonlinear kernel neighborhood.

It can be shown that there is no way to partition an A by B image into N (not necessarily equal) sections such that a context classifier using a nonlinear neighborhood can be implemented without involving inter-Flexible Processor data transfers. This will be demonstrated for the nonlinear kernel, and will thus be true for all nonlinear neighborhoods. There are three cases to consider. If there is a horizontal border between two subimages stored in different Flexible Processors, then pixels 1 and 2 in Figure 2C2.24(a) will be different in different Flexible Processors. If there is a vertical border, pixels 2 and 3 will be in different Flexible Processors. If there is a diagonal border, pixels 1 and 2 will be in different Flexible Processors. The way in which to assign pixels to Flexible Processors in

order to minimize computation time will depend upon the particular image
size, number of Flexible Processors used, time required for inter-Flexible
Processor communications and the shape and size of the neighborhood. These
factors will also determine the effectiveness of the use of the Flexible
Processor array for performing context classifications based on a given
neighborhood.

The discussion of performing classifications with the Flexible Pro-
cessor System demonstrates one way in which a multiple-processor system
can be used to speed up the processing of image data. Future work involves
programming the context classifier on the Flexible Processor simulator
using different size and shape neighborhoods and determining the most effi-
cient assignment of pixels to Flexible Processors for each case examined.
The implementation of the classifier on the simulator and eventually on
the actual FP system will provide hard data to verify the effectiveness
of the parallel processing approach.

Through the use of parallel, pipelined, and/or special purpose compu-
ter systems such as the CDC Flexible Processor System, the types of compu-
tations required for the context classifier and other computationally de-
manding processes can be implemented efficiently. This will not only re-
duce the computation time required to do contextual classification but
will also allow the investigation of techniques which may otherwise be
considered infeasible.

6. The Flexible Processor Array System Simulator

6.1 Introduction

Each Flexible Processor (FP) has a complicated microprogrammable internal architecture. This was overviewed in Section 4. As stated earlier, an advantage of this microprogrammable architecture is that it allows parallelism at the instruction level. This makes user verification of the correctness of FP algorithms and accurate mathematical timing analyses of these algorithms very difficult. Thus, in order to debug, verify, and time FP algorithms, a simulator for an array of FPs has been developed. This simulator runs under the UNIX operating system on a PDP-11 series computer, and has been used successfully to program a maximum likelihood classifier, as was discussed in Section 5. It displays the contents of the FP registers on a terminal screen, in a format demonstrated in Appendix 2C3. This section describes the modifications made to the "original" simulator [27] and the organization and operation of the current simulator.

6.2 Modifications Made to the Simulator

The original simulator, written to simulate a single FP [27], was used as a basis for the current version. The modifications made to the original simulator come under four categories: (1) corrections, (2) additional capabilities, (3) improved execution time, and (4) increased documentation about the design of the simulator (program comments).

Our use of the simulator revealed some "bugs" in the system, all of which have been corrected. The simulator now appears to perform as it should.

The original simulator could simulate only one FP. The current version can simulate up to sixteen, the maximum number allowed in an actual system. Each FP in the new version has 20 times more memory capacity (per FP) than previously allowed. The current maximum FP program length is 2000 lines.

The current program code is 15% longer than the original simulator. The simulator occupies 38,040 bytes of main memory during execution. While this is a substantial increase in the space required, most of the increases in space are due to special buffering techniques employed.

Normally, output to the terminal is done one character at a time. This requires the program to generate an interrupt to the operating system for each character to be displayed. The operating system then checks several flags, adds special characters where needed, awakens the device driver, tells the device driver which terminal gets the output, and does the output. The output from a single execution step requires exactly one screen, which is 3370 characters. Buffering is done so that the computer handles the interrupt routine once per screen instead of once per character. The only change in the interrupt routine is that instead of displaying one character, the computer displays 3370. This reduces the load on the system by 3369 interrupt routines per screen of output. Most of the time required for output is not due to the physical transfer of data; rather, it is due to the overhead of the interrupt routine. The net result is that the simulator output is over 3300 times faster with buffering than without. While the different command levels require different size buffers, the buffering has decreased the average time required for a display by a factor of 45.

The PDP-11 series computer uses 16 address bits; thus the maximum amount of data address space is limited to 65,536 bytes. Each simulated FP memory and registers require approximately 60,000 bytes, so a special paging routine was written to page the simulated FP memories and registers in and out of main memory as required. Output to disk is done in units of 2^{16} bytes. This makes the swapping routine run in 1 second. Without buffering, this routine took 2.5 hours of straight transfer time. This program can run on a PDP-11/34 in a time-shared environment.

At the beginning of every major portion of program code, comments describing the program flow and variables modified have been added. This facilitates understanding of the routines and makes program modifications easier.

6.3 Organization of the Simulator

The simulator is divided into four programs, all written in C [28], a language much like PL/I or PASCAL. Each of the four programs performs a different task. "Monh.c" is the system monitor, which interfaces the simulator to the user. "EXECh.c" is the simulator, which simulates all of the system instructions except the I/O and the shift instructions. "Shioh.c" simulates the rest of the instruction set. The "helph.c" program contains a brief help file for the user who is stranded in the monitor routine. In addition, helph.c contains special routines that make the program consistent with all versions of the UNIX operating system. This makes the program portable for use on any system that supports UNIX and the C programming language. In addition, this routine contains all the paging algorithms that are used, making the routines localized, easing possible debugging problems in the future. Some of the modifications to the simulator were done with the aid of LARS programmer Craig Strickland as consultant and debugger.

6.4 Operation of the Simulator

The program structure for a single FP simulation can be represented by the following control tree diagram:
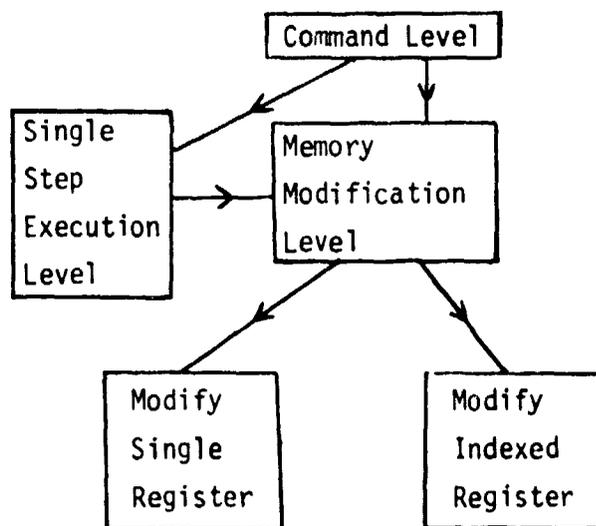
All register files are considered index registers. The 16 FP system is basically the same tree structure, but there is one more level in

the control tree, as follows:
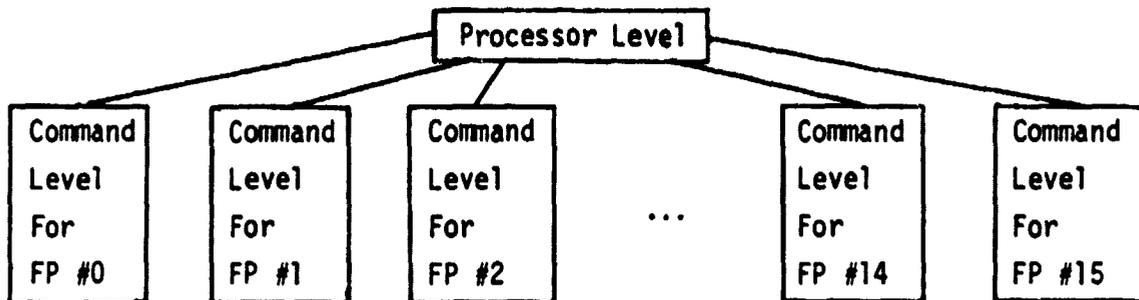
The structure beneath the command level is the same as for the single FP case. If the monitor receives a '#', it will move one node closer to the root of the control tree on any of the branches.

In the Command Level, there are 10 possible commands, which are as follows:

s	Single step program.
m	Go to memory level.
l	Load assembled object code.
t	Print the contents of the registers after the input offset (used for debugging simulator).
v	Save the current register values in a file called status.
e XXX	Execute XXX program steps.
stop	Exit from monitor routine.
! unix	Execute system command.
#	Move up one node to processor level
p	Print out all the registers
h,H, help, Help	Print out the help file, and the values in a file called current node.

If an s is chosen, the simulator will simulate the execution of one program step and will move to the single step node. The following is the command set for the single step node.

s	Single step program.
m	Go to memory level.
e XXX	Execute XXX program steps.
#	Move up one node to command level.
p	Print out all the registers.
h,H, help, Help	Print out the help file, followed by the name of the current node.
dtemp	Print out the contents of the temporary file.
dlarge	Print out the contents of the large file.
dmem	Print out the contents of the micro-memory.

If the m is typed, the only valid arguments are a '#' or a register name. The monitor will print the old value of the register and ask for a new one if the register named is a single register. If the register selected is a register file, the monitor will ask for the index. Upon receiving the index, the monitor will print the old value and prompt the user for input. Valid commands are as follows:

c XXX	Changes the old values to XXX.
i	Increments the index without changing the old value.
^	Decrements the index without changing the old value.
#	Return to original level (either the command level or the single step level, depending on the level in which the m was typed).

Invalid input will yield a "What?" asking for a correct command.

These are all of the functions supported by the simulator at this time. Appendix 2C3 contains flowcharts overviewing the operation of the simulator. As mentioned previously, the maximum likelihood classifier has been implemented using the simulator. We are currently in the process of implementing a contextual classifier.

7. Summary and Concluding Remarks

During this contract year, notable progress has been achieved with respect to the research objectives set out for this task. Specifically:

1. Procedures have been investigated for determining and representing the contextual information in a given scene. The performance of the contextual classifier is found to be sensitive to the accuracy with which the p-context distribution is estimated. Although good results have been achieved, both with real and simulated data (Section 3), further work is needed on methods for determining the context distribution.

2. The contextual classifier algorithm has been analyzed with respect to achieving efficient implementation on a multiprocessor system. It has been shown that under rather severe restrictions on the shape of the contextual neighborhood, an "ideal" speedup by a factor of N, for an N-processor system, can be achieved. Easing of these restrictions definitely incurs a cost in terms of computation time, the details of which are the subject of ongoing analysis (Section 5).

3. Actual implementation of the contextual classifier on multiprocessor systems has been limited to development of a simulator for the CDC Flexible Processor Array System and implementation, on the simulator, of a maximum likelihood classifier (Sections 5 and 6, Appendixes 2C2 and 2C3). Computations performed by the maximum likelihood classifier are identical to many of the computations required for the contextual classifier, but the overall algorithm is considerably simpler. Thus implementing the maximum likelihood classifier provided a useful means for beginning to learn how to program a Flexible Processor Array System.

In support of the above achievements, the mathematical formulation of the contextual classifier has been put on firmer ground and some insights gained into the nature of the spatial context (Section 2). A significant amount of effort has gone into understanding the architectural details of the Flexible Processor, in order to use its facilities effectively (Section 4).

At this point in the study, we may conclude that the contextual classifier does indeed lead to improved classification accuracy by utilizing spatial context information in multispectral earth resources data. Although the computational demands of the proposed contextual classifier are substantial, multiprocessor systems such as the CDC Flexible Processor Array System can be used to achieve efficient implementation of this and other image processing algorithms.

Ongoing research in connection with this project will be directed toward better understanding the nature of contextual information in multi-spectral image data and exploiting the computational efficiencies to be gained through parallelism and other special features of advanced data processing system architectures.

8. References

1. Swain, P. H. and Davis, S. M., eds., Remote Sensing: The Quantative Approach, McGraw-Hill International Book Co., New York, 1978.

2. Kettig, R. L. and Landgrebe, D. A., "Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects," IEEE Trans. Geoscience Electronics, Vol. GE-14, pp. 19-26, Jan. 1976.

3. Haralick, R. M., Shanmugam, K., and Dinstein, I., "Textural Features for Image Classification," IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-3, pp. 610-621, Nov. 1973.

4. Kettig, R. L. and Landgrebe, D. A., "Computer Classification of Remotely Sensed Multispectral Image Data by Extraction and Classification of Homogeneous Objects," LARS Technical Report 050975, Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, IN 47907, May 1975.

5. Welch, J. R., and Salter, K. G., "A Context Algorithm for Pattern Recognition and Image Interpretation," IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-1, pp. 24-30, Jan. 1971.

6. Yu, T. S. and Fu, K. S., "Statistical Pattern Recognition Using Contextual Information," Technical Report TR-EE 78-17, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907, Mar. 1978.

7. Robbins, "Asymptotically Subminimax Solutions of Compound Statistical Decision Problems," Proc. Second Berkeley Symp. Mathematical Statistics and Probability, pp. 157-163, University of California Press, 1951.

8. VanRyzin, J., "The Compound Decision Problem With m x n Finite Loss Matrix," Annals of Mathematical Statistics, Vol. 37, pp. 412-424, 1966.

9. VanRyzin, J., "The Sequential Compound Decision Problem With m x n Finite Loss Matrix," Annals of Mathematical Statistics, Vol. 37, pp. 954-975, 1966.

10. Cover, T. and Shenhar, A., "Compound Bayes Predictors for Sequences With Apparent Markov Structure," IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-7, pp. 421-423, June 1977.

11. Vardeman, S., "A Note on the Applicability of Sequence Compound Decision Schemes," Scandinavian Journal of Statistics, Vol. 6, No. 2, 1979.

12. Vardeman, S., "Solutions to k-Extended Compound Decision Problems, Bootstrap and Bayes," in preparation.

13. Gilliland, D. and Hannan, J., "On the Extended Compound Decision Problem," Annals of Mathematical Statistics, Vol. 40, pp. 1536-1541, 1969.

14. Ballard, J., Gilliland, D. and Hannan, J., "$O(N^{-\frac{1}{2}})$ Convergence to k-Extended Bayes Risk in the Sequence Compound Decision Problem With m x n Component," Research Memo RM-333, Statistics and Probability, Michigan State University, 1975.

15. Swain, P. H., Siegel, H. J. and Smith, B. W., "Contextual Classification of Multispectral Remote Sensing Data Using a Multiprocessor System," to appear in IEEE Trans. Geoscience Electronics, Apr. 1980.

16. Allen, G. R., Bonrud, L. O., Cosgrove, J. J. and Stone, R. M., "The Design and Use of Special Purpose Processors for the Machine Processing of Remotely Sensed Data," Proc. Conf. Machine Processing of Remotely Sensed Data, IEEE Cat. No. 73CH0834-2GE, pp. 1A-25 - 1A-42, Oct. 1973.

17. Siegel, H. J., "Preliminary Design of a Versatile Parallel Image Processing System," Proc. Third Biennial Conf. Computing in Indiana, pp. 11-25, Indiana University, Bloomington, IN, Apr. 1978.

18. Siegel, H. J., Siegel, L. J., McMillen, R. J., Mueller, Jr., P. T. and Smith, S. D., "An SIMD/MIMD Multimicroprocessor System for Image Processing and Pattern Recognition," Proc. 1979 IEEE Computer Society Conf. Pattern Recognition and Image Processing, IEEE Cat. No. CH1428-2, pp. 214-224, Aug. 1979.

19. Fu, K. S., "Special Computer Architectures for Pattern Recognition and Image Processing - An Overview," Proc. 1978 Natl. Computer Conf., pp. 1003-1013, June 1978.

20. Control Data Corp., "Cyber-Ikon Image Processing System Design Concepts," Digital Image Systems Division, Control Data Corp., Minneapolis, MN, Jan. 1977.

21. Control Data Corp., "Cyber-Ikon Flexible Processor Programming Textbook," Digital Image Systems Division, Control Data Corp., Minneapolis, MN, Dec. 1978.

22. Kast, J. L., Swain, P. H. and Phillips, T. L., "The Feasibility of Using a Cyber-Ikon System as the Nucleus of an Experiment Agricultural Data Center,: LARS Contract Report 021678, Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, IN 47907, Feb. 1978.

23. Krause, K. W., "Use of the CDC Cyber-Ikon for a Bayes' Maximum Likelihood Classifier," unpublished EE696 project report, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907, Aug. 1978.

24. Swain, P. H. and Davis, S. M., eds., Remote Sensing: The Quantitative Approach, McGraw-Hill International Book Co., New York, 1978.

25. MacDonald, R. B., Hall, F. G. and Erb, R. B., "The Use of Landsat Data in a Large Area Crop Inventory Experiment (LACIE)," Proc. Symp. Machine Processing of Remotely Sensed Data, IEEE Cat. No. 75CH1009-0-C, pp. 1B-1 - 1B-23, June 1975.

26. ESL, Inc., "Advanced Scientific Array Processor," descriptive manual, ESL, Java Dr., Sunnyvale, CA.

27. Krause, K. W., "Use of the CDC Cyber-Ikon Simulator," unpublished EE696 project report, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907, Aug. 1978.

28. Kernighan, B. W. and Ritchie, D. M., <u>The C Programming Language</u>, Prentice-Hall, Englewood Cliffs, NJ, 1978.

APPENDIX 2C2

IMPLEMENTATION OF THE MAXIMUM LIKELIHOOD CLASSIFIER
ON A FLEXIBLE PROCESSOR

A. INITIALIZATION OF THE FPs BY THE HOST COMPUTER

1. Initialize memory to zeroes.

2. Send FP size, p, sigma, X, and U.

3. Calculate det(sigma)

4. Calculate inv(sigma)

5. Calculate ln(det(sigma))

6. Calculate ln(p(w))

7. Send FP ln|sigma|, ln(p(w))

8. Send FP inv(sigma)

B. INTERRUPT ROUTINE FOR FLEXIBLE PROCESSOR

```
┌─────────────────────┐
│ Interrupt routine   │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│  save   all         │
│ registers           │
│                     │
│  increment new      │
│ value pointer       │
│                     │
│ store   value from  │
│ in given location   │
└─────────────────────┘
           │
           ▼
        ◇ size                    no      ┌─────────────┐
        ◇ = pointer?   ───────────────────▶│ unsave      │
           │                               │ registers   │
           │ Yes                           └─────────────┘
           │                                      │
           │                                      ▼
           ▼                               ┌─────────────┐
┌─────────────────────┐                    │   return    │
│ choose max          │                    └─────────────┘
│ value  ̄ ..d         │
│ print it            │
└─────────────────────┘
           │
           ▼
┌─────────────────────┐
│        END          │
└─────────────────────┘
```
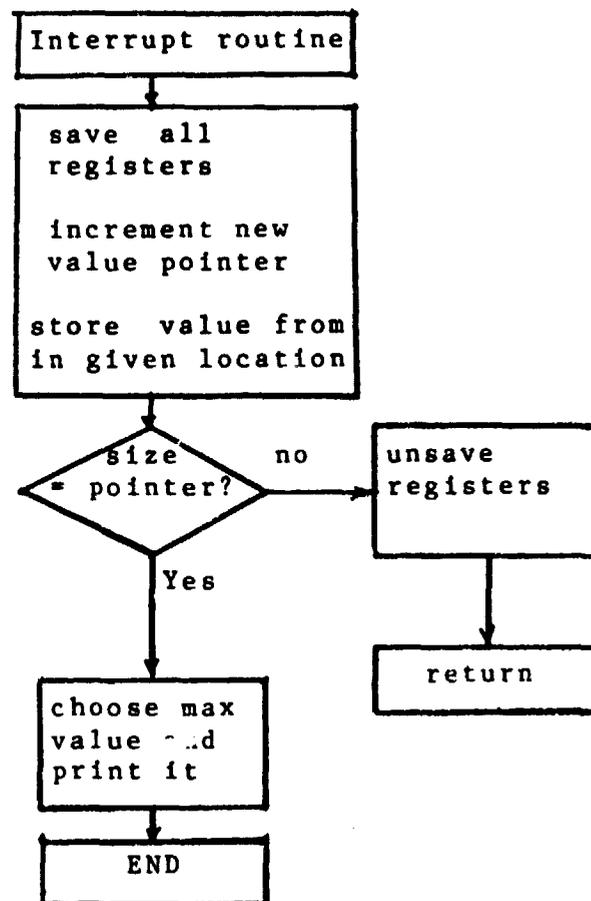
C.1. Load Data Into FP.

1)   Zero all registers. This includes all in-
dex registers, index compare registers, large file
address registers, maintenence compare registers
and temporary file address (both read and write)
registers.

2)   Read the first number and store it in re-
gister F.

3)   Copy the number stored in the F register
into the index compare registers number 0 and 1.
(This number is the dimension of sigma.)

4)   Load all conditions. (This means that the
index compare registers are going to test for
equality to n.) Index register three will check
for equality to zero.

5)   Test and increment Index register three.
If it is not equal to zero read a number, load it
into the F register.

6)   Move the F register to temporary file zero
while incrementing the write counter.

7)   If index register 0 does not equal n, go
to step 5.

8)   Zero all index registers while moving n to
the P register of the multiply while trapping in-
terrupts. (This can be done using the "sr" com-
mand.)

9)   With interrupts trapped, move multiply
output to condition register 2. (This means that
the condition registers are now set to check for
index regis- ter 0 and 1 equal to n, index regis-
ter 2 equal to n squared and index register 3
equal to zero.)

10)   Test and increment index register 2. If it
is n squared, exit.

11)   Read a number, store it in large file
zero, while simultaneously incrementing its ad-
dress buffer.

12)   Jump to step 10.

C.2. Storage Format

The Storage format used in the processing   scheme is as follows:

Temporary files		Large Files	
	n	Sigma[1,1]	First Covariance
	Hold	Sigma[2,1]	Matrix.
normalixed data	X[1,1]	:	
vector for	X[1,2]	Sigma[n,1]	
class one	X[1,3]	Sigma[1,2]	
	X[1,4]	:	
normalized data	Y[1,1]	Sigma[2,n]	
vector ror	Y[1,2]	:	
class two	Y[1,3]	Sigma[n,n]	
	Y[1,4]	:	
		Sigma[n,n]	
		:	
		Sigma[1,1]	Second Covariance
		Sigma[2,1]	Matrix
		:	
		Sigma[n,1]	
		Sigma[1,2]	
		:	
		Sigma[2,1]	
		:	
		Sigma[2,n]	
		:	
		Sigma[n,n]	
mean vector for		U[1,1]	
class one		U[1,2]	
		U[1,3]	
		U[1,4]	
mean vector for		V[1,1]	
class two		V[1,2]	
		V[1,3]	
		V[1,4]	

C.3. First Matrix Multiplication

1)   Initialize all registers. Move 1 to the read address of temporary file zero. Zero all other index registers, large file addresses, temporary file addresses.

2)   Move temporary file 0 to the E register (while incrementing the read address pointer.)

3)   Move large file 0 to the G register (while incrementing the address pointer.)

4)   Call floating point multiply routine.

5)   Store result in temporary file 1, while increasing the write pointer.

6)   If index 0 is n, jump to the subroutine called sum.

7)   If index 1 is n, jump to the next multiply routine.

8)   Increment index reg 0.

9)   Go to step 1.

10)   Increment index 1 by 1.

11)   Zero F register. (This is used as the accumulater for the floating point add.)

12)   Zero index register 0.

13)   Zero temporary file 1 read address.

14)   Test and increment index 0. If it = n, go to step 16.

15)   Call floating point add subroutine. (40 cycles.) (this routine has been modified to increment the temporary file 1 read pointer as it goes along, s this is not necessary.)

16)   Go to step 14.

17)   Temporary file 0 pointer = 1.

18)   Store f in large file 1 (while incrementing the pointer.) (F contains the result of the n floating point adds.)

19)   Return to calling routine

C.4.   Second Matrix Multiplication

1)   Zero all pointers to large files and temporary file 1 address.

2)   Write a 0 to temporary file address 0.

3)   Transfer temporary file zero memory location 0 to index register 3.

4)   Test and decrement register 3. If zero go to wrap up.

5)   While incrementing the pointer to temporary file 0, move the contents to the E register.

6)   While incrementing the pointer to large file 1, move the contents to the G register.

7)   Call the floating point multiplication routine.

8)   Call the floating point add routine.

9)   Send the result to temporary file 1.

10)   Go to step 4.

D. FLOATING POINT ADDITION ROUTINE
F = E + G

```
┌──────────────┐
│ floating pt. │
│ addn. routine│
└──────────────┘
        │
        ▼
┌──────────────┐
│ strip signs  │
│  and save    │
│    for       │
│  later use   │
└──────────────┘
        │
        ▼
     ◇ one          yes    ┌──────────────┐
      of the    ──────────▶│ return other │
     numbers              │    number    │
      = 0? ◇               └──────────────┘
        │ no
        ▼
      ◇          yes   ┌──────────────┐
   |E|<|G| ──────────▶│   reverse    │
      ◇               │    the       │
        │              │   numbers    │
        │◀─────────────└──────────────┘
        ▼
┌──────────────┐
│    align     │
│   binary     │
│   points     │
└──────────────┘
        │
        ▼
    ◇ was         yes   ┌──────────────┐
     E<0 and  ────────▶│ result = G-E │
     G>0? ◇            │  normalize   │
        │              │   result     │
        │              │   return     │
        ▼              └──────────────┘
    ◇ was         yes   ┌──────────────┐
     E>0          ────▶│ result = E-G │
     and              │  normalize   │
     G<0? ◇           │   result     │
        │             │   return     │
        ▼             └──────────────┘
┌──────────────┐
│ result = E+G │
│  normalize   │
│   result     │
│ Sng (result) =│
│  old Sgn(E)  │
│   return     │
└──────────────┘
```

E. FLOATING POINT MULTIPLICATION ROUTINE

$$F = E + G$$

```
┌─────────────────┐
│  floating point │
│ multiply routine│
└─────────────────┘
         │
         ▼
┌─────────────────┐
│multiply mantissas│
│  add exponents  │
│  F0 = product   │
│  F1 = sum       │
└─────────────────┘
         │
         ▼
      ╱◇╲           yes      ┌─────────────┐
    ╱     ╲    ──────────▶   │   F1 = 0    │
   ◇ F0 = 0? ◇               │   return    │
    ╲     ╱                  └─────────────┘
      ╲◇╱
         │ no
         ▼
┌─────────────────┐
│ normalize F0,F1 │
│     return      │
└─────────────────┘
```

F. FLOATING POINT COMPARE ROUTINE
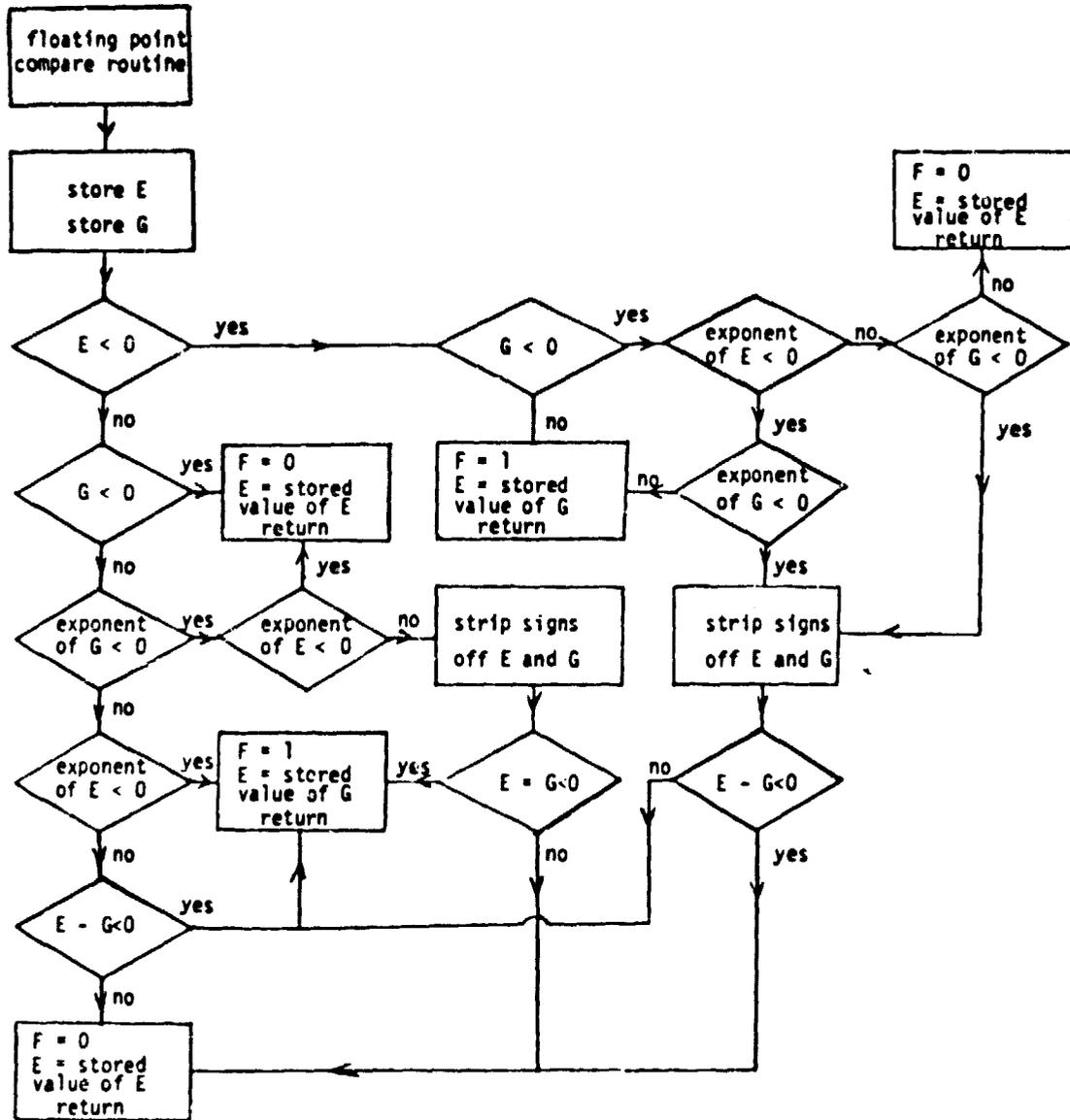
$E = \max(E, G)$

$F = 1$ if $E = G$

$= 0$ if $E \neq G$

G. ACTUAL FLEXIBLE PROCESSOR PROGRAM

```
     ORG     (0)A
*               AVERAGE TIME: 612 CYCLES PER PIXEL (DOWN 10%)
*               MIN TIME:     218 CYCLES PER PIXEL (DOWN 30%)
*
*****************************************************************
*****BAYES MAXIMUM LIKELIHOOD CLASSIFIER VER. 091579  2:50*****
*****FOR TWO PIXELS.............................................****
*****************************************************************
     TC *    *    *    *          FPLR               MAR        NOP
     TC *    *    *    *          S 0000             *          *
*   FIRST INTERRUPT ROUTINE. THIS ROUTINE HANDLES THE INTER- *
*   RUPT TO LOAD THE COVARIANCE MATRICES, THE MEAN VECTORS    *
*   AND THE DATA VECTOR.                                      *
     ORG     001E
     TC *    *    *    *          VINR               MAR        NOP
     TC *    *    *    *          S 0000             *          *
* THIS ROUTINE WILL HANDLE THE INTERRUPT WHEN THE HOST JUST   *
* NEEDS TO ENTER THE DATA VECTOR.                             *
     ORG     00FE
     TC *    *    *    *          S 0000             *          *
* THERE VALUES ARE TO BE LOADED INTO COMPARE REGISTER 3.      *
* THESE WILL TEST THE RESPECTIVE REGISTERS FOR INEQUALITY TO  *
* THEIR COMPARE REGISTERS.                                    *
     TC *    CLA *    *          S 0000             TOWA       T1WA
     TC *    *    *    *          S 0001             TFON       TF1N
*   THIS WILL CLEAR ALL OF THE INDEX REGISTERS AND ZERO THE   *
*   TEMPORARY FILE WRITE ADDRESSES.                           *
     TC *    *    *    *          S 0000             NOP        CMR0
*   THIS WILL CLEAR THE TEMPORARY FILE 0 READ ADDRESS AND THE *
*   CONDITION REGISTER TO PREVENT SPURIOUS RESULTS.           *
     TC *    *    *    *          S 0000             NOP        CMR2
* THIS WILL ZERO THE OTHER CONDITION REGISTER AND THE TEMP    *
* FILE READ ADDRESS. THE DIMENSION OF THE INCOMING DATA IS    *
* ASSUMED TO BE 4X4.   IF THEMATIC MAPPER DATA IS TO BE USED  *
* THE MATRIX WILL BE FIVE BY FIVE.                            *
     TC *    *    *    *          S 0004             F0         ICR3
* THIS WILL STORE N IN THE INDEX COMPARE REGISTERS,           *
     TR *    *    *    *    *                 NOP   NOP   F0    ICR1
     TC *    *    *    *          S 0010             NOP        ICR2
* THIS IS JUST SETTING UP THE COUNTER VARIABLES FOR THE LOOP.*
WAIT TC *    *    *    *          WAIT               MAR        NOP
     TC *    *    *    *          S FFFF             BRG0       BRG1
     TR *    *    *    PUPU *              MULT      E0         *
* THE HOST WILL START EXECUTION AT 100 AND WAIT HERE FOR THE  *
* HOST TO INTERRUPT THE FP, AT WHICH POINT THE FP WILL DO A   *
* PROGRAM JUMP TO S0007, WHERE THERE WILL BE A JUMP TO THE    *
* CORRECT ROUTINE.                                            *
*    THIS IS THE WAIT ROUTINE, WHICH WAITS FOR AN INTERRUPT.  *
FPMR TR *    *    *    *    E                  A1   E0    A0    Q
*  LOAD MULTIPLICAND                                          *
     TR *    *    *    *    G                  A0   P     A1    G0
* LOAD MULTIPLIER                                             *
     TC *    *    *    *          S 0004             *          CMR0
* THIS CONDITION WILL CHECK TO SEE OF F0 < 0.                 *
     TC *    *    *    *          S 0002             *          CMR3
*    IS INDEX0 = COMPARE REGISTER 0?                          *
     TR *    CL0 *    PUQU ADD                 A1   *     A0    F1
     TR *    *    *    PUQU *                 MULT  F0    F1    E1
     TR *    *    *    *                       *    *     F0    ICR0
*IF THE VALUE RETURNED IS ZERO, ZERO BOTH REGISTERS,   RETURN*
```

```
        TC AU  •    JP •              $ 0000            F0           F1
        TC AD  •    DF •              $ 0000            •            •
•  IF FO IS JUSTIFIED, RETURN. THE PRODUCT IS NORMALIZED.           •
        TC •   •    •  •              $ 000C            •            CMR1
        TC TNN •    JP •              $ 8GFF            G1           •
        TR TNN •    DF •      AND               A0      •      A1    F1
•    SAVE THE EXPONENT IN G1, CLEAR E1 FOR A COUNTER                 •
        TR •   •    •  •      ZRO               F1      G1     A0    E1
        TR •   CLO •  •       ZRO               A0      E0     •     •
•  BY HERE, PRODUCT CANNOT BE ZERC.THE NORMALIZATION PROCESS        •
•  WILL TAKE LESS THAN FOUR REPEATS OF THIS LOOP. IF IT EVER        •
•  TAKES MORE, THERE IS SOMETHING BRANCHING DIRECTLY TO THIS        •
•  PROCESS.                                                         •
NRM  TR FNN INO •  •       E+1               A0      EO     A1      •
        TC FNN •  •   •  •            NRM               MAR          •
        SH FNN •  •   •  •                    NZIN   LZIN   NZIN    S
•    BY NOW, THE RESULT MUST BE NORMALIZED!!!!!                     •
        TR •   •   •   •      G                A0      •      A1     F1
        TR •   •   •   •      E                A0      G1     A1     G0
        TR •   •   •   •      •                F0      EO     F1     E1
        SH •   •   •   •      •                LCIR   LCIR   LZIN   S
        TR •   •   •   •      AN6               A0      E0     A1     E1
•THIS WILL TAKE THE NORMALIZED RESULT, SHIFT IT LEFT, ADJUST •
•THE EXPONENT, SO THAT IT AGREES WITH THE MANTISSA.          •
        TC •   •   •   •              $ 01FF            G1           •
        TC •   •   •   •              $ FFFF            •            G0
        TR •   •   JP •      ACB                A0      •      A1     F1
•  THIS WILL "MASK OFF" ANY CARRIES INTO THE UNUSED PORTION OF•
•  THE EXPONENT.                                             •
        SH •   •    DF •    •                  NZIN   RCIR   NZIN    S
•  THIS ROUTINE DOES THE INITIAL SETUP OF THE VARIABLES      •
FPLR TC •    CL2 •  •              $ 0000                LOAD        L1AD
•THIS CLEARS ALL THE INDEX REGISTERS AND THE LARGE FILE WRITE•
•  POINTERS.                                                 •
        TC •   •   •   •              $ 0020                NOP        ICR2
        TC •   •   •   •              $ 0010                NOP        CMR3
•    THE 0010 TESTS FOR INDEX2 <> ITS COMPARE.               •
•THIS WILL LOAD THE COMPARE REGISTER TO CHECK FOR INDEX      •
•REGITER EGUAL TO ITS STORED VALUE.                          •
        TC •   •   •   •              $ 0009                TOWA       T1WA
•THIS LOADS THE TEMPORARY FILE WITH THE OCTAL LOCATION OF THE•
•MEAN VECTOR.                                                •
        TC •   •   •   •              $ 0088                F0          •
        IO •   •   •   •   •                    F0      •      DS    LSB  • • • •
•THIS LOADS THE BANK AND ADDRESS LOCATION OF THE COVARIANCE  •
•MATRIX                                                      •
•          THIS ROUTINE LOADS THE COVARIANCE MATRIX.         •
IMR  IO •    IN2 •  •    •                     •      •      DS    R  • •      •
        TR •   •   •   •   •                    70     F0     21     F1
•    THIS LOADS THE MANTISSA INTO THE F1 REGISTER.           •
•    AND  LOADS THE EXPONENT INTO THE F0 REGISTER.           •
        TC AD •   •   •             IMR                 MAR          NOP
        TR •   •   •   •   •                    F0     LFOU    F1    LF1U
•    THIS IS THE ROUTINE THAT LOADS THE MEAN VECTOR.         •
MNR  TC •   CL3 •  •              $ 0040                NOP          CMR3
        TC •   •   •   •              $ 0008                NOP          ICR3
•  THE 0040 IN CMR3 TESTS FOR INDEX3 <> ITS COMPARE          •
IMNR TR •   IN3 •  •    •                     20     F0     21     F1
•  THIS DOES THE I/O CALL AND LOADS THE NUMBER INTO THE FO-F1•
•  REGISTER PAIR                                             •
```

```
*   THIS DOES THE I/O CALL FOR THE NEXT NUMBER                        *
        SH *   *   *   *   *                      *     LN31  *      S
*  THIS SHIFTS THE MEAN VECTOR TO THE 'LEFT, AND NEGATES THE          *
*  SIGN BIT.                                                          *
        SH *   *   *   *   *                      *     RCIR  *      S
*                                                                    *
*THIS NEGATES THE SIGN OF THE MEAN VECTOR AND SHIFTS IT IN TO*
*THE SIGN POSITION.  THIS IS DONE BECAUSE THE VECTOR MANTISSA*
*IS IN S&M FORM.  THIS WAY, AN ADDITION TO THE VECTOR WILL     *
*ACTUALLY  PERFORM THE OPERATION OF SUBTRACTING THE VECTOR     *
*FROM THE ADDEND.                                                *
        IO AD  *   *   *   *               *      * .   DS     R  * *        *
*THIS DOES THE I/O CALL FOR THE NEXT NUMBER (IF THERE IS ONE)*
        TC AD  *   *   *        IMNR                MAR        NOP
        TR *   *   *   *   *                  FO    LFOU   F1  LF1U
*LOAD THE NEXT ELEMENT IN THE VECTOR, AND STORE THE NEW VALUE*
*THIS ROUTINE LOADS THE NORMALIZES THE DATA VECTOR.  IT CAN   *
*BE CALLED TO EXECUTE BY ITSELF.                                *
        TC *   *   *   *       $ 0004                 *        ICR3
VINR    TC *   *   *   *       $ 000A               TOBA       T1BA
        TC *   *   *   *       $ 0040               NOP        CMR3
*    THE 0040 TESTS FOR INDEX 3 <> ITS COMPARE                 *
        TC *   CL3 *   *       $ 0027               FO         NOP
        IO *   *   *   *   *                  FO    *    DS    LSB * *       *
*    THIS INITIALIZES THE LOCATION OF THE READ POINTER.       ·*
        IO *   *   *   *   *                  *     *    DS    R  * *        *
*    THIS STARTS THE FIRST READ.                                *
        TC *   *   *   *       $ 0020               LOAD       L1AD
LOIP    TR *   IN3 *   *   *                  Z0    FO    Z1   F1
*  THIS LOADS THE MANTISSA AND EXPONENT INTO THE FO-1 REGISTER*
*  PAIR.                                                       *
*      THE FOLLOWING DOES THE I/O CALL.
        IO AD  *   *   *   *                  *     *    DS    R  * *        *
*      THIS IS EXECUTED BEFORE THE JUMP AND IT WILL LOAD THE NEEDED DATA INTO
*      THE Z0-Z1 REGISTER PAIR  BEFORE IT IS NEEDED, ELIMINATING A TWO CYCLE
*      NOT READY WAIT.
        TC AD  *   *   *       LOIP                 MAR        NOP
        TC *   *   *   *       $ 0000               BRG0       BRG1
*      AFTER NORMALIZING THE DATA VECTOR, STORE IT, REPEAT UNTIL ALL
*      THE ELEMENTS ARE FINISHED, THEN REPEAT THE CYCLE UNTIL ALL FOUR ELEMENTS
*      ARE FINISHED BEING PROCESSED.
        TC *   *   *   *       $ 0020               LOAD       L1AD
        TC *   *   *   *       $ 000A               TORA       T1RA
        TC *   CL3 *   *       $ 0002               TOWA       T1WA
LO1P    TR *   IN3 *   *   *                  TFOU  BRG0  TF1U BRG1
        TC *   *   SR *          . FPAR             MAR        *
        TR *   *   *   *   *                  LFOU  FO    LF1U F1
        TC *   *   *   *       $ 0040               *          CMR3
        TC AD  *   *   *       LO1P                 MAR        *
        TR *   *   *   *   *                  FO    TFOU   F1  TF1U
        TC *   CL3 *   *       $ 000A               TORA       T1RA
LO2P    TR *   IN3 *   *   *                  TFOU  BRG0  TF1U BRG1
        TC *   *   SR *          FPAR               MAR        *
        TR *   *   *   *   *                  LFOU  FO    LF1U F1
        TC *   *   *   *       $ 0040               *          CMR3
        TC AD  *   *   *       LO2P                 MAR        *
        TR *   *   *   *   *                  FO    TFOU   F1  TF1U
*      THIS STORES THE DATA NORMALIZED DATA VECTOR IN LOCATIONS 2-5 OF THE
*      TEMPORARY FILE.  THE SECOND VECTOR WILL APPEAR IN LOCATIONS 6-9 OF
*      THE TEMPORARY FILE.
```

```
*     THIS WILL FALL THROUGH TO THE MATRIX PROCESSING ROUTINE.
* THIS IS THE BEGINNING OF THE MATRIX MULTIPLY ROUTINE.
STRA TC *   CLA *  *       $ 0000              LOAD       L1AD
     TC *   *   *  *       $ 0000              BRG0       BRG1
     TC *   *   *  *       $ 0001              TOBA       T1BA
     TC *   *   *  *       $ 0001              TFOU       TF1U
     TC *   *   *  *       $ 0002              TOBA       T1BA
MLTY TR *   IN1 *  *    *              TFOU EO    TF1U  E1
* THIS LOADS THE MULTIPLICAND INTO THE EO-E1 REGISTER PAIR.
     TC *   *   SR *   FPMR                    MAR        NOP
* THIS DOES THE PROGRAM JUMP TO THE FLOATING POINT MULTIPLY ROUTINE.
     TR *   *   *  *    *              LF1U G1    LFOU  GO
* THIS STEP IS DONE BEFORE THE JUMP IS ACTUALLY EXECUTED. THIS WILL LOAD THE
* MULTIPLIER INTO THE GO-G1 REGISTER PAIR. (F=EXG FLOATING POINT MULT)
     TC *   *   SR *       FPAR                MAR        NOP
* THIS STEP WILL DO A JUMP TO THE FLOATING POINT ADDITION ROUTINE.THIS ROUT-
* INE CALCULATES THE SUM OF THE CONTENTS OF THE F REGISTER AND THE BRG REGIS-
* TER PAIR. THE RESULT OF THE ADD IS THEN STORED IN THE F REGISTER.
     TC *   *   *  *       $ 0004              NOP        ICR1
     TC *   *   *  *       $ 0004              NOP        CMR3
*    THE 0004 TESTS FOR INDEX1 <> ITS COMPARE
* THIS IS EXECUTED BEFORE THE JUMP.  IT WILL JUST LOAD THE CONDITION REGISTER
* WITH THE NEXT CONDITION TO BE TESTED.
     TC AD  *   *  *       MLTY                MAR        NOP
     TR *   *   *  *    *              FO   BRG0 F1    BRG1
* ON INDEX REGISTER 1 NOT EQUAL TO ITS COMPARE, JUMP TO BEGINNING OF MULTIPLY
* ROUTINE.
     TC *   *   *  *       $ 0001              TOBA       T1BA
     TR *   *   *  *    *              TF1N EO    NOP   NOP
*    GET ADDRESS OF JTH ITEM IN THE DATA VECTOR.
     TR *   *   *  *   ACO                A1   TFOD AO    TF1D
     TR *   *   *  *   ACO                AO   TORA AO    T1RA
* THE ABOVE WAS A CHANGE TO INSURE THAT THE PROGRAM WORKS, THIS IS KEPT.
* THIS WILL UPDATE THE ADDRESS FOR THE NEXT ROUND, STORE IT, AND POINT TO THE
* ITEM IN QUESTION.
     TR *   IN2 *  *    *              TFOC EO    TF1C  E1
* THIS WILL LOAD THE MULTIPLIER FOR THE SECOND MULTIPLY INTO THE EO-E1 REG-
* ISTER PAIR.  SIMULTANEOUSLY, THIS WILL ZERO THE TEMP FILE POINTERS. THEY
* WILL NOW POINT TO THE LOCATION OF THE ACCUMULATOR.
     TR *   *   *  *    *              BSRO G1    BSR1  GO
     TC *   *   SR *   FPMR                    MAR        NOP
     TR *   *   *  *   G                  AO   G1   A1    GO
* THIS IS JUST A SUBROUTINE JUMP TO THE FLOATING POINT MULTIPLY ROUTINE.
* F=EXG
     TC *   *   SR *       FPAR                MAR        NOP
     TR *   *   *  *    *              TFON ERG0 TF1N  BRG1
*    F=F+BRG. THIS CALCULATES THE SUBTOTAL OF THE MATRIX MULTIPLY.
     TR *   *   *  *    *              FO   TFON F1    TF1N
     TC *   *   *  *       $ 0002              TOBA       T1BA
* THE ABOVE TWO STEPS LOAD THE SUB TOTAL INTO THE TEMPORARY FILE LOCATION
* ZERO.  IT THEM RESETS THE READ AND WRITE POINTERS OF THE TEMPORARY FILE TO
* LOCATION TWO.
     TC *   *   *  *       $ 0004              NOP        ICR2
     TC *   *   *  *       $ 0010              NOP        CMR3
*    THE 0010 TESTS FOR INDEX2 # ITS COMPARE.
* THIS WILL DO A TEST FOR INDEX O NOT EQUAL TO ITS COMPARE REGISTER.
     TC AD  CL1 *  *       MLTY                MAR        NOP
     TC *   *   *  *       $ 0000              BRG0       BRG1
     TR *   *   *  PUQL *                 *    *    MULT  MCR3
     TC *   *   *  *       $ 0000              TOBA       T1BA
```

```
        TC •   •   •   •           $ 0026              LOAD        L1AD
        TR •   •   •   •     •                 TFON  LFON  TF1N  LF1N
        TC •   •   •   •           $ 0000              TFON        TF1N
ST2A    TC •   CLA •   •           $ 0010              LOAD        L1AD
        TC •   •   •   •           $ 0000              BRG0        BRG1
        TC •   •   •   •           $ 0001              TOBA        T1BA
        TC •   •   •   •           $ 0005              TFON        TF1N
        TC •   •   •   •           $ 0006              TOBA        T1BA
ML2Y    TR •   IN1 •   •     •                 TFOU  E0    TF1U  E1
```
* THIS LOADS THE MULTIPLICAND INTO THE E0-E1 REGISTER PAIR.
```
        TC •   •   SR •     FPMR                       MAR         NOP
```
* THIS DOES THE PROGRAM JUMP TO THE FLOATING POINT MULTIPLY ROUTINE.
```
        TR •   •   •   •     •                 LF1U  G1    .LFOU  G0
```
* THIS STEP IS DONE BEFORE THE JUMP IS ACTUALLY EXECUTED. THIS WILL LOAD THE
* MULTIPLIER INTO THE G0-G1 REGISTER PAIR. (F=EXG FLOATING POINT MULT)
```
        TC •   •   SR •     F.PAR                      MAR    .    NOP
```
* THIS STEP WILL DO A JUMP TO THE FLOATING POINT ADDITION ROUTINE. THIS ROUT-
* INE CALCULATES THE SUM OF THE CONTENTS OF THE F REGISTER AND THE BRG REGIS-
* TER PAIR. THE RESULT OF THE ADD IS THEN STORED IN THE F REGISTER.
```
        TC •   •   •   •           $ 0004              NOP         ICR1
        TC •   •   •   •           $ 00C4              NOP         CMR3
```
*   THE 0004 TESTS FOR INDEX1 <> ITS COMPARE
* THIS IS EXECUTED BEFORE THE JUMP. IT WILL JUST LOAD THE CONDITION REGISTER
* WITH THE NEXT CONDITION TO BE TESTED.
```
        TC AD •    •   •           ML2Y               MAR         NOP
        TR •   •   •   •     •                 F0    BRG0  F1    BRG1
```
* ON INDEX REGISTER 1 NOT EQUAL TO ITS COMPARE, JUMP TO BEGINNING OF MULTIPLY
* ROUTINE.
```
        TC •   •   •   •           $ 0001              TOBA        T1BA
        TR •   •   •   •     •                 TF1N  E0    NOP   NOP
```
*   GET ADDRESS OF JTH ITEM IN THE DATA VECTOR.
```
        TR •   •   •   •     AC0                A1    TFOD  A0    TF1D
        TR •   •   •   •     AC0                A0    T0RA  A0    T1RA
```
* THE ABOVE WAS A CHANGE TO INSURE THAT THE PROGRAM WORKS, THIS IS KEPT.
* THIS WILL UPDATE THE ADDRESS FOR THE NEXT ROUND, STORE IT, AND POINT TO THE
* ITEM IN QUESTION.
```
        TR •   IN2 •   •     •                 TFOC  E0    TF1C  E1
```
* THIS WILL LOAD THE MULTIPLIER FOR THE SECOND MULTIPLY INTO THE E0-E1 REG-
* ISTER PAIR. SIMULTANEOUSLY, THIS WILL ZERO THE TEMP FILE POINTERS. THEY
* WILL NOW POINT TO THE LOCATION OF THE ACCUMULATOR.
```
        TR •   •   •   •     •                 BSR0  G1    BSR1  G0
        TC •   •   SR •     FPMR                       MAR         NOP
        TR •   •   •   •     G                  A0    G1    A1    G0
```
* THIS IS JUST A SUBROUTINE JUMP TO THE FLOATING POINT MULTIPLY ROUTINE.
* F=EXG
```
        TC •   •   SR •     FPAR                       MAR         NOP
        TR •   •   •   •     •                 TFON  BRG0  TF1N  BRG1
```
*   F=F+BRG. THIS CALCULATES THE SUBTOTAL OF THE MATRIX MULTIPLY.
```
        TR •   •   •   •     •                 F0    TFON  F1    TF1N
        TC •   •   •   •           $ 0002              TOBA        T1BA
```
* THE ABOVE TWO STEPS LOAD THE SUB TOTAL INTO THE TEMPORARY FILE LOCATION
* ZERO. IT THEM RESETS THE READ AND WRITE POINTERS OF THE TEMPORARY FILE TO
* LOCATION TWO.
```
        TC •   •   •   •           $ 0004              NOP         ICR2
        TC •   •   •   •           $ 0010              NOP         CMR3
```
*   THE 0010 TESTS FOR INDEX2 # ITS COMPARE.
* THIS WILL DO A TEST FOR INDEX 0 NOT EQUAL TO ITS COMPARE REGISTER.
```
        TC AD CL1 •    •           ML2Y               MAR         NOP
        TC •   •   •   •           $ 0000              BRG0        BRG1
        TR •   •   •   PUGL •                   •     •     MULT  MCR3
```

```
*       THE PROGRAM WILL NOW FALL THROUGH TO THE OTPT SECTION OF THE PROGRAM.
* THIS IS THE OUTPUT ROUTINE. *
        TC *    *  *   *          $ 0000           TOBA           TIBA
        TC *    * *   *           $ 0026           LOAD           L1AD
        TR *    *   *   *     *                    TFIN   G1   TFON   GO
        TC *    *   *   *         FCMP             MAR            *
        TR *    *   *   *     *                    LFON   CO   LFIN   E1
FINL TR *    *   *  PUPU *                         MULT   PRGO   *    *
        TC *    *   *   *         $ 0000           TORA           TIRA
OTPT IO *    *   *   *   *                    *    TFON   DS   W   *  *  *  *
        IO *    *   *   *   *                      TFIN   *    DS   W   *  *  *  *
        TC *    *   *   *         WAIT             MAR            NOP
*       THIS IS THE FLOATING POINT ADDITION ROUTINE. 9/4/79,  3:45:00.
FPAR TR *    *   *   *   *                         BSR1   G1   F1   E1
        SH UNS CLO *   *     *                     LZIN   NZIN   LZIN   S
        SH *    *   *   *         .                RZIN   NZIN   RZIN   S
*       THIS WILL STRIP THE SIGN OF THE MANTISSA AND SAVE IT FOR FUTURE USE.
        TR *    *   *   * .                   *    *      BSRO   ICRO
        TC TNN *    *   *        $ 0000             *                  CMRO
        TC TNN *    *   *        $ 0010             *                  CMR1
        TR TNN *   JP *    *                        *      *    *    *
        TR TNN *   DF *    *                        *      *    *    *
        TR *    *   *   *         ZRO              AO     *    A1   CMR1
*       THIS WILL COMPARE THE BRG TO ZERO, IF IT IS, RETURN.
        TR *    *   *   *         ZRO              AO     EO   A1   GO
*       THIS WILL ZERO THE REGISTERS TO PREVENT SPURIOUS RESULTS.
        TR *    *   *   *         E-G              AO     NOP  A1   ICRO
*       IF |E|<|C|, THE PROGRAM WILL REVERSE THE NUMBERS AND CONTINUE.
*       SINCE ADDITION IS COMMUTATIVE, THIS SHOULD NOT AFFECT THE RESULTS.
        TR *    *   *   *         XOR              A1     EO   AO   NOP
        TC *    *   *   *         $ 0080            *                GO
        TR *    *   *   *         AND              A1     *    AO   GO
        TC *    *   *   *         $ 8000           EO               *
        TR *    *   *   *         E-G              A1     *    AO   GO
        TC *    *   *   *         $ 0010            *                CMRO
        TC FNN *    *   *         NSH                     MAR       *
*       IF THE EXPONENT ON ONE OF THE TWO NUMBERS IS LESS THAN ZERO
*       AND THE OTHER IS NOT, SUBTRACTION TO YIELD THE NUMBER OF SHIFTS
*       WILL NOT YIELD THE CORRECT ANSWER, AND THUS SPECIAL HANDLING
*       MUST BE ADDED TO COMPENSATE FOR THIS PROBLEM. THE WAY THAT THIS
*       ROUTINE HANDLES THE PROBLEM IS IT EXCLUSIVE ORS THE TWO NUMBERS
*       TOGETHER AND THEN STRIPS OFF EVERYTHING BUT THE SIGN BIT.  THIS
*       IS THEN SUBTRACTED FROM A CONSTANT (FOR SPEED).THE CONSTANT IS
*       8000, THUS IF THERE IS A 1 IN THE SIGN POSITION, THE RESULT WILL
*       NOT BE NEGATIVE, INDICATING THAT THE CORRECTION MUST TAKE PLACE.
        TC TNN *    *   *         $ 0020            *          *     CMRO
        TR *    *   *   *         E-G              A1     G1   *     *
*       THIS WILL TEST FOR E-G->G NEGATIVE. THIS IS TO INSURE THAT |BRG| >=
*       |F|, SIMPLIFYING THE ALGORITHM GREATLY.
        TC TNN *    *   *         SWAP                    MAR        NOP
*       THIS INVOLVES THE SWAP ROUTINE THAT WILL FORCE THE ABOVE TO BE TRUE.
        TR *    *   *   *         ZRO               *     *    AO    CMRO
        TC *    *   *   *         $ 0010            *               CMR1
* THE ZEROES THAT ARE LOADED INTO CONDITION MASK REGISTER O TELL
* THE MACHINE NOT TO CHECK FOR ANY OF THE CONDITIONS REPRESENTED.
* THE 0010 LOADED INTO CMR1 TELL THE MACHINE TO CHECK FOR THE COMPARE
* REGISTER GREATER THAN INDEX REGISTER ONE.  IN THIS CASE, THIS WILL
* DETERMINE WHETHER THE TWO NUMBERS ARE EQUAL OR EQUAL AND OPPOSITE IN
* MAGNITUDE AND SIGN.
        TR *    *   *   *         ZRO              AO     G1   A1    E1
```

```
       TC TNN •   •   •      EGUL                      MAR        NOP
    •  IF THEY ARE, THE PROGRAM WILL JUMP TO A SPECIAL ROUTINE.
    •  BY THIS POINT IN THE PROGRAM, |E|>|G|.
       TR •   •   •   •      •              BSRO   EO   FO    GO
       TC •   •   •   •      $ 0010                 •          IDXO
       TC •   •   •   •      $ 0020                 •          CMR1
       TC TNN •   •   •      RTNF                  MAR         NOP
    •  IF THE NUMBER OF SHIFTS REQUIRED > 16, RETURN THE VALUE IN THE
    •  F REGISTER.
       TR •   •   •   •      ZRO               A1   G1   AO    CMR1
       TC •  CLO •   •       $ 0001                 •          CMR3
    •  THIS LOADS THE DATA TO BE PROCESSED AND IT PROGRAMS THE CPU TO CHECK
    •  FOR REGOWINDYO. THIS IS REPRESENTED BY A ONE IN THE FIRST POSITION. THIS
    •  CHECK IS INVOLKED BY THE AD COMMAND.
 SHFT  SH •   INO •   •      •         (        RZIN  NZIN  NZIN  S
       TC AC •   •   •       SHFT               .     MAR          NOP
  • INDEX REGISTER CONTAINS THE AMOUNT BY WHICH G>E, (THE NUMBER OF ORDERS
  • OF MAGNITUDE. THIS ROUTINE SHIFTS E TO THE RIGHT UNTIL THE TWO ORDERS OF
  • MAGNITUDE ARE EQUAL.
       TC •   •   •   •      $ 0000              G1                •
       TC •  .•   •   •      $ 0020              •          CMRO
       TC FPN •   •   �'      GPOS              MAR         NOP
    •  IF G1 >= 0, ITS SIGN IS TAKEN TO BE POSITIVE, AND THE NUMBERS ARE
    •  HANDELED IN A CORRESPONDING MANNER.
    •  BY THIS POINT, G MUST BE NEGATIVE.
       TC •   •   •   •      $ 0002                 •          CMRO
       TC TPN •   •   •      SSGN                  MAR         NOP
    •  IF E IS NEGATIVE, AND G IS NEGATIVE, THE SIGNS ARE THE SAME AND THE
    •  TWO NUMBERS ARE JUST ADDED AND ONE OF THE SIGNS IS PRESERVED.
       TC •   •   •   •      •              •.   •    •     •
    •  AT THIS POINT, |G|>|E|, THE RESULTANT SIGN WILL BE THAT OF G.
    •  WITHOUT REGARD TO SIGN, THE RESULT WILL BE THE OLD SIGN OF G
    •  PLUS |G-E|.
 DSGN  TR •   •   •   •      EN                AO   EO   •     •
       TR •   •   •   •      E+1               AO   EO   •     •
       TR •   •   •   •      ADD               F1   EO   AO    GO
    •  THIS CALCULATES G-E.
       TC •   •   •   •      $ 0010                 •          CMRO
    •  IF THE RESULT IS >= ZERO, THERE IS NOT A ONE IN TTHE FIRST BIT POSITION,
    •  SO THE NUMBER IS NOT NORMALIZED, AND MUST BE SHIFTED UNTIL THERE APPEARS
    •  A '1' IN THE FIRST BIT POSITION.
 NORM  TR FNN •   •   •      E-1               AO   EO   •     •
       TC FNN •   •   •      NORM                  MAR         NOP
       SH FNN •   •   •      •              NZIN  NZIN  LZIN  S
    •  THIS ROUTINE NORMLIZES THE DATA
       TR •   •   •   •   .  G                AO   FO   •     •
       TC •   •   •   •      $ 80FF                •          GO
       TR •   •   •   •      AND               •    •    AO    F1
       TC •   •   •   •      $ 0002                •          CMRO
       SH •   •   •   •      •              NZIN  LZIN  NZIN  S
       SH TPN •   •   •      •              NZIN  ROIN  NZIN  S
       SH FPN •   •   •      •              NZIN  RZIN  NZIN  S
       TC •   •  JP •        $ 0000                •     -     •
       TC •   •  DF •        $ 0000                •          •
    •  THIS ROUTINE SETS THE SIGN TO THE SORRECT SIGN AND RETURNS TO THE CALLIN
    •  ROUTINE.
 GPOS  TC TPN •   •   •      DSGN                  MAR         NOP
       TR •   •   •   •      •              •    •    •     •
    •  BEFORE THE JUMP TO GPOS, THE CONDITION REGISTER WAS SET TO CHECK FOR
    •  E<0. IF IT IS, THE SIGNS ARE OPPOSITE AND THE DATA IS TREATED
```

```
•    CORRESPONDINGLY.
•    BY DEFAULT, BOTH G AND E HAVE .THE SAME SIGN, SO THE RESULTS ARE JUST
•    ADDED.
     TR  •   •   •   •   •                        •     •     F0    G0
SSGN TR  •   •   •   •   ADD                       A1    G1    A0    G0
     SH  •   •   •   •   •                         NZIN  NZIN  RCIR  S
     TC  •   •   •   •   $ 0010                          NOP         CMR0
•    THIS CHECKS FOR A CARRY OUT OF HE MSB, INDICATING NORMALIZATION IS
•    NECESSARY.
     TR TNN •   •   •   G                          F1    E0    •     •
     TR TNN •   •   •   E+1                         A0    E0    •     •
     TR TNN •   •   •   G                          A0    F0    •     •
     TC TNN •   JP  •          $ C0FF                          •           G0
     TR TNN •   DF  •   AND                         •     •     A0    F1
•    IF IT IS, THEN THE NUMBER IS NORMALIZED AND THE SUBROUTINE RETURNS.
     SH  •   •   JP  •   •                          NZIN  NZIN  LCIR  S
     TR  •   •   DF  •   G                          A0    F0    •     •
•    THIS ROUTINE EXCHANGES THE TWO REGISTERS INVOLVED SO THAT |G|>|E|
SWAP TR  •   •   •   •   •                          F1    G1    F0    G0
     TR  •   •   •   •   •                          BSR0  F0    BSR1  F1
     TC  •   •   •   •          FPAR                 MAR         NOP
     TR  •   •   •   •   G                           A0    BRG0  A1    BRG1
•    THIS CALLS THE ORIGINAL ROUTINE.
•    THIS IS THE ACTION TAKEN WHEN THE ROUTINES HAVE THE SAME MAGNITUDE.
EQUL TC  •   •   •   •   $ 0000                          NOP         CMR1
     TC  •   •   •   •   $ C002                          NOP         CMR0
     TC FPN •   •   •   EPOS                         MAR         NOP
     TC  •   •   •   •   $ 0020                          NOP         CMR0
     TC TP  •   •   •   SSGN                         MAR         NOP
     TC FP  •   •   •   SWAP                         MAR         NOP
     TC  •   •   •   •   $ 0000                          NOP         NOP
EPOS TC FP  •   •   •   SSGN                         MAR         NOP
     TC  •   •   •   •   $ 0100                          NOP         CMR0
     TR  •   •   •   •   E=G                         NOP   NOP   NOP   NOP
     TC TN  •   •   •   ZAPP                         MAR         NOP
     TC  •   •   •   •   $ 0020                          NOP         CMR0
     TR  •   •   •   •   E-G                         A1    G1    A0    NOP
     TC TN  •   •   •   DSGN                         MAR         NOP
     TC  •   •   •   •   $ 0000                         G1          CMR0
     TC  •   •   •   •   DSGN                         MAR         NOP
     TR  •   •   •   •   •                           NOP   NOP   BSR1  F1
     TC  •   •   •   •        $ 0000                  •           •
ZAPP TR  •   •   JP  •   ZRO                         A0    F0    A1    F1
     TC  •   •   DF  •   $ 0000                            NOP         CMR0
•    THIS ROUTINE HANDLES NUMBERS THAT HAVE DIFFERENT EXPONENTIAL SIGNS.
NSH  TC  •   CLO •   •          $ 0010                     NOP         CMR0
•    BUG IN ASSEMBLER.  NULL LINE WILL NOT BE ASSEMBLED.  BY THIS POINT
•    IN THE PROGRAM, THE EXPONENT ON ONE OF THE TWO NUMBERS MUST
•    BE LESS THAN ZERO. THIS PART OF THE ROUTINE WILL FORCE THE NEGATIVE
•    PART TO BE STORED IN BRG REGISTER. SINCE A SWAP CAN TAKE PLACE,
•    ALL THE ORIGINAL FLAGS MUST BE RESET IN THE EVENT OF ASHIFT.
     TR  •   •   •   •   G                           A1SW  •     A0RZ  G0
     TC TNN •   •   •          GLZ                    MAR         NOP
•    THE G/BRG REGISTER CONTAINS THE NEGATIVE EXPONENT, NO SWAP NEEDED.
     TR FNN •   •   •   •                            F1    G1    F0    G0
     TR  •   •   •   •   •                            BSR0  F0    BSR1  F1
     TR  •   •   •   •   G                            A0    BRG0  A1    BRG1
     TR  •   •   •   •   •                            BSR1  G1    F1    E1
     SH UNS •   •   •   •                             LZIN  NZIN  LZIN  S
     SH  •   •   •   •   •                             RZIN  NZIN  RZIN  S
```

```
*      THIS SWAPS THE .TWO NUMBERS AND RESETS ALL THE FLAGS NEEDED PY THE
*      REST OF THE ROUTINE.
GLZ    TC *    *    *    *         $ 0000.              . EO            GO
       TR *    *    *    *    E-G                  A1SW *       AOSW    GO
       TR *    *    *    *    G                    A1RZ NOP     AORS    ICRO
*.     CALCULATE THE NUMBER OF SHIFTS NEEDED, IF IT IS < 0, IT IS
*      ACTUALLY > 80 (16), SO RETURN THE VALUE IN THE F REGISTER.
*
       TC TNN *    *    *         RTNF                 MAR             NOP
       TR *    *    *    *    G                    AOSW EO      A1RZ    NOP
       TC *    *    *    *         $ 0010              *               GO
       TR *    *    *    *    E-G                  A1   NOP     AO      GO
       TC FNN *    *    *         RTNF                 MAR             NOP
*      IF THE NUMBER OF SHIFTS REQUIRED IS > 16, RETURN THE DATA IN THE
*      F REGISTER.
       TC *    *    *    *         $ 0000              *               CMRO
       TC *    *    *    *         $ 0000              *               E1
       TR *    *    *    *    *                    BSRO EO      FO      GO
       TC *    *    *    *         SHFT                MAR             NOP
       TC *    *    *    *         $ 0001              *               CMR3
*      PREPAIR TO SHIFT THE DATA AND RETURN TO SHIFTING ROUTINE.
RTNF   TR *    *    JP *    *                      *    *       *       *
       TR *    *    DF *    *                      *    *       *       *
*      RETURN THE CONTENTS OF THE F REGISTER.
* THIS IS JUST FOR A BREAK POINT AND IT IS TO BE REMOVED WHEN*
* THE PROGRAM IS ACTUALLY INSERTED INTO THE CODE.                *
FCMP   TC UNS *    *    *         $ 0000              TOBA            TIBA
* THIS ACCEPTS THE DATA IN THE E REGISTER AND G REGISTER AS  *
* INPUTS. INITIALLY, THE PROGRAM STORES THE ORIGINAL DATA IN *
* TEMPORARY FILE. THE E REGISTER GOES IN LOCATION 0 AND THE  *
* G REGISTER GOES IN LOCATION 1. THE FOLLOWING WILL ALSO     *
* STRIP OFF THE SIGN BIT                                     *
       TR *    *    *    *    E                    AO   TFOU    A1      TF1U
       TR *    *    *    *    G                    AO   TFOU    A1      TF1U
* THIS ROUTINE STRIPS OFF THE SIGN BIT. THE CORRECT SIGN BIT *
* IS SAVED IN THE PAST REGISTER.                             *
       SH *    *    *    *    *                    LZIN NZIN    LZIN    S
       SH *    *    *    *    *                    RZIN NZIN    RZIN    S
       TR *    *    *    *    E                    A1SW EO      AOSW    E1
       TR *    *    *    *    G                    AOSW G1      A1SW    GO
* THE 0002 IN CMRG WILL CHECK FOR E1 NEGATIVE. THIS IS DONE  *
* IN THE PAST SENSE. IF E1<0, THEN JUMP TO THE ROUTINE THAT  *
* WILL HANDLE THAT CASE.                                     *
       TC *    *    *    *         $ 0002              NOP             CMRO
       TC TPN *    *    *         EMNG                 MAR             *
* BY THIS POINT, THE E REGISTER MUST NOT BE NEGATIVE (>=0)   *
* THE 0020 IN THE CMRO WILL TEST FOR G<0. IF G<0, E IS THE   *
* GREATER OF THE TWO NUMBERS. IF NOT, THEY ARE BOTH >= 0.    *
       TC *    *    *    *         $ 0020              *               CMRO
       TC TPN *    *    *         EGRT                 MAR             *
* THIS WILL DETERMINE IF THERE IS A DIFFERENCE IN EXP SGN.   *
       TC TPN *    *    *         $ 0000              *               *
       TC TNN *    *    *         GXNG                 MAR             *
* THIS WILL DO A JUMP IF THE SIGN OF G IS 1, OR G NEGATIVE   *
* IN THE EXPONENT PORTION.                                   *
       TC *    *    *    *         $ 0002              *               CMRO
       TC TNN *    *    *         GGRT                 MAR             *
* BY HERE, THE EXPONENT OF G IS POSITIVE. IF THE EXPONENT OF *
* E IS NEGATIVE, BOTH MANTISSAS BEING POSITIVE, E<G          *
       TR FNN *    *    *    E-G                  AO   EO      A1      E1
```

```
      TC TNN •  •  •           GGRT,       MAR         •
      TC FNN •  •  •           $ 0000           FO         F1
      TC •  •  JP •            $ 0000           TOBA       T1BA
      TR •  •  DF •     •                  TFON  EO   TF1N  E1
• SINCE BOTH EXPONENTS AND MANTISSAS ARE NONNEGATIVE, THIS  •
• ROUTINE CALCULATES E-G, EXPONENTS IN THE HOBP AND MANTISSAS•
• IN THE LOBPS. IF THE RESULT IS < 0, G>E, ELSE RETURN E.   •
GGRT  TC •  •  •  •           $ 0001           TOBA       T1BA
      TC •  •  JP •           $ 0001           FO         F1
      TR •  •  DF •     •                  TFON  EO   TF1N  E1
• IF F1>=0, E>G, RETURN TF[1]                               •
EGRT  TC •  •  •  •           $ 0000           FO         F1
      TC •  •  JP •           $ 0000           TOBA       T1BA
      TR •  •  CF •     •                  TFON  EO   TF1N  E1
• THIS IS THE SECTION OF THE PROGRAM THAT IS CALLED IF E IS  •
• NEGATIVE. (MANTISSA)                                       •
EMNG  TC •  •  •  •           $ 0020           •          CMRO
      TC FPN •  •  •          GGRT        MAR         NOP
• THIS SECTION DOES THE COMPARE IF BOTH THE OPERANDS ARE < 0 •
• THIS WILL DETERMINE IF THERE IS A DIFFERENCE IN EXP SGN.   •
NCMP  TC •  •  •  •           $ 0000           •          •
      TC TNN •  •  •          GBNG        MAR         •
• THIS WILL DO A JUMP IF THE SIGN OF G IS 1, OR G NEGATIVE   •
• IN THE EXPONENT PORTION.                                   •
      TC •  •  •  •           $ 0002           •          CMRO
      TC FNN •  •  •          NNPP        MAR         •
• BY HERE, THE EXPONENT OF G IS POSITIVE. IF THE EXPONENT OF •
• E IS NEGATIVE, BOTH MANTISSAS BEING NEGATIVE, E>G          •
      TC TNN •  •  •          $ 0000           FO         F1
      TC •  •  JP •           $ 0000           TOBA       T1BA
      TR •  •  DF •     •                  TFON  EO   TF1N  E1
• THE ABOVE WILL RETURN E                                    •
GBNG  TC TNN •  •  •          EBNG        MAR         •
• BOTH G'S EXPONENT AND SIGN ARE NEGATIVE. IF TRUE, THE SAME •
• HOLDS TRUE FOR E. IF THIS IS FALSE, RETURN G.              •
      TC •  •  •  •           $ 0001           TOBA       T1BA
      TC •  •  JP •           $ 0001           FO         F1
      TR •  •  DF •     •                  TFON  EO   TF1N  E1
EBNG  TR •  •  •  •     E-G             AO   EO   A1   E1
      TC FNN •  •  •          GGRT        MAR         •
      TC •  •  •  •           $ 0000           FO         F1
      TC •  •  JP •           $ 0000           TOBA       T1BA
      TR •  •  DF •     •                  TFON  EO   TF1N  E1
•  BOTH THE MANTISSA AND THE EXPONENT OF BOTH E AND G ARE    •
• LESS THAN ZERO.  CALCULATE E-G. IF RESULT POSITIVE, G>E    •
GXNG  TC FNN •  •  •          EGRT        MAR         NOP
      TC •  •  •  •           $ 0000           •          •
TEST  TR •  •  •  •     E-G             AO   EO   A1   E1
      TC FNN •  •  •          EGRT             MAR         NOP
      TC TNN •  •  •          GGRT             MAR         NOP
      TC •  •  •  •           $ 0000           •          •
• AT THIS (PRECEEDING LINE) BOTH E AND G ARE POSITIVE. THE   •
• SIGN OF THE EXPONENT OF G IS NEGATIVE. IF THE SIGN OF THE  •
• EXPONENT OF E IS POSITIVE, E>G, HENCE RETURN E.            •
NNPP  TR •  •  •  •     E-G             AO   EO   A1   E1
      TC TNN •  •  •          EGRT             MAR         NOP
      TC FNN •  •  •          GGRT             MAR         NOP
      TC •  •  •  •           $ 0000           •          •
•
```

```
         TC TNN •   •  •           GGRT.         MAR          •
         TC FNN •   •  •           $ 0000              F0          F1
         TC  •   •   JP •          $ 0000              TOBA        T1BA
         TR  •   •   DF •    •                   TFON  E0    TFIN  E1
• SINCE BOTH EXPONENTS AND MANTISSAS ARE NONNEGATIVE, THIS    •
• ROUTINE CALCULATES E-G, EXPONENTS IN THE MOBP AND MANTISSAS•
• IN THE LOBPS. IF THE RESULT IS < 0, G>E, ELSE RETURN E.     •
GGRT TC •   •   •  •              $ 0001              TOBA        T1BA
         TC  •   •   JP •         $ 0001              F0          F1
         TR  •   •   DF •    •                  TFON  E0    TFIN  E1
• IF F1>=0, E>G, RETURN TF[1]                                 •
EGRT TC •   •   •  •              $ 0000              F0          F1
         TC  •   •   JP •         $ 0000              TOBA        T1BA
         TR  •   •   DF •    •                  TFON  E0    TFIN  E1
• THIS IS THE SECTION OF THE PROGRAM THAT IS CALLED IF E IS   •
• NEGATIVE. (MANTISSA)                                        •
EMNG TC •   •   •  •              $ 0020              •           CHRO
         TC FPN •   •  •          GGRT                MAR         NOP
• THIS SECTION DOES THE COMPARE IF BOTH THE OPERANDS ARE < 0  •
• THIS WILL DETERMINE IF THERE IS A DIFFERENCE IN EXP SGN.    •
NCMP TC •   •   •  •              $ 0000              •           •
         TC TNN •   •  •          GBNG               MAR          •
• THIS WILL DO A JUMP IF THE SIGN OF G IS 1, OR G NEGATIVE    •
• IN THE EXPONENT PORTION.                                    •
         TC  •   •   •  •         $ 0002              •           CHRO
         TC FNN •   •  •          NNPP               MAR          •
• BY HERE, THE EXPONENT OF G IS POSITIVE. IF THE EXPONENT OF  •
• E IS NEGATIVE, BOTH MANTISSAS BEING NEGATIVE, E>G           •
         TC TNN •   •  •          $ 0000              F0          F1
         TC  •   •   JP •         $ 0000              TOBA        T1BA
         TR  •   •   DF •    •                  TFON  E0    TFIN  E1
• THE ABOVE WILL RETURN E                                     •
GBNG TC TNN •   •  •              EBNG               MAR          •
• BOTH G'S EXPONENT AND SIGN ARE NEGATIVE. IF TRUE, THE SAME  •
• HOLDS TRUE FOR E. IF THIS IS FALSE, RETURN G.               •
         TC  •   •   •  •         $ 0001              TOBA        T1BA
         TC  •   •   JP •         $ 0001              F0          F1
         TR  •   •   DF •    •                  TFON  E0    TFIN  E1
EBNG TR •   •   •  •      E-G                   A0    E0    A1    E1
         TC FNN •   •  •          GGRT               MAR          •
         TC  •   •   •  •         $ 0000              F0          F1
         TC  •   •   JP •         $ 0000              TOBA        T1BA
         TR  •   •   DF •    •                  TFON  E0    TFIN  E1
•  BOTH THE MANTISSA AND THE EXPONENT OF BOTH E AND G ARE     •
• LESS THAN ZERO.  CALCULATE E-G. IF RESULT POSITIVE, G>E     •
GXNG TC FNN •   •  •              EGRT               MAR         NOP
         TC  •   •   •  •         $ 0000              •           •
TEST TR •   •   •  •      E-G                   A0    E0    A1    E1
         TC FNN •   •  •          EGRT               MAR         NOP
         TC TNN •   •  •          GGRT               MAR         NOP
         TC  •   •   •  •         $ 0000              •           •
• AT THIS (PRECEEDING LINE) BOTH E AND G ARE POSITIVE. THE    •
• SIGN OF THE EXPONENT OF G IS NEGATIVE. IF THE SIGN OF THE   •
• EXPONENT OF E IS POSITIVE, E>G, HENCE RETURN E.             •
NNPP TR •   •   •  •      E-G                   A0    E0    A1    E1
         TC TNN •   •  •          EGRT               MAR         NOP
         TC FNN •   •  •          GGRT               MAR         NOP
         TC  •   •   •  •         $ 0000              •           •
#
```

APPENDIX 2C3
FLEXIBLE PROCESSOR SYSTEM SIMULATOR

I. Simulator Flowcharts

    A. Setting Up Simulation

    B. Input FP# and Operation to be Performed

    C. Execute Single Execution Step

    D. Read and Modify Register or Program Memory Content

    E. Subroutine "Exec" for Executing Single Instructions.
   Subroutine "Skip" for Executing Sequence of Instructions.

II. Simulator Displays

    A. Simulator Output Display

    B. Simulator Display of Temporary File

    C. Simulator Display of Large File

I A. SETTING UP SIMULATION
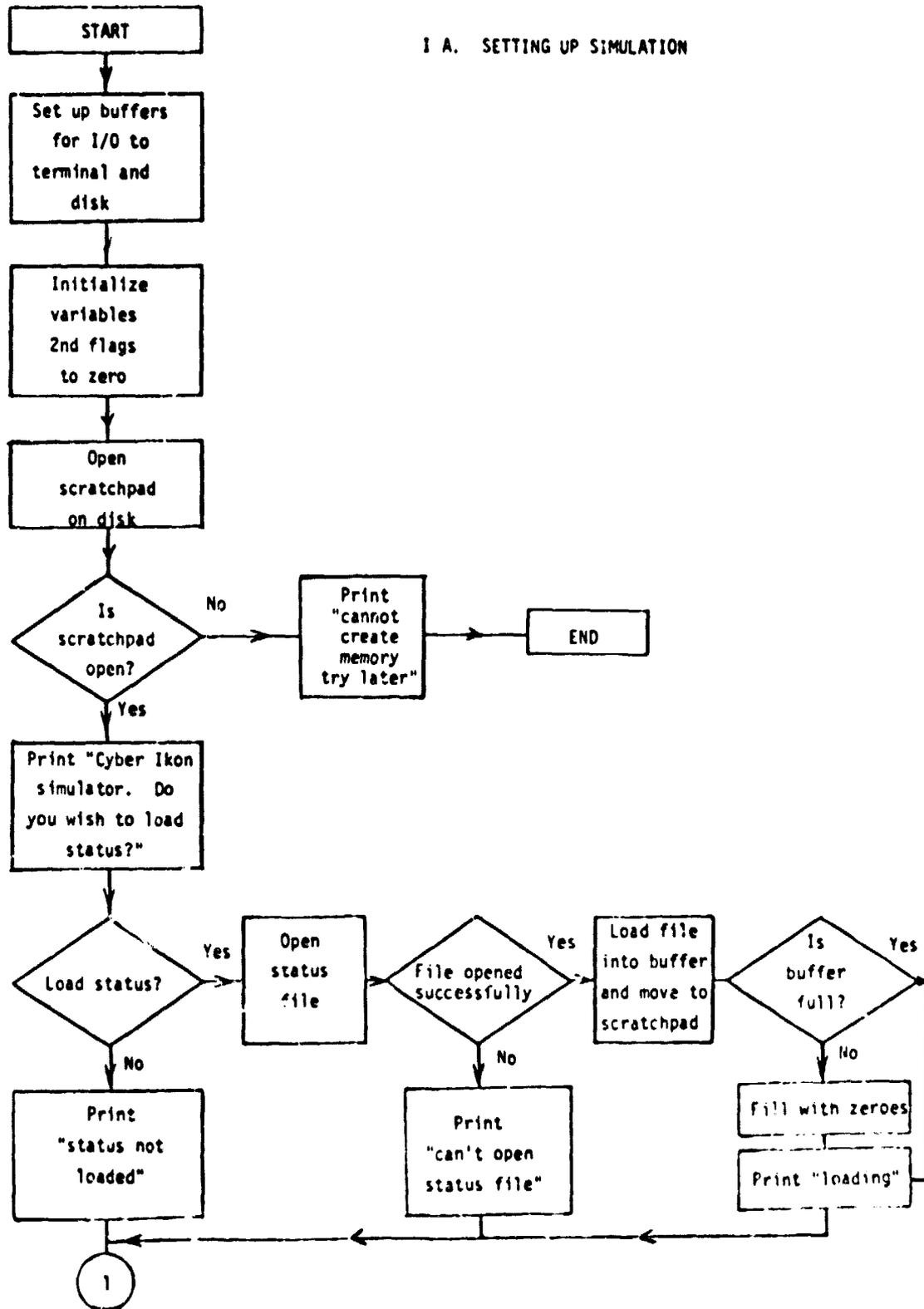
I B. INPUT FP# AND OPERATION TO BE PERFORMED
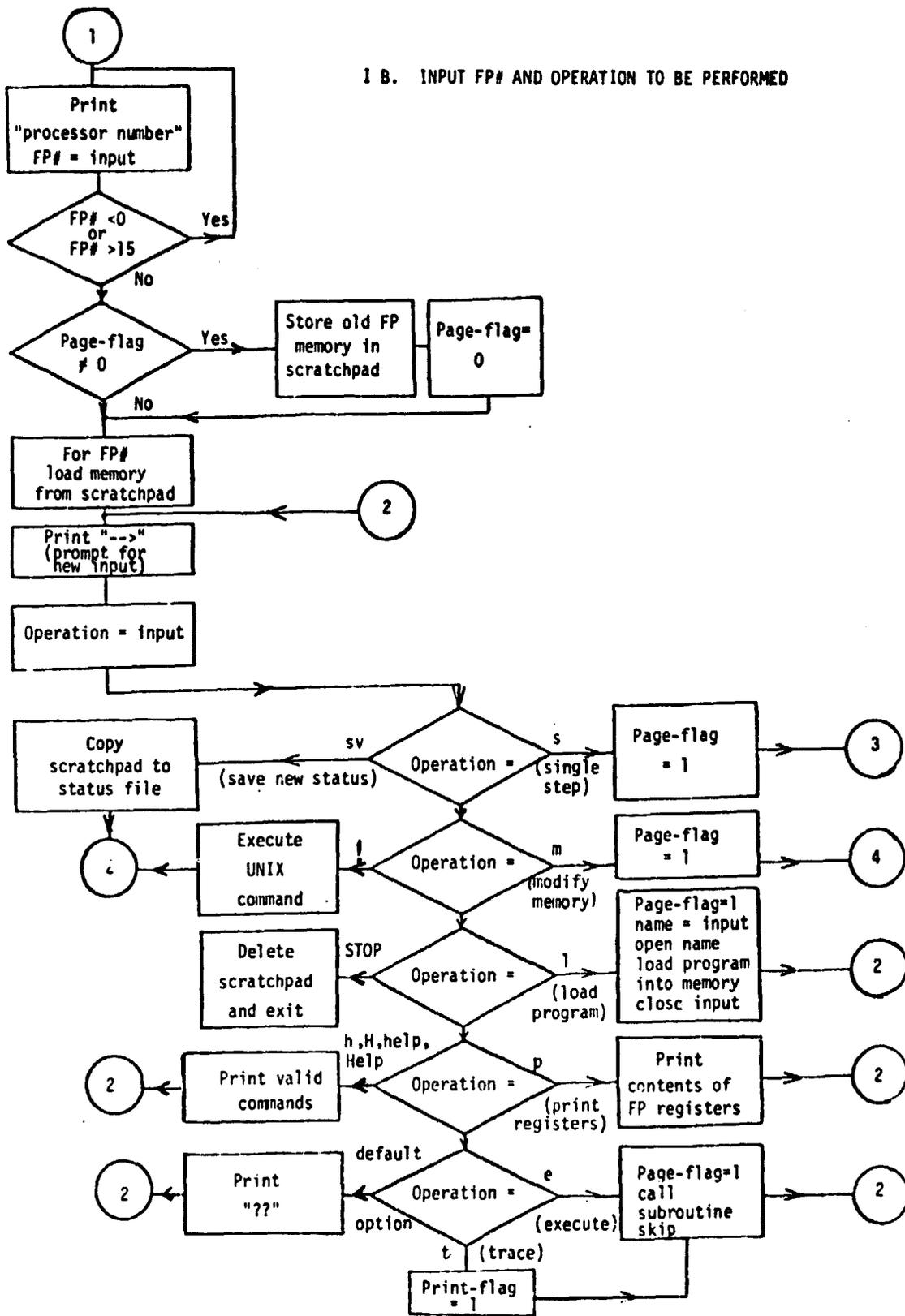
I C. EXECUTE SINGLE EXECUTION STEP

I D. READ AND MODIFY REGISTER

OR PROGRAM MEMORY CONTENT

(4)

Print "Register"

reg = input

Is
reg = "#" or
reg invalid?   Yes (2)

No

Is
named register
or file or pro-
gram mem.?   Yes   Print "Index"

index = input

No

index = 0

Is
index
invalid?   No   Print
"no action
taken"   (4)

Yes

Set pointer to
start address of
appropriate
memory area and
offset by index

Invalid

Print contents
of register   input =   c   Set register
to input

input =   +,i   index = index+1

-,d

index = index - 1   Is
index
invalid?   Yes   (4)

No
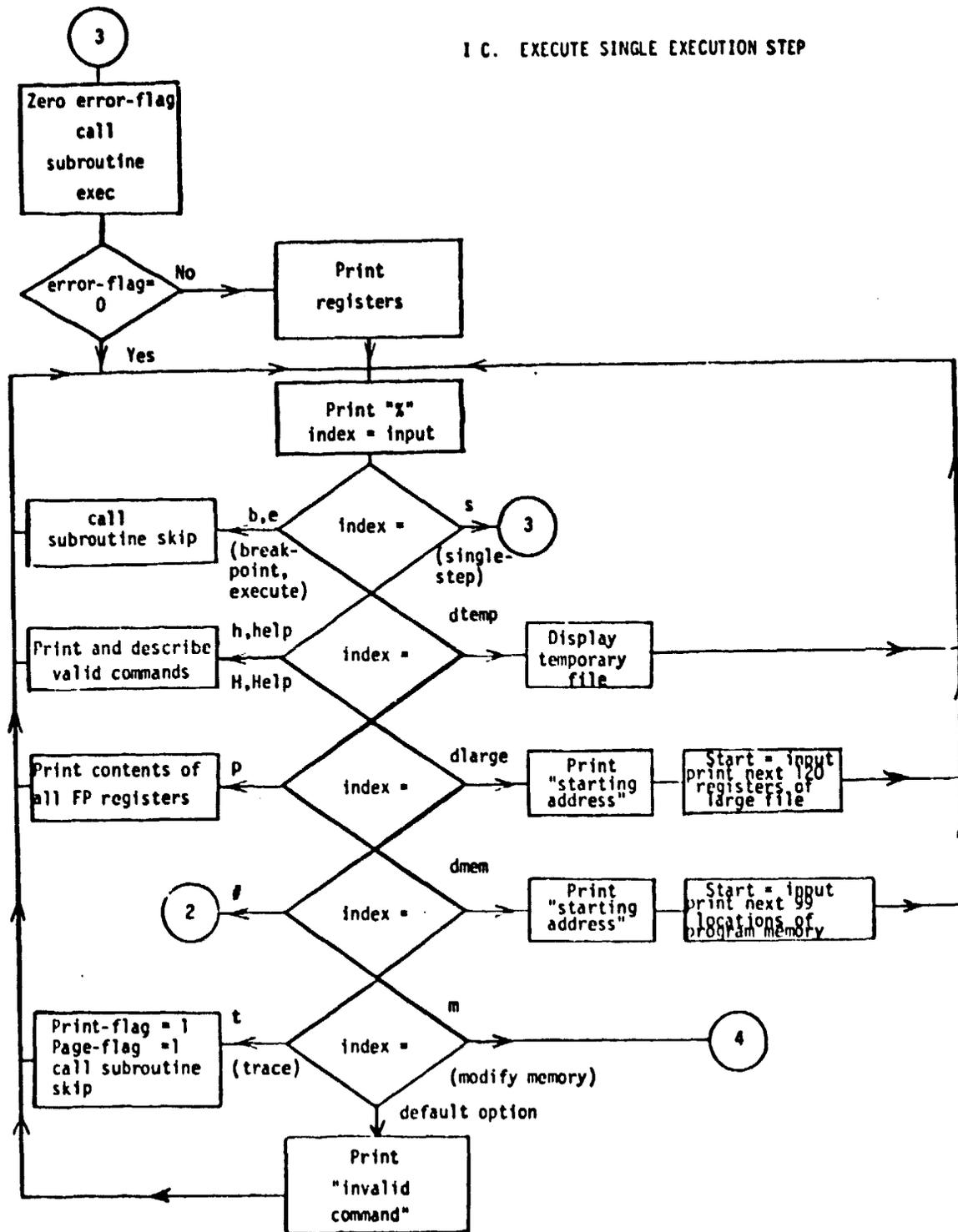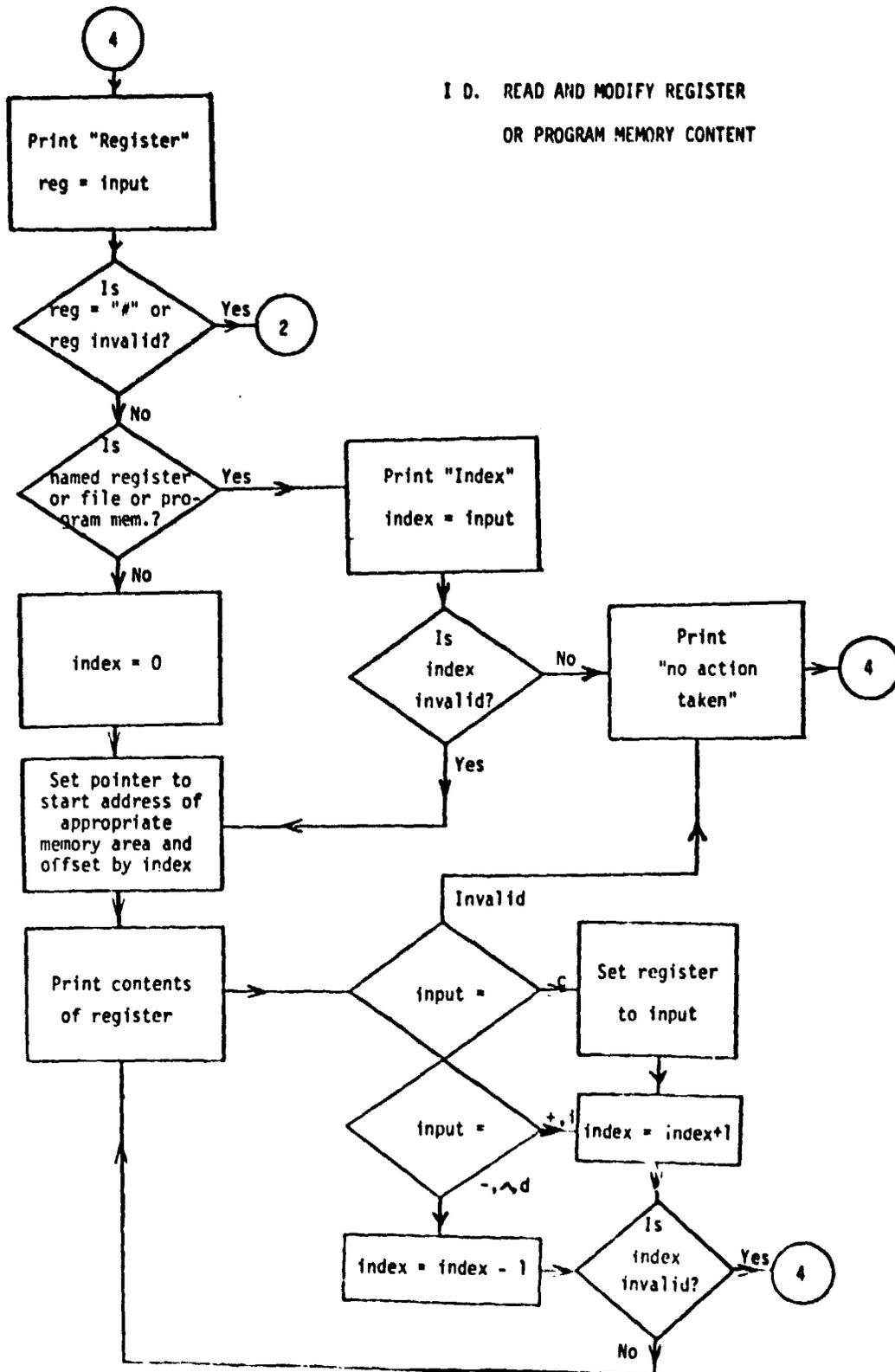
I E. SUBROUTINE "EXEC" FOR EXECUTING SINGLE
INSTRUCTIONS. SUBROUTINE "SKIP" for
EXECUTING SEQUENCE OF INSTRUCTIONS.
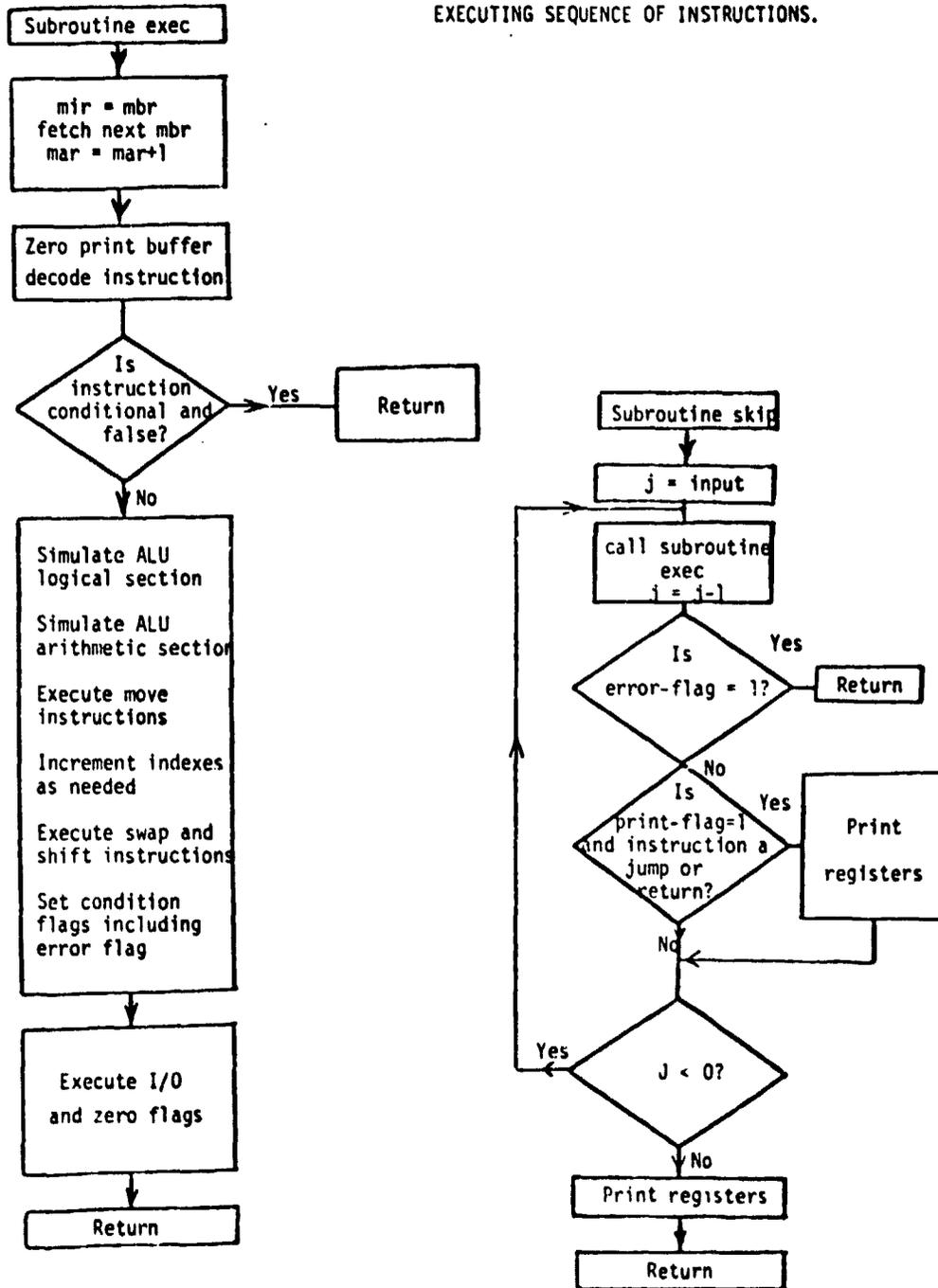
II A. SIMULATOR OUTPUT DISPLAY

```
CYCLES:      MAR:              MIR0: MIR1: MIR2:            tr unn nop np pupu
0000  05f3   016d  0100        0100  0005  ac0   mult brg0 nop nop      E0:    E0:
                                                                        0000   1f2a

IDX0: IDX1: IDX2: IDX3:        ICR0: ICR1: ICR2: ICR3:                  E1:    E0:
0003  0004  0004  0000         0003  0004  0004  0004                   0000   1f2a

CMR0: CMR1: CMR2: CMR3:        MCR0: MCR1: MCR2: MCR3:                  G1:    G0:
0010  0000  0000  0010         0000  0000  0000  9c40                   0000   a5ce

BRG1: BRG0:        IMR0: IMR1: INTR:  MARC:                             F1:    F0:
0000  27d9         0000  0000  0000   0000                              0011   a5ce

NOH:  PAST:        MMT0: MMT1: MMT2:         OUP0: OUP1:                P:     Q:
0014  0014         0000  0000  0000          0000  0000                 a265   c500

IF0A: IF1A:        LF0A: LF1A:  TF0AR: TF0AH: TF1AR: TF1AH:             MULT:
0000  0000         0010  0010   0002   0002   0002   0002               27d9

AINOUT: ORESIN: ARESIN: AROUT:  QINOUT: AQZIN:
0000    0000    0000    0000    0000    0000
0000    0000    0000    0000    0000    0000
0000    0000    0000    0000    0000    0000
0000    0000    0000    0000    0000    0000
```

II B. SIMULATOR DISPLAY OF TEMPORARY FILE

```
temp[0] =        0000  0000
temp[1] =        0001  0001
temp[2] =        0006  d240
temp[3] =        0005  b640
temp[4] =        0006  9400
temp[5] =        0006  c500
temp[6] =        0000  0000
temp[7] =        0000  0000
temp[8] =        0000  0000
temp[9] =        0000  0000
temp[a] =        0000  0000
temp[b] =        0000  0000
temp[c] =        0000  0000
temp[d] =        0000  0000
temp[e] =        0000  0000
temp[f] =        0000  0000
```

II C. SIMULATOR DISPLAY OF LARGE FILE

```
lf[0]  =  8fc0  0005   c000  00fd     b080  8002     c800  00fc
lf[4]  =  c000  00fd   9780  0006     9c80  0001     f880  8001
lf[8]  =  b080  8002   9c80  8001     a440  0004     b580  8000
lf[c]  =  c800  00fc   f880  8001     b580  8000     ff80  0003
lf[10] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[14] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[18] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[1c] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[20] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[24] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[28] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[2c] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[30] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[34] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[38] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[3c] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[40] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[44] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[48] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[4c] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[50] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[54] =  0000  0000   0000  0000     0000  0000     0000  0000
lf[58] =  0000  0000   0000  0000     0000  0000     0000  0000
```

2C3. AMBIGUITY REDUCTION FOR TRAINING SAMPLE LABELING
David A. Landgrebe*

This task, envisioned as a multiyear task, began only 2 months before
the end of this report period. As a result the material presented here
does not provide final results; rather it provides background material,
an approach description, and a discussion of certain aspects of the problem.

1. Introduction and Background

The proper training of the classifier in a remote sensing data analysis
system is one of the pivotal steps to good system performance. The original
method used for training classifiers was to define a set of classes based on
user need, then to choose an adequate set of prelabeled sample pixels of
these classes by which to compute class statistics. Because it was assumed
that the labels would be established by ground observation they were always
assumed perfectly accurate.

However, in some application situations ground observations (or at
least observations from the ground) are not always possible. Thus, cases
arise where the labels associated with training pixels are not entirely
accurate.

In any application situation, there nearly always exists a wide assort-
ment of ancillary information, some of which is subjective in nature, other
objective, which should be able to materially contribute to the accuracy of
such a pixel labeling process. Examples are data about the terrain, weather
and climate, seasonal characteristics and the spatial context of pixels.
The question is what mechanism should be used to incorporate such information
into the labeling process. Thus, the objective for the current work is:

> To devise and evaluate quantitative and objective means for
> optimally arriving at classifier training sets using remotely
> sensed spectral observations together with any other types of
> ancillary data and knowledge which may become available.

1.1 Approach

In selecting an approach to pursue this objective it is important
to note that the ancillary data to be used is varied in type and not well
defined. The information content of such data may be known only somewhat
vaguely a priori, and in some cases it will certainly be difficult to quantify.
Such a situation suggests defining an approach which, instead of being based
on a direct deterministic calculation, might be iterative in nature so as
to provide a "convergence of evidence."

A problem in the field of picture processing with somewhat similar
characteristics is being studied using a method, known as relaxation, which
is iterative in character. It was therefore decided to study this approach
to see if it might be adaptable to the problem at hand. Basically, the idea
would be to use the ancillary information to reduce any ambiguity which might
be associated with a given label on a given pixel. At the out.-et there would
exist an exhaustive list of labels and a set of pixels with a (preliminary)
label association for each. There would be a measure of certainty of the
accuracy of this association in quantitative form. The process would then
be one of utilizing the ancillary information in an iterative fashion to
cause reinforcement of the degree of certainty for the correct label of the
pixel at the expense of all of the incorrect ones.

To begin the work of devising the details of this technique a careful
review of the literature generated so far regarding relaxation methods in
picture processing has been undertaken. A brief outline of this literature
follows.

1.2 Review of Literature

The class of relaxation techniques related to the present investigation
evolved from an algorithm proposed by Rosenfeld et al.[1] in 1976. This
procedure develops (spatial) consistency among sets of objects (such as
pixels) by means of measures of correlations between their labels; con-
sequently spatial context information is provided by a set of correlation
coefficients. Other algorithms vary only slightly from this in structure,

but appear quite different in the means by which they imbed context data into the relaxation process. Zucker and Mohammed [2] have suggested two algorithms one of which is essentailly the same as that of Rosenfeld but in which context data are carried by sets of conditional probabilities rather than correlations. Both this algorithm and that of Rosenfeld derive label estimates on objects during the relaxation process by forming a weighted arithmetic average of the "evidences" provided by neighboring objects.* The second algorithm of Zucker and Mohammed replaces this by a geometric average. There are certain operational characteristics of this variation, however, that recommend it as unsuitable in pixel labeling. More recently Peleg [3] has derived an algorithm that also uses a conditional probability description of context. However, whereas earlier algorithms were derived on heuristic considerations, Peleg's is based upon probabilistic foundations. Yamamoto [1] has recommended a variation on Rosenfeld's original algorithm which has simpler forms of the compatibility coefficients. In addition to using correlations for compatibility coefficients, Peleg and Rosenfeld [12] devise a set of coefficients based upon mutual information considerations.

A quite different approach has been adopted by Faugeras and Berthod [5]. Rather than being based upon an explicit relaxation formula, their scheme establishes a criterion that provides a measure of the consistency of the labeling on a set of objects along with a measure of redundancy. It then determines a final label distribution by optimizing this measure.

The behavior of relaxation labeling processes is not well understood as yet, and consequently there exists considerable interest in trying to develop a theoretical basis by which to describe the various procedures and with which their operational characteristics can be predicted. The first extensive discussion of theoretical issues related to algorithms of the type discussed above seems to be that of Zucker and Mohammed [6], which is

---

*There appear to be two implicit definitions of "neighborhood" used in the literature on probabilistic relaxation labeling, one of which includes the pixel whose label is currently being modified and one which excludes that pixel. Rosenfeld's investigations [1,11] embody the former whereas the studies by Zucker et al.[2,6,7] use the latter. Notwithstanding Rosenfeld's choice, he gives the "current" pixel a low weighting to avoid it dominating the relaxation procedure.

an expanded version of [2]. Later theoretical treatments include Zucker, Leclerc and Mohammed [7], Ullman [8], Haralick, Mohammed and Zucker [9] and Zucker, Krishnamurthy and Haar [10]. In particular, Zucker, Leclerc and Mohammed present a generalized form of algorithm that degenerates to the previous well known procedures in special cases. Moreover Zucker, Krishnamurthy and Haar have recommended methods for accelerating the process.

Some of the investigations referred to above have used pixel labeling examples to illustrate their algorithms [3,4,5]. However, there appear to be no detailed studies of the effectiveness of relaxation in this particular application, although Lev, Zucker and Rosenfeld [13] and Eklundh, Yamamoto and Rosenfeld [11] have given it preliminary consideration.

Though not specifically concerned with pixel labeling a number of authors have considered the problems of line and edge detection in imagery by pixel-specific means based upon relaxation procedures [3,5,12,13,14,15,16].

1.3 Discussion

In parallel with the literature survey a software implementation of the algorithm of Zucker and Mohammed [2] has been constructed and is being exercised. The purpose of this effort is to further study the possibilities of using a relaxation approach. A more complete report of this effort is now in preparation. In summary it can be reported that the technique does indeed appear to have some promising aspects for the problem at hand. However, some significant modification and adaption will be needed in order for the approach to be useful in the pixel labeling environment. An example of this is the following aspect of current algorithms which was discovered during the software-implementation study.

An essential ingredient of each of the schemes in the references is that the initial scene labeling is used only once, viz. when the algorithm is initialized, and thereafter the success of the final labeling is dependent upon both the attributes of the algorithm and the accuracy of the contextual data; both of these tend to assume increased significance relative to the initial labeling as relaxation proceeds. This may be appropriate in picture labeling problems such as the "toy triangle" example often used [1,2] since

the initial labeling is seen mainly as an initialization procedure and the contextual information is often known without error. The situation is usually quite different, however, in pixel labeling cases such as those undertaken in the interpretation of Landsat images.

For example, when it is desired to determine a label for every pixel in an image the contextual information would generally not be known exactly and indeed may only be an estimate based upon typical image data of a similar type. Further, the initial labeling, by and large, would represent "the best one can do" based upon all spectral information at hand, apart from context. In such a situation the information is therefore contained very much in both the context and the initial labels. As relaxation is applied, it is desirable that both of these sources be used to produce final labels which are, as far as possible, consistent with both the context and the initial labels.

Thus while the pixel labeling problem has by implication the initial pixel labels as the primary information and context (and later other variables) as ancillary or supporting information, the existing algorithms seem to imply more reliance upon context than upon the initial labels.

As a result of this, one characteristic observed consistently with the software implementation of the current algorithm is tnat as iteration proceeds, the results typically improve for a while, then peak and begin to decline. Apparently the contextual information, used in conjunction with the initial labels, does indeed improve the accuracy at least until the point where the influence of the initial labels has faded too far. Thus minimally one would need to determine a suitable stopping rule; a perhaps more suitable approach would be to modify the relaxation procedure more fundamentally so that it no longer has this peaking characteristic. Possible approaches to accomplish this which at the same time facilitate the incorporation of other types of ancillary data are being sought at present.

2. References

1. Rosenfeld, A., R. Hummel & S. Zucker, "Scene Labeling by Relaxation Algorithms," IEEE Trans. Syst. Man, Cyber, vol. SMC-6, pp. 420-433, June 1976.

2. Zucker, S. & J. Mohammed, "Analysis of Probabilistic Relaxation Labeling Processes," Proc. 1978 IEEE Conf. Pattern Recognition and Image Processing, Chicago, IL, May 1978, pp. 307-312.

3. Peleg, S., "A New Probabilistic Relaxation Scheme," Proc. 1979 IEEE Conf. Pattern Recognition and Image Processing, Chicago, IL, Aug. 1979, pp. 337-343.

4. Yamamoto, H., "A Method of Deriving Compatibility Coefficients for Relaxation Operators," Comp. Graph. Image Processing, Vol. 10, pp. 256-271, 1979.

5. Faugeras, O., & M. Berthod, "Scene Labeling: An Optimization Approach," Proc. 1979 IEEE Conf. Pattern Recognition and Image Processing, Chicago, IL, Aug. 1979, pp. 318-326.

6. Zucker, S., and J. Mohammed, "Analysis of Probabilistic Relaxation Labeling Processes," Technical Report 78-3R, Dept. of Electrical Engineering, McGill University, Montreal, Jan. 1978.

7. Zucker, S., Y. Leclerc & J. Mohammed, "Continuous Relaxation and Local Maximum Selection," Technical Report 78-15R, Dept. of Electrical Engineering, McGill University, Montreal, Dec. 1978.

8. Ullman, S., "Relaxation and Constrained Optimization by Local Processes," Comp. Graph. Image Processing, 10, (1979), pp. 115-125.

9. Haralick, R., J. Mohammed & S. Zucker, "Compatabilities and the Fixed Points of Arithmetic Relaxation Processes," Technical Report **-16R, Dept. of Electrical Engineering, McGill University, Montreal, 1979.

10. Zucker, S., A. Krishnamurthy & R. Haar, "Relaxation Processes for Scene Labeling: Convergence, Speed and Stability," IEEE Trans. Syst. Man, Cyber, Vol. SMC-8, Jan. 1978, pp. 41-48.

11. Eklundh, J., H. Yamamoto & A. Rosenfeld, "Relaxation Methods in Multi-spectral Pixel Classifications," Technical Report 662, Computer Science Center, University of Maryland, College Park, July 1978.

12. Peleg, S., & A. Rosenfeld, "Determining Compatibility Coefficients for Curve Enhancement Relaxation Processes," IEEE Trans. Syst. Man, Cyber, SMC-8, (July 1978), 548-555.

13. Lev, A., S. Zucker & A. Rosenfeld, "Iterative Enhancement of Noisy Images," IEEE Trans. Syst. Man, Cyber., SMC-7, (June 1977), 435-442.

14. Schachter, B. J., A. Levi, S. Zucker, & A. Rosenfeld, "An Application of Relaxation Methods to Edge Reinforcement," _IEEE Transactions on Systems, Man and Cybernetics_, Vol. SMC-7 No. 11, November 1977.

15. Zucker, S., R. Hummel & A. Rosenfeld, "An Application of Relaxation Labeling to Line and Curve Enhancement," _IEEE Transactions on Computers_, Vol. C-26, No. 4, April 1977.

16. Zucker, S., R. Hummel, & A. Rosenfeld, "Correction to 'An Application of Relaxation Labeling to Line and Curve Enhancement'," _IEEE Transactions on Computers_, Vol. C-26, No. 9, Sept. 1977.

2D. MULTISENSOR, MULTIDATE SPATIAL FEATURE MATCHING, CORRELATION REGISTRATION, RESAMPLING AND INFORMATION EXTRACTION

Paul E. Anuta*

1. Introduction

The work and results described in this section constitute the second and final year of a research investigation into the problems of combining and utilizing multiple data types for remote sensing surveys. The use of more than one type of data in a computer-based application of remote sensing has increased steadily over the past several years. Techniques for merging and utilizing various types of data are not well developed and this task seeks improved techniques for getting maximum benefit from available data.

The study considered the merging of different remote sensing data types, information extraction from the combined data and digitization and merging of ancillary data. The multidata-merging problem was explored, and results reported in the final report of the first year [1]. Information extraction of merged Landsat and SAR data is discussed in this, the second-year, final report as well as ancillary data digitization. A new concept for a multidata merging system emerged from the study and is also described below.

2. Landsat SAR Data Set Description

Registration of the Landsat MSS and SAR data types was studied in detail in the first year of the project and results were reported in the final report issued in November 1978. The merged SAR/MSS data set formed the basis of research done in the current year and we will briefly describe the data here.

The Landsat data are from frame 5-792-16152 imaged on June 19, 1977 over Phoenix, Arizona. Considerable trouble was encountered in obtaining these data as the digital tape was initially indicated to be unavailable even though the imagery was satisfactory. Further investigations revealed

ORIGINAL PAGE IS
OF POOR QUALITY
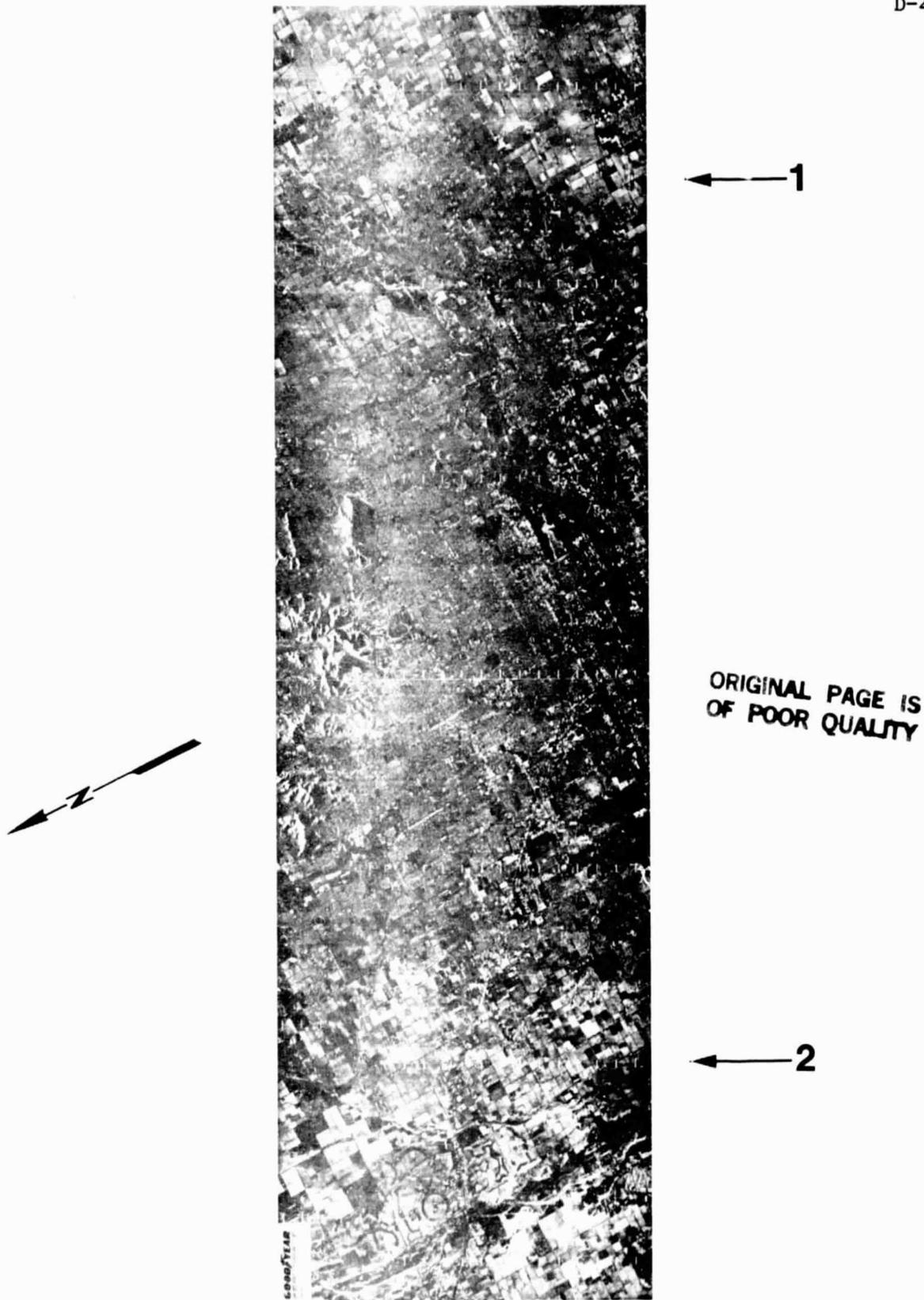
Figure D-1.   Goodyear SAR Data Set over Phoenix, Arizona used in the
study.   Flown on June 17, 1977 using an AN/APD-10 X band
radar in an Air Force RF-4 aircraft.   Area covered in
approximately 12 by 38 miles at a resolution of approx-
imately 10 feet.

that band 4 was unusable and a request was made to obtain the remaining bands since these were the only data which would match the SAR data in time.

The SAR data were flown by Goodyear Aerospace on June 17, 1977, using an AN/APD-10X band system. The full data set is shown in Figure D-1. The data were obtained in film format and were scanned and digitized to produce a digital image data file at a resolution of approximately 14 meters. This represents a reduction of the original system resolution of 3.3 meters due to film and scanning degradations. The characteristics of this and all the other data sets associated with the Phoenix site are listed on Table D-1.

The primary parameter to be selected was the resolution for the merged data sets. The deciding factor was a strong interest in the SEASAT SAR data which were to have a 25 meter resolution. Thus, this figure was chosen as a desirable compromise between the 79 meter Landsat and 14 meter SAR data. This choice would enable evaluation of the SEASAT SAR resolution in the crop field recognition environment.

A 512 by 512-point grid was defined over the crop area between Sun City and Phoenix, Arizona, centered approximately at the point that Grand Avenue enters Sun City from the east. At the 25 meter resolution, the area covered is 12.8 km square or 163.8 square kilometers (7.95 miles square or 63.3 square miles). The Landsat and SAR data were registered to this grid, using the LARS registration system and results of the previous year's registration study [1]. The 79 meter resolution Landsat data were interpolated to 25 meters resolution, using cubic interpolation and the 14 meter SAR data were undersampled using the nearest-neighbor rule to achieve 25 meter pixel spacing. A dot matrix printer image of Landsat band 5 for the block is shown in Figure D-2 and the SAR image is shown in Figure D-3.

The June 1977 SAR and Landsat data formed the data base for the crop classification study. Although acquisition and registration of other data types for this and other areas were planned, these data could not be obtained and preprocessed in time for analysis in the study. SEASAT SAR, Coastal Zone Color Scanner, Return Beam Vidicon were among those considered. Digital SEASAT SAR and RBV data were obtained during the study and discussion of work done on these data is included in another section.

Table D-1. Data Sets Generated for Phoenix, AZ Site.

Run No.	Lines	Cols.	Pixel Size	Channels	Tapes	File	Description
77090000	4400	1364	14 x 14M	1	4615	1	Digitized SAR film of June 17, 1977 flight.
72069109	512	512	25 x 25M	5	4621	1	Landsat data from Oct. 1972 as Channels 1, 2, 3, 4 registered with SAR data from 7709000 as Channel 5.
72069110	512	512	25 x 25M	7	161	1	Channels 1-5 same as 72069109. Channel 6 is gradient magnitude of Channel 2 (Band 5) and Channel 7 is gradient magnitude of SAR.
72069111	512	512	25 x 25M	11	4543	1	Landsat data from June 19, 1977 registered to channels in 72069110 and included as Channels 2, 9, 10, 11. (Band 4 Landsat data are bad, thus Channel 8 is not useable.)

121
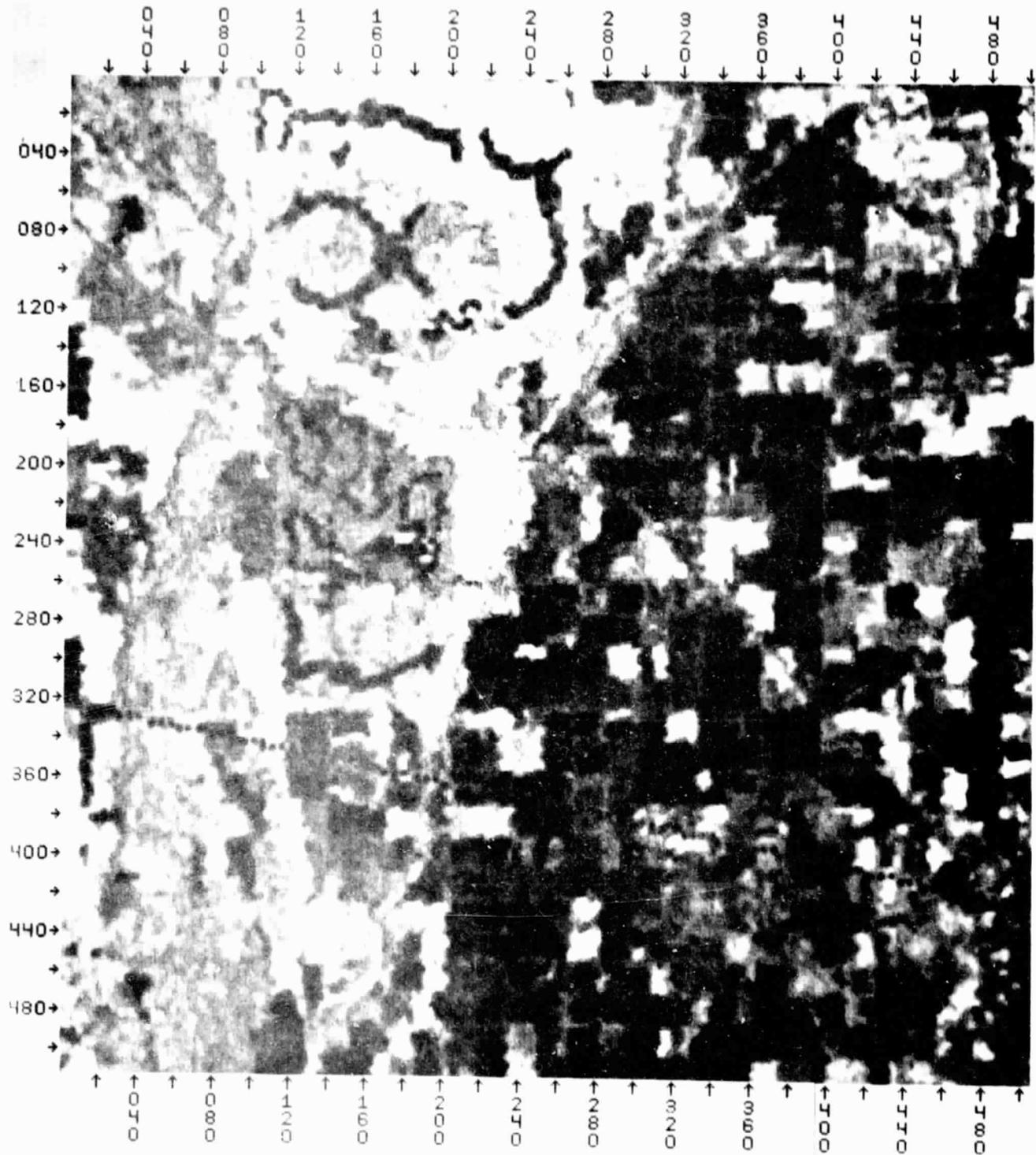
Figure D-2. Landsat Image, Channel 2 (0.6-0.7 μm), Cubic Resampling
to a 25 x 25 Meter Resolution (Phoenix, AZ)
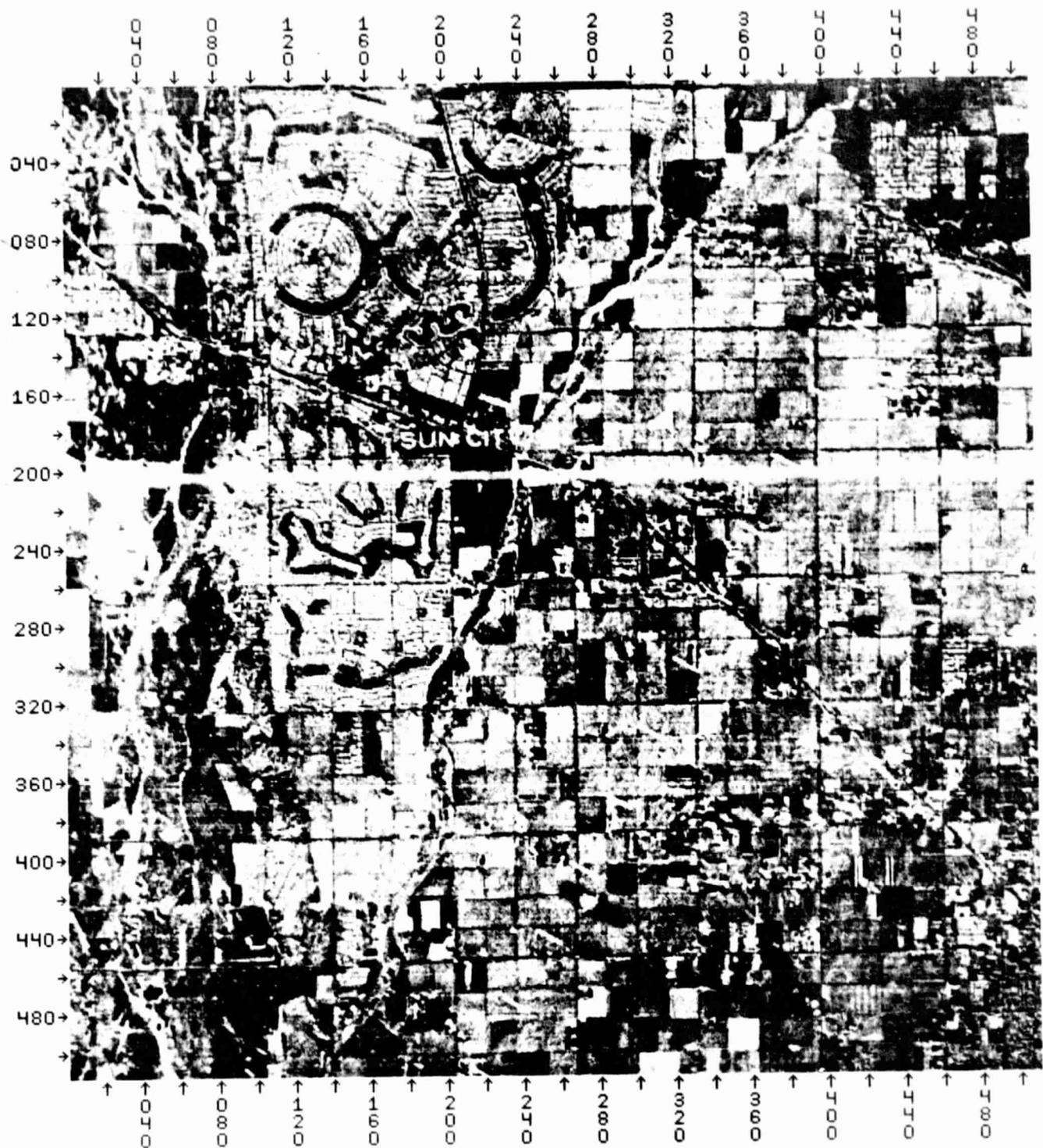
Figure D-3. Aircraft SAR (3 cm), N.N Resampled and Registered
to 25 x 25 Meter Landsat (Phoenix, AZ).

3. Crop Classification Using Landsat MSS and SAR Data

3.1 Agricultural Characteristics of Data Set

In the previous section, the characteristics of the merged data set were discussed. The agricultural "scene" consists primarily of cotton fields with urban encroachment by Phoenix on the east and Sun City on the west. A ground truth mission was conducted in March of 1978 in the area with retroactive truth obtained for the data time of June 1977. Ground truth for a total of over 600 fields was obtained for agricultural areas both east (Area 1) and west (Area 2) of Phoenix. The 512 by 512-point data set created covered only the west Phoenix area.

In order to simulate a segment size area, a 3 by 5-mile block was selected from the agricultural area indicated in Figure D-1. The crop area between Sun City and Phoenix is limited in size and many housing tracts exist throughout the area, thus a full 5 by 6-mile segment could not be chosen. The 3 by 5-mile block is on the order of a segment and was assumed to provide a reasonable simulation.

Within the 3 by 5-mile segment containing 15 sections, there were 76 ground truthed fields. The contents of these fields and the number of pixels in each are indicated in Table D-2. Note that the majority of fields in the area are cotton, thus a good distribution of crop types did not exist. There are 17,054 pixels for which ground truth was collected and there are 62,699 pixels in the 3 by 5-mile block, thus only a portion of the segment could be analyzed. Thus, due to data set limitations, the segment area was only 7.3% of a LACIE type segment. Nonetheless some interesting results were obtained.

3.2 Classifier Training

The cluster block approach was taken in training the classifier. Blocks of pixels containing samples from each class were clustered, using the LARSYS * CLUSTER routine. A total of five blocks were used containing 4,772 pixels with cluster and information classes as listed in Table D-3. The statistics from the five cluster jobs were merged using the * MERGE processor in LARSYS.

Table D-2. Classes Analyzed in SAR/Landsat Data

Class	No. Fields	No. of Pixels
Cotton	40	9377
Alfalfa	12	3345
Barley	2	364
Urban	22	3968
TOTAL	76	17,054

Table D-3. Cluster Block Contents.

Block	No. Pixels	No. Clusters	Classes in Cluster
1	1160	10	Alfalfa, cotton, fill
2	1300	10	Cotton, oranges, wheat
3	812	6	Cotton, alfalfa, barley
4	756	6	Alfalfa, cotton
5	744	10	Cotton, urban

The results were statistics decks for the information classses: cotton, alfalfa, barley and urban. Some occurrence of orange groves, wheat and bare soil existed but the number of samples was too small to warrant keeping them in the analysis. Statistics were generated for the three bands of Landsat and for the three Landsat bands plus the SAR band. Training field classification accuracy figures are not available since the statistics are derived from clusters covering several fields and statistics subsequently merged to a final set of class statistics.

3.3 Classification Analysis

The Landsat/SAR data set was classified using both pixel and field classifiers and using Landsat only and Landsat plus SAR bands. The results of these tests are presented in Table D-4 with some additional results to be discussed later. The best overall results were obtained using the field classifier and spectral data only. Addition of the SAR channel reduced test classification accuracy in most cases, except alfalfa and barley. The large amount of cotton in the test caused the poor performance when SAR was added to pull the overall result way down. In general, the SAR seems to reduce separability of the spectral classes and it would appear that direct addition of this particular SAR data to the spectral data is undesirable.

Multifrequency radar data with multiple polarization, collected over a sequence of times, has been shown to provide accurate crop classification [2]. The single time, single polarity, X band case case apparently contains insufficient information in this case. The experiment should be tried on corn, soybeans, wheat and other grains of interest to AgRISTARS programs since the single band, single polarization case is all that is likely to be routinely available from satellite platforms in the next decade.

3.4 Data Base Approach to Classification

The availability of high resolution imagery of the earth scene from the satellite platform provides the opportunity to employ scene structure as an input to the classification process. High resolution refers to the

Table D-4. Classification Results for Phoenix Site Test Fields, % Correct.

Class	Spectral Pixel Classifier	Spectral Field Classifier	Spectral + SAR Pixel	Spectral + SAR Field	Spectral Majority Classifier	Spectral Plurality Classifier
Cotton	70.2	87.6	48.1	43.3	95.2	100.0
Alfalfa	59.8	35.5	67.8	80.7	60.1	93.3
Barley	63.5	42.3	20.3	55.2	100.0	100.0
Urban	46.9	55.0	28.5	49.3	25.9	84.7
Overall	62.6	68.8	46.8	52.3	72.3	95.1

RBV or SEASAT SAR order of resolution of 20 to 25 meters in contrast to MSS resolution of 79 meters and thermal IR resolution of 150 meters or more. The basic concept is to use a single channel high resolution image as a mapping band to define scene structure and to then use spectral samples from within the objects in the scene for classification of those objects. The approach is clarified in Table D-5.

Using this approach, the results in Table D-4 can be reinterpreted in terms of knowing all field boundaries in the scene. These would be obtained from high resolution SAR, RBV, MLA or any source of current imagery of the scene to be analyzed. Boundary extraction is a problem in scene segmentation and not treated here. Given to a boundary definition for all scene objects, the classifier can be trained from known scene objects and the classifier applied to the data set. This sequence is diagrammed in Figure D-4.

In the experiments carried out here, both field and pixel classification was carried out for the SAR Landsat data set. The field classifier results using spectral data were seen to be better than the pixel results but neither was very good. Knowing the field boundaries allows the results of pixel classification to be analyzed according to majority or plurality rules. In this approach, the class having a majority or a plurality is assumed to be the correct class for all points in the field. This is considered to be a valid approach since we are assuming we have boundaries enclosing every field and the contents of the field are homogeneous.

The chart in Figure D-5 compares overall test results for all cases discussed earlier, plus majority and plurality results which are tabulations obtained from the pixel classification results from each field. These results are significantly better than for the individual pixel or field classifier results. Thus the plurality rule applied to pixel classifier results for the case of known homogeneous fields appears to be an attractive approach.

Table D-5. Rationale for Object Classification

• For cases in which the same highly structured
  scene sample area is to be classified each
  year for many years, a data base of object
  boundaries can be maintained.

• Image segmentation technology can be used to
  establish object boundaries initially and
  update boundaries each year.

• Spectral classification of objects rather
 than pixels or blobs can then be carried out.

```
┌─────────────────────────────────────┐
│  Delineate boundaries of all fields  │
│  and other objects in scene using a  │
│  high resolution  image and store in │
│  data base registered to spectral data│
└─────────────────────────────────────┘
                    │
                    ▼
┌─────────────────────────────────────┐
│  Train pixel classifier using ground │
│  truth and standard procedures.      │
│  Classify all pixels in spectral data.│
└─────────────────────────────────────┘
                    │
                    ▼
┌─────────────────────────────────────┐
│  Apply majority or plurality rule    │
│  to each object and classify entire  │
│  object as one decision              │
└─────────────────────────────────────┘
```
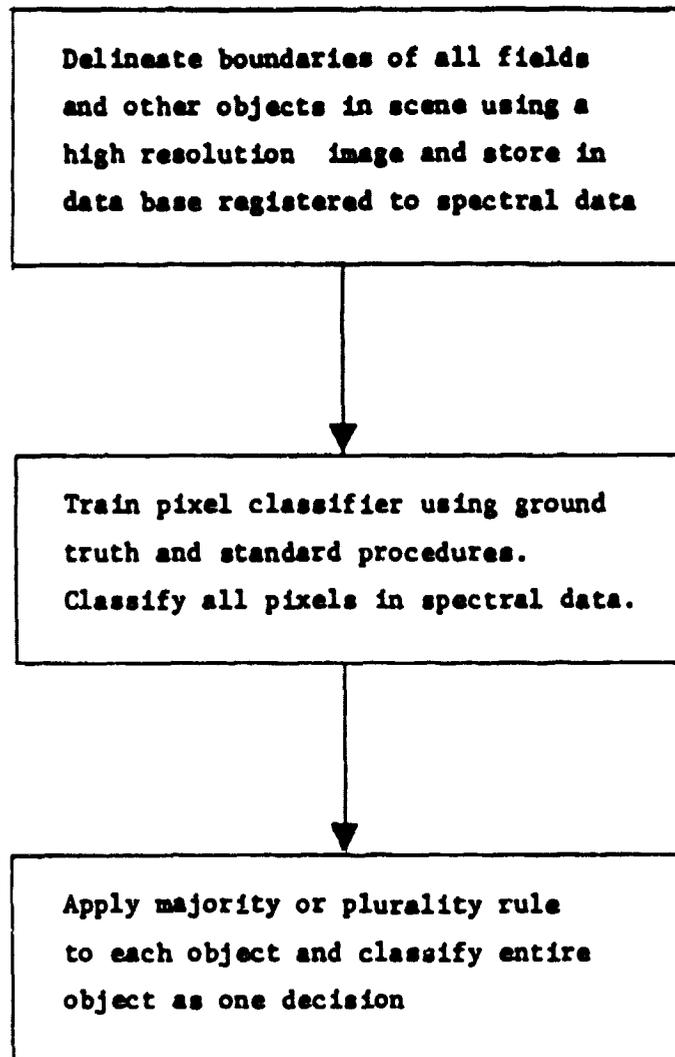
Figure D-4. Object Classification Using Boundary Data Base
             and Majority or Plurality Decision Rule.

Overall Clasification Results
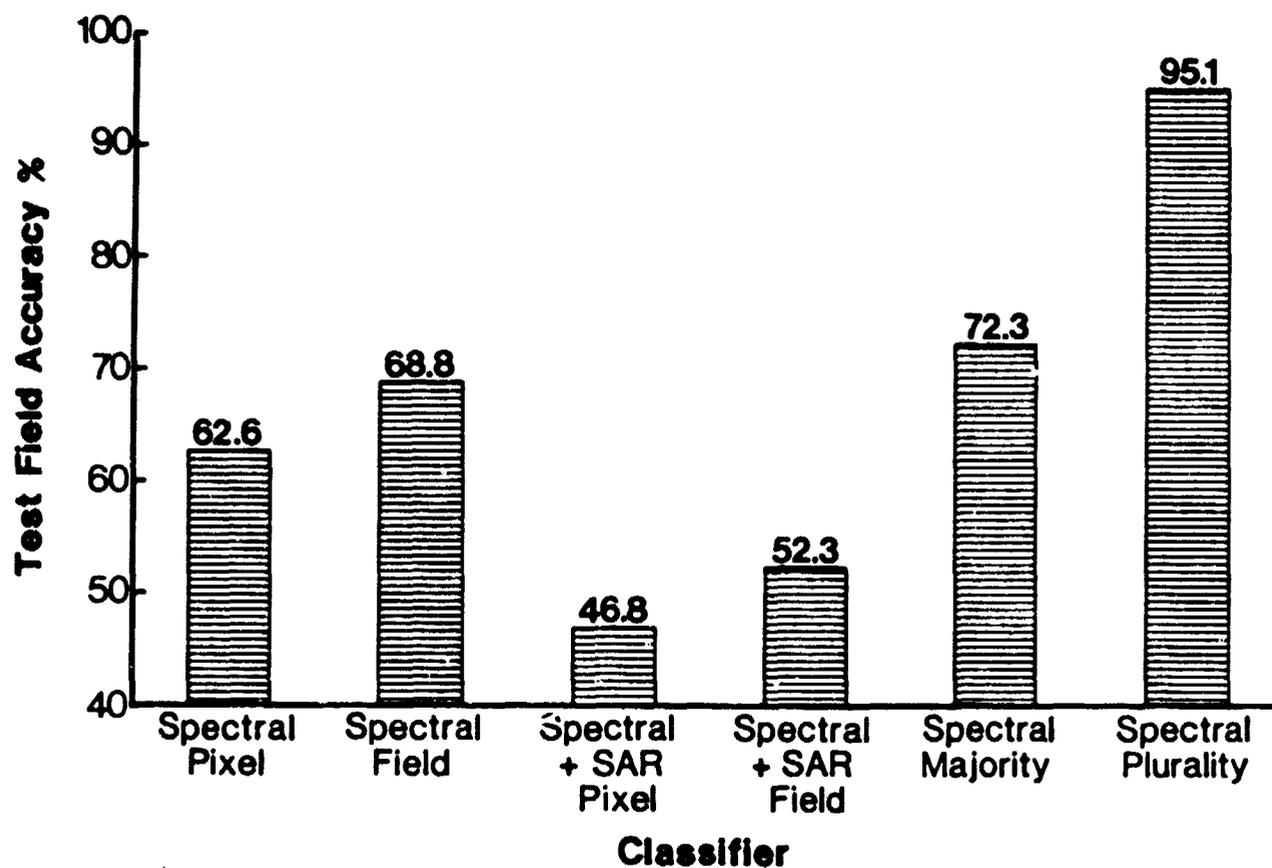
Figure D-5. Comparison of Overall Text Field Classification Accuracies.

4. Ancillary Data Digitization

The problem of conversion of ancillary data sources to a digitized gridded format was studied as another aspect of the multidata task. The standard approach to map digitization is to manually follow all linear features with a digitizing cursor and to store the sequence of digitized coordinates for later processing. A laborious procedure is required to assign labels to each line from the map and a gridding algorithm is then used to create an image-like data set from the digitize line data. This project further evaluated an alternate method for digitizing maps which would be more efficient.

In the SR&T research year ending May 1977, a color map digitizing method was developed and reported on [3]. The approach used spectral classification to extract polygons from color coded maps [4]. Testing was carried out only on a pastel colored, noisy map and results were very good. In the current task, the use of high saturation pure colors was evaluated and results were extremely accurate, as expected.

Two forest resource unit maps were hand-colored into 17 units and the result photographed and digitized on a color separating microdensito-meter. Three channel digital data sets representing the blue, green and red primary separations were generated. A training sample was chosen from each color and used to train the LARSYS pixel classifier. Figure D-6 con-tains one of the map units. The maps were then classified and an evalua-tion was made of errors.

Table D-6 contains a list of errors in each color for interiors of polygons. Significant errors also existed at edges of polygons due to painting irregularities and mixed pixels. The edge errors could not be readily visually evaluated and the interior errors were assumed to repre-sent the performance of the method. As can be seen, the error rate is very low, on order of .2%, and it can be concluded that this method is an attractive approach to map digitization.

Figure D-6. Forest Operating Area Map Segment, Hand-Colored for Scanning
and Digitizing. There Are 19 Different Areas Color Coded
With Acrylic Polymer Paint.

Table D-6. Errors in Color Map Classification.
Number of Errors.

Color	AU268	Color	AU271
White	53	White	96
Bright Red	3	Red	0
Red	48	Lt. Green	0
Dark Red	17	Med. Green	0
Orange	9	Dk. Green	0
Lt. Green	4	Yellow	0
Green	2	Gold	0
Dk. Green	318	Orange	355
Pink	0	Dk. Orange	2
Lavender	1	Lt. Blue	9
Lt. Yellow	0	Med. Blue	1
Yellow	7	Dk. Blue	1
Lt. Brown	0	Brown	0
Brown	0	Tan	0
Lt. Blue	1	Lt. Brown	0
Blue	11	Gray	0
Dk. Blue	4	Purple	2
TOTAL	478	TOTAL	457

| TOTAL AREA (Pixels) | 279,006 | | 243,714 |

5. Multidata Merging Procedures

The basic problem motivating this task was the combination of
dissimilar data types to enable coordinated digital analysis for various
applications. The concept of a self-defining data set (SDDS) set forth
here as an approach to solving the problem of merging arbitrary areas
from diverse data types. An SDDS would have the following characteristics:

. Complete geometric description of data set included in header.

. For multitype, multitemporal data sets, each channel will be
fully described.

. Format flexible to accommodate different word sizes and resolutions.

The concept was inspired by the CCT-AM tape format set forth for the
Landsat digital image tapes which will be produced by EDC. Full geometric
and ¯adiometric information is to be provided in the headers of these
tapes. The proposal here is to provide this information for any data type
which may be used for remote sensing data analysis.

To test the SDDS idea, a basic software system was developed in this
task which carries out the basic functions needed. The problem has two parts:
(1) generating an SDDS, and (2) combining SDDS's to form a merged data set
in a user-defined coordinate reference. Parameters judged necessary to
define an SDDS are listed in Table D-7.

The basic function needed to specify image geometry is control point
determination. A software element was created which utilizes a stored
file of ground control points and attempts to locate these points in
uncorrected imagery. The diagram in Figure D-7 shows the elements of the
experimental system. The program LOCGCP accepts control point coordinates
in latitude and longitude and estimates the location of the point in the
image data using ephemeris and one initial control point. Small images
surrounding the estimated point can then be printed out on a line or dot
printer. Experiments were carried out on the use of the standard
alphanumeric CRT computer terminal to provide a rapid, low cost, low resolu-

Table D-7. Parameters for an SDDS.

Data Set Parameters

· Center latitude and longitude and
corresponding line and column.

· Azimuth of center column at center line.

· Polynomial function for relating lines
and columns to georeference coordinates
(lat, long or UTM northing, easting).

Channel Parameters

· Time of collection
· Type of data
· Bits per word
· Cell size
· Band size
· Band center
· Full-scale calibration
· Mean
· Variance

Uncorrected
Data (1)

Self-Defining
Set Form of
LARSYS Format (2)

Image Data in User-
Defined Grid (3)
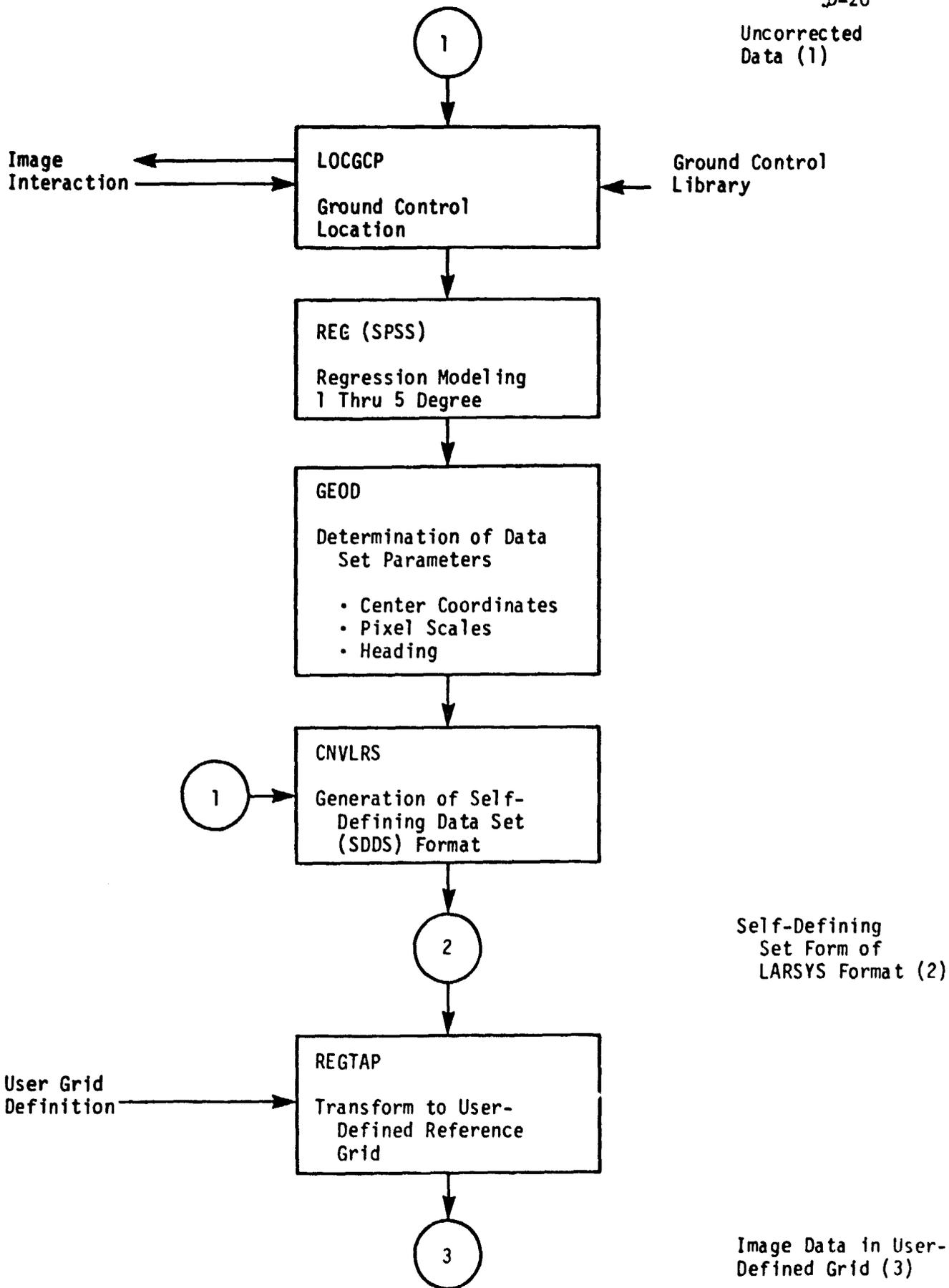
Figure D-7. Software Elements Developed to Study
Self-Defining Data Set Concept

tion image display. Several gray level sets were tested and none was found satsifactory. A two-level thresholded image was found to be quite useful in many image cases. Data were used for the low or dark level and M's for the bright level. The program LOCGCP displays such an image with the threshold rapidly adjustable by the operator. Figures D-8 and D-9 contain two examples of successful displays in eastern Florida. Figure D-8 is a bridge over an estuary on the east coast and the control point coordinates are marked by an asterisk cursor which in this case has been moved under keyboard control to the center of the bridge. In the second example, the control point is the intersection of two interstate highways and the cursor is located in the middle of the interchange. Many subimages do not produce useful binary images and must be imaged on the line printer dot matrix printer or, if possible, on a high-resolution image display. It is interesting to note that subimages which are unuseable on the binary CRT are also very difficult to interpret on any other output media.

The second step in the SDDS generation process is the regression modeling of the image distortions. This step is handled by standard regression programs and coefficients relating geographic to image coordinates are generated. This area is a subject of continuing study and the form and number of coefficients needed for any one image cannot be stated at this time. The functional relationship goes to the GEOD program and generates additional parameters needed for geometric description. The CNVLRS program reads a data set without full geometric description and generates the SDDS.

The user then can access the SDDS tapes and generate his own merged data set by processing any number of input SDDS files. The REGTAP program reads user grid-definition parameters, selects areas to be used from input SDDS's and writes out a new SDDS with the geometric characteristics and data types he specified.

In order to test the concept on an existing data system, the LARSYS tape format was taken as the logical starting point. Table D-7 contains

suggested changes to the indentifications record for the image data tape
which pursue the SDDS concept. Ideally a zero-based format plan would
be established; however, the current LARSYS format although designed
for single data aircraft multispectral scanner data, has proven itself
quite durable in adapting to multitype, multidate data sets. The ele-
ments added in Table D-7 are adequate for testing purposes while new
unconstrained general formats are discussed.

The data-merging system outlined here was tested on one data set for
the study. The Landsat frame from the Picayune, Miss., area was selected
since it was the first frame for which both fully geometrically corrected
and uncorrected data were available. The control point finding and regres-
sion modeling portions were tested on the uncorrected frame. Geometric
description is also needed for the fully corrected data even though no
distortions exist. An affine model is used to relate line-column to UTM
or other projection coordinates for the fully corrected data. Both data
forms, after conversion to SDDS format, were processed through REGTAP to
produce a north-oriented data set for one 1:24000 USGS quadrangle. The
package is experimental; however, it is available for use by qualified users.

6. Multitype Data Set Acquisition

The basic aim of this task was to develop merging and analysis techni-
ques for multiple data types. Landsat, SAR, and ancillary map data types
were included in the study. Many other data types were of interest but
unfortunately none could be obtained and reformatted in time to include
in the study. Examples of two additional data types were acquired near
the end of the task and reformatting software was developed for these data .

The two data examples were SEASAT SAR data and Landsat RBV data. Film
format RBV data had been acquired earlier in the year for the Phoenix site
but the quality of the data was judged too poor to warrant digitizing and
analysis. Figure D-10 contains a photo of the test site made from the RBV
frame. The digital RBV data are for the Oroville, Calif., EDC example
and was useful only for developing the reformatting program. The SEASAT
SAR digital data was in LARSYS format and only ID editing and login opera-
tions were needed. No further work was performed on the RBV and SEASAT
SAR data in the contract period.
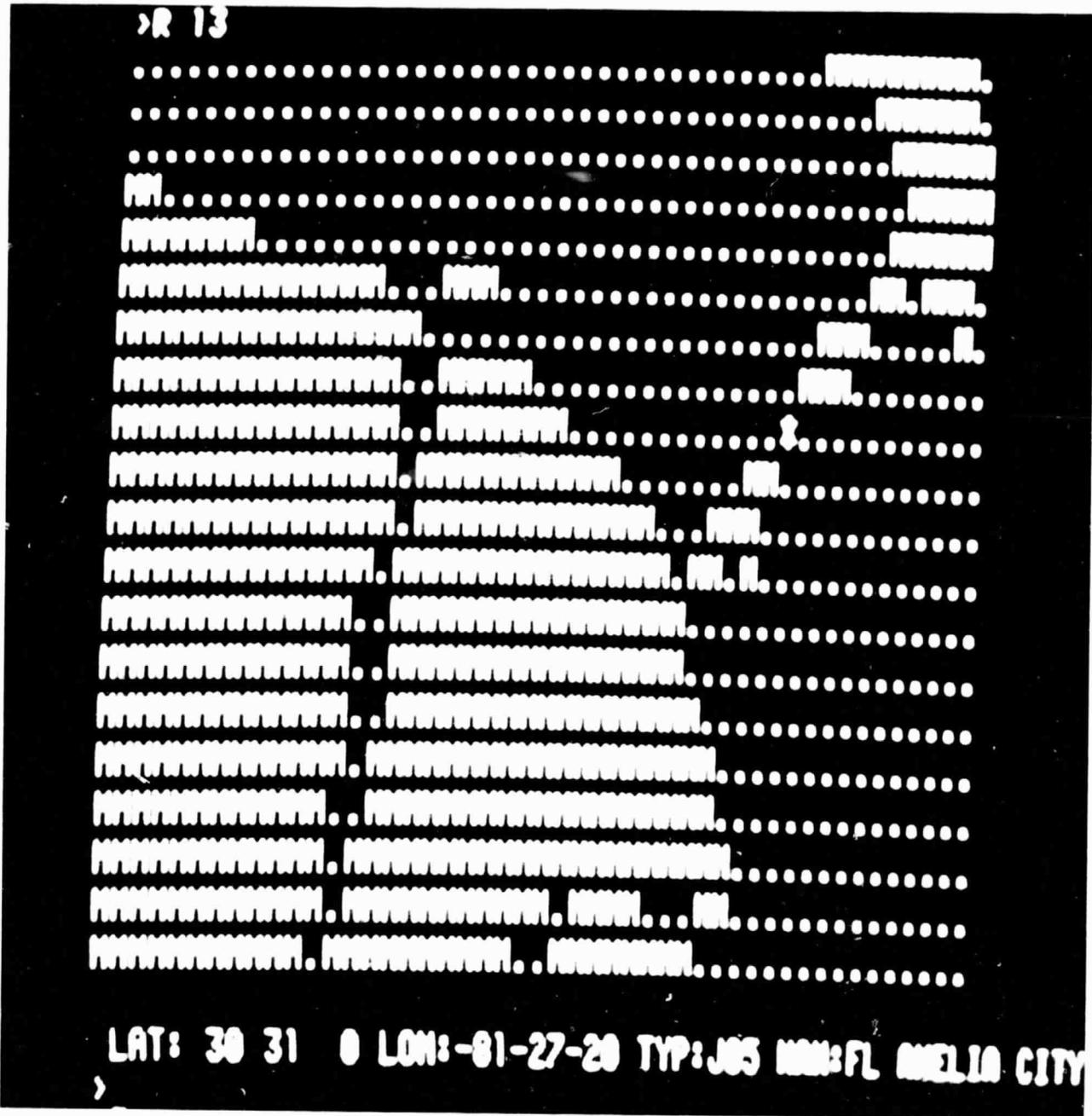
Figure D-8. Landsat Band 5 Image produced on CRT Terminal Screen
using two levels (Amelia City, FL area).
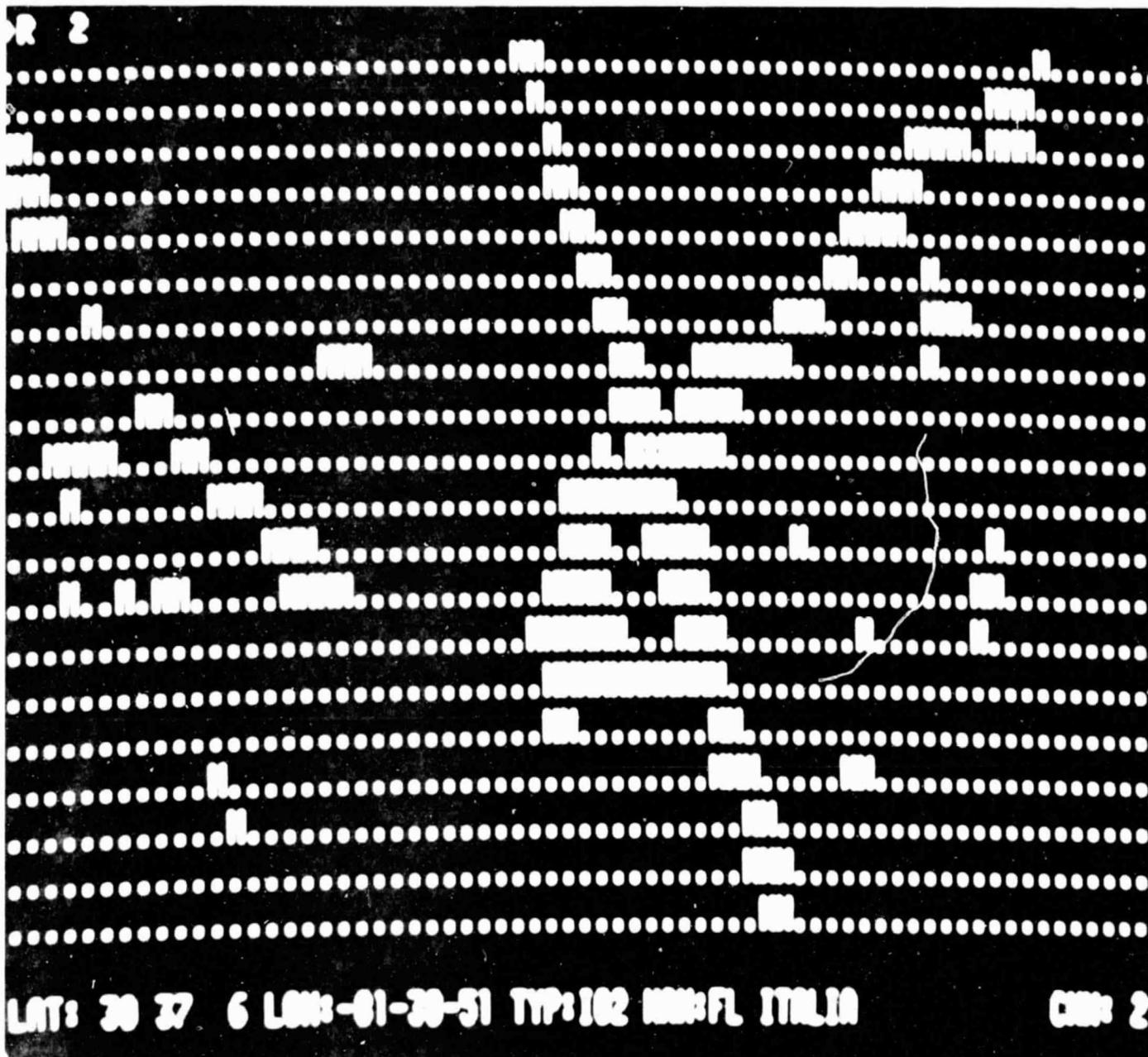
Figure D-9. Landsat Band 5 Image produced on CRT Terminal Using Two Levels.

Table D-8. Proposed Changes to LARSYS Format

Additions to ID Record

Words

21	Latitude of center
22	Longitude of center
23	Column of center
24	Line of center
25	Horizontal pixel scale
26	Vertical pixel scale
27	Projection code
31-40	Coef for line geometric function
41-50	Coefficients for column geometric function

Changes to Channel ID Words   K=0,29

5K + 51	Data type code
5K + 52	Collection date
5K + 53	Bits per word
5K + 54	Full-scale calibration
5K + 55	Band center wavelength
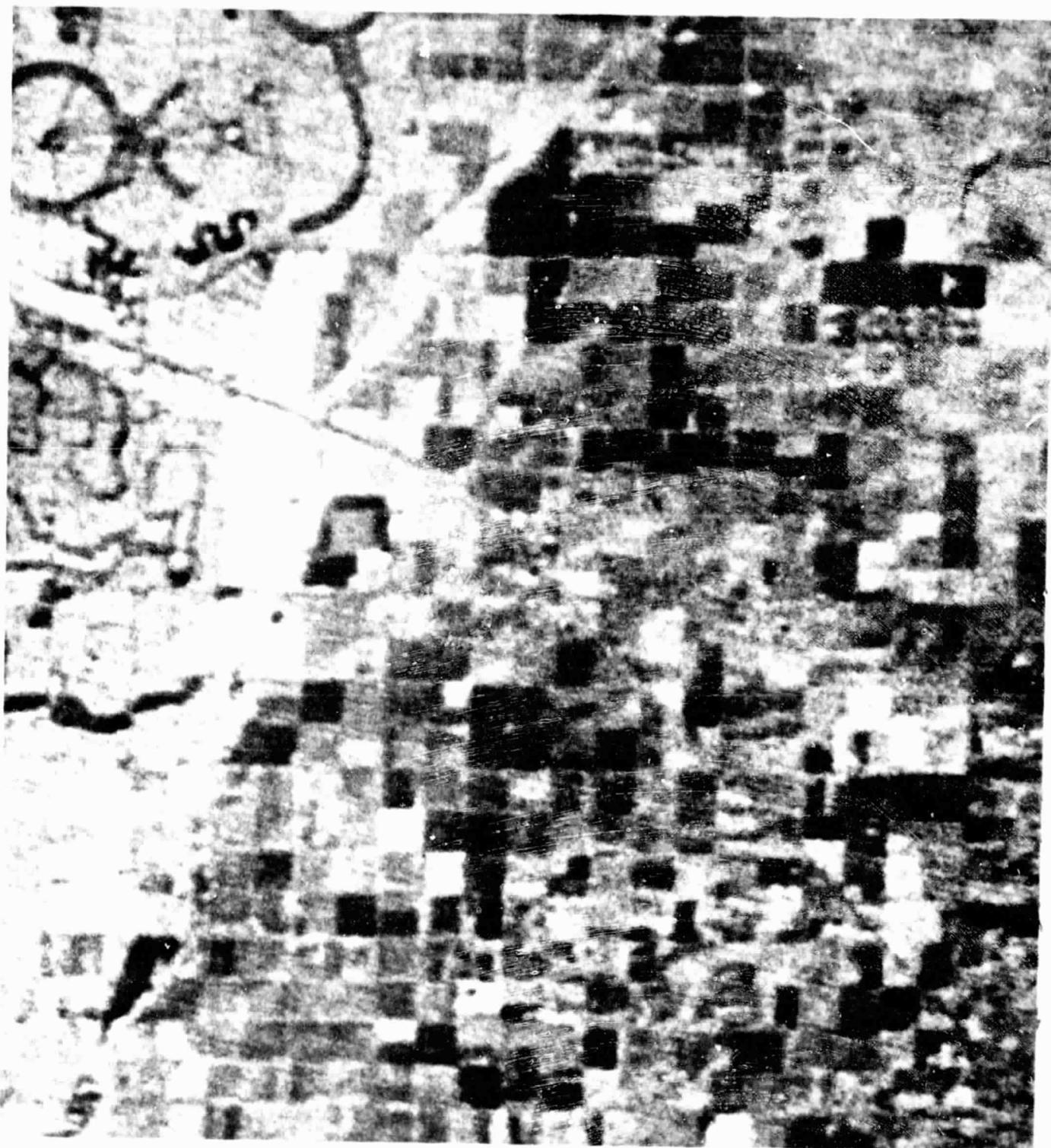
Figure D-10. Landsat RBV Data for Phoenix, AZ Area
From Frame 30104-17212A, June 17, 1978.
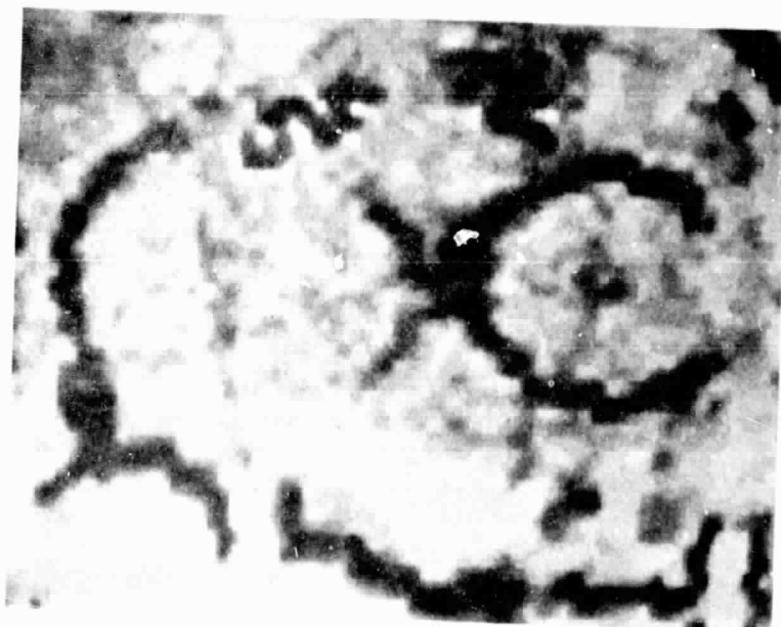
Figure D-11.   Landsat Image of Sun City, AZ Area (Original in Color).
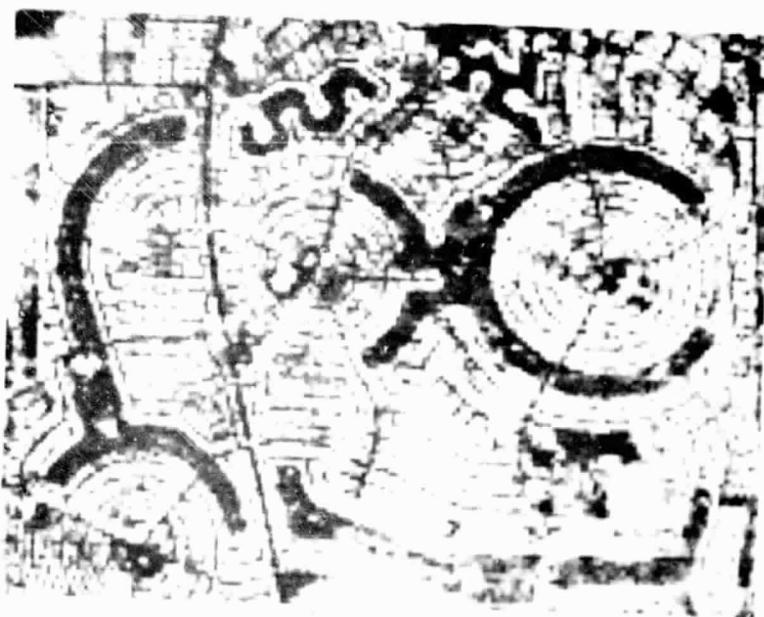
Figure D-12.   SAR Image of Same Area Covered in Figure D-11.
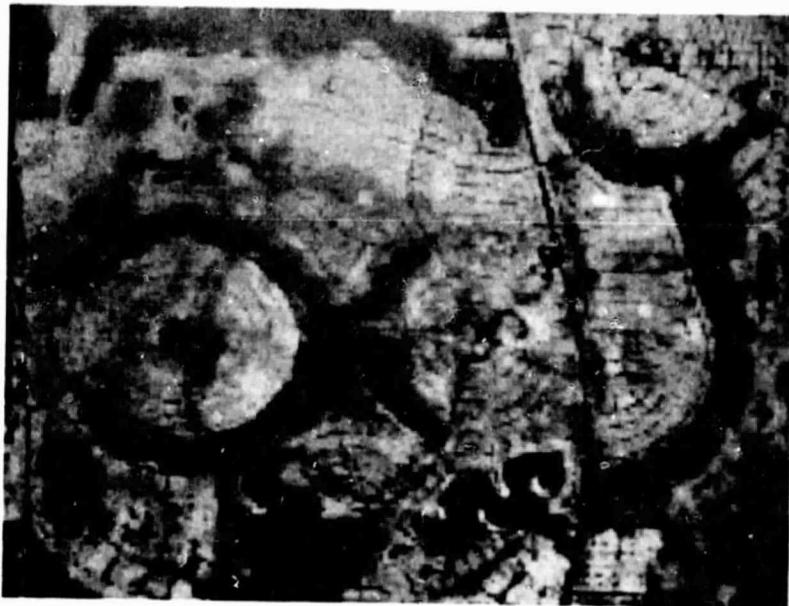
Figure D-13. Mixed Landsat and SAR Image (Original in Color)

7. Image Enhancement Experiments

The last part of the multidata task was the investigation of the image enhancement benefits of combining different data types in black and white and color renditions. The SAR Landsat data set for the Phoenix area was used for this activity. The LARS digital display system was employed to photographically mix channels of the data set through color filters onto film. The results of these experiments are shown in Figures D-11,12,13. Figure D-11 contains a combination of Landsat bands 4, 5, and 6 to produce a standard false color infrared reproduction reflecting the 79 meter MSS resolution. The scene is the center of Sun City, Ariz., where circular park areas are surrounded by housing units. Figure D-12 is the SAR data for the same block at the 25 meter resolution. Figure D-13 contains a mixture image in which the SAR band replaced the blue color in the blue, green, red sequence. The reproductions in the report are in black and white; however, the resolution properties can still be observed.

The mixture image contains the street patterns and fine structure of the area which is lost in the Landsat. The obvious benefit here is in delineating boundaries of scene objects for aiding in classifier training and results evaluation. Similar combinations were prepared for agricultural areas and again a general sharpening of field and subfield edges was observed. The benefit of such combinations is subjective and analyst evaluation is needed to fully evaluate best combinations for aiding training sample selection.

8. Summary

The multidata merging and information evaluation task investigated several aspects of utilization of different data types for remote sensing surveys. The primary topics studied were: (1) Merging of different types of remote sensing data, (2) digitization and merging of ancillary data, and (3) information extraction from the combined data sets. Due to difficulties in obtaining data, only Landsat and synthetic aperture radar data types were studied. Digitization and merging of color map ancillary data sources were studied and a color classification method was validated. A self-

defining digital data set structure was defined and implemented to facilitate merging of different data types.

The most significant result of the study is the self-defining data set approach and its implementation in an experimental software system. Full development of the software will greatly simplify the user creation of complex multiple type data sets for any application. The addition of side-looking radar data to Landsat MSS data did not provide improved classification performance for the predominately cotton agricultural test case. Multi-frequency radar data have shown more promise in other work. The chief benefit perceived in radar imagery is providing a current high resolution view of the scene to enable fine-detail structure to be mapped. With the map structure determined, spectral classification can proceed on the interiors of scene objects using lower spatial resolution multi-spectral data. This approach is suggested for further study in the repetitive multicrop monitoring applications. The current RBV data provide a potential high resolution mapping capability and future satellite side-looking radar systems could provide superior scene structure images.

9. References

1. Anuta, P. E., Hixson, M. M. and Swain, P. H., "Vol. III Processing Techniques Development," Final Report of NASA Contract NAS9-15466, Contract Report 112778, Laboratory for Applications of Remote Sensing (LARS), Purdue University, November 1978.

2. Bush, T. F. and Ulaby, F. T., "Cropland Inventories Using an Orbital Imaging Radar," Remote Sensing Laboratory Technical Report 330-4, University of Kansas Center for Research, Inc., January 1977.

3. Landgrebe, D. A., "Final Technical Report for NASA Contract NAS9-14970," Vol. II, Laboratory for Applications of Remote Sensing (LARS), Purdue University, May 31, 1977.

4. Chu, Nim-Yau and Anuta, P. E., "Automatic Color Map Digitization by Spectral Classification," Photogrammetric Engineering & Remote Sensing Journal, Vol. 45, No. 4, April 1979, pp. 507-515.