To the Graduate Council:

I am submitting herewith a thesis written by Dianne H. McCollough entitled "Interpretation of a Programmable Tse Computer Instruction Set." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Robert E. Bodenheimer

Robert E. Bodenheimer, Major Professor

We have read this thesis
and recommend its acceptance:

Michael G Thomason

C P Irving

Accepted for the Council:

L Evans Peck

Vice Chancellor
Graduate Studies and Research

Thesis
77
.M33
copy 1

INTERPRETATION OF A PROGRAMMABLE TSE COMPUTER INSTRUCTION SET

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Dianne H. McCollough

August 1977

1328525

ABSTRACT

The growing need for the processing of two-dimensional data arrays has necessitated the development of components which operate on each bit of a binary data array simultaneously. Presently, conventional serial machines take comparatively much more time to process an array bit by bit than would a machine which could process all bits of the array simultaneously. Thus, D. H. Schaefer and J. P. Strong have proposed a new class of computers (tse computers) in which a binary array or image is used as the basis of operations. Components which are being designed at the Goddard Space Flight Center (GSFC) are the building blocks of these computers. Rickard and Bodenheimer have designed a programmable tse computer with an instruction set, the interpreting of which is the subject of this research.

First, definitions of and comparisons among emulation, simulation, interpretation, and translation are presented. Interpretation is the method chosen for the study of the programmable tse computer instruction set.

The history of tse computers and their components is given and is followed by a detailed presentation of Golay transforms whose functions are facilitated by tse computers. Since the programmable tse computer instruction set has been especially tailored to perform Golay transforms, Golay's skeletonizing and swelling algorithms are presented.

Next, a general description of the programmable tse computer is given. This includes information on its flexibility, its micro-computer based control unit, its organization, and its logic. Each instruction is fully described, and the details of how the interpretation is carried

out on the PDP 11/40 computer are given. The interpretation gives the user the ability to program the tse instructions and study special functions without having the actual computer for which these instructions are written.

Some suggestions for future research are also included. One method of improving the interpreter is to enlarge the data arrays to accommodate more complex images to be processed by Golay transforms.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	INTRODUCTION	1
	Methods of Modeling	1
	Interpretation of Tse Instructions	3
2.	TSE COMPUTERS	4
	History of Tse Computers	4
	Components of Tse Computers	5
	Usefulness of Tse Computers	9
3.	GOLAY TRANSFORMS	11
	Square and Hexagonal Arrays	11
	Hexagonal to Rectangular Modification	13
	Golay Subfields	18
	Golay Indexes	18
	Golay Neighbor Planes	22
	Golay's Skeletonizing and Swelling Algorithms	25
4.	RELATIONSHIP BETWEEN TSE LOGIC DEVICES AND GOLAY TRANSFORMS	34
	Control, Masks, and Memory	34
	Index Recognition Circuit Using Tse Components	35
5.	PROGRAMMABLE TSE COMPUTER	37
	Organization of Programmable Tse Computer	37
	Control Unit	39
	Tse Computer Instruction Set	41
	Programs to Perform Golay's Skeletonizing and Swelling Algorithms	46

CHAPTER PAGE

6. INTERPRETATION OF TSE COMPUTER INSTRUCTION SET 49
Outline of the Interpreter Program 49
Interpreter Registers, Masks, and Storage 50
Input and Output 52
7. CONCLUSION 57
LIST OF REFERENCES 60
APPENDIXES 62
APPENDIX A. SCHEMATIC SYMBOLS FOR TSE LOGIC DEVICES 63
APPENDIX B. MAIN PROGRAM 66
APPENDIX C. SECONDARY PROGRAM 81
VITA 93

LIST OF TABLES

TABLE		PAGE
3.1	Indexes of Golay's Surrounds	23
5.1	Tse Instructions	42
5.2	Register and Mask Constant Definitions	45
5.3	Program to Perform Golay's Skeletonizing Algorithm	47
5.4	Program to Perform Golay's Swelling Algorithm	48
A.1	Schematic Symbols for Tse Logic Devices	64
B.1	Main Program	67
C.1	Secondary Program	82

LIST OF FIGURES

FIGURE		PAGE
2.1	Examples of basic tse operations	6
2.2	An interleaver, (Courtesy of Earth Observation Systems Division, Goddard Space Flight Center)	8
2.3	EXCLUSIVE-OR circuit	10
3.1	Square Array	12
3.2	Hexagonal Array	14
3.3	Triangular Array	15
3.4	Hexangular arrangement of hexagonal pattern	16
3.5	Rectangular arrangement of hexagonal pattern	17
3.6	Subdivision of hexagonal array into three subfields . . .	19
3.7	Subdivision of hexagonal array into four subfields	20
3.8	Subdivision of hexagonal array into seven subfields . . .	21
3.9	Neighbors of module $p_{i,j}$ in even and odd rows	24
3.10	Original image before Golay transform	27
3.11	Resultant image after Golay transform on subfield one . .	28
3.12	Resultant image after Golay transform on subfield two . .	29
3.13	Resultant image after Golay transform on subfield three (first iteration)	30
3.14	Resultant image after Golay transform on subfields one, two, and three (second iteration)	31
3.15	Resultant image after Golay transform on subfields one, two, and three (third and final iteration)	32

FIGURE PAGE

4.1 Space iterative combinational index recognition circuit for indexes one, two, and three 36

5.1 Organization of programmable tse computer 38

5.2 Logic of programmable tse computer 40

6.1 Two word encoded form of tse instructions 51

6.2 Input image to Golay's skeletonizing algorithm 53

6.3 Output image from Golay's skeletonizing algorithm 54

6.4 Input image to Golay's swelling algorithm 55

6.5 Output image from Golay's swelling algorithm 56

CHAPTER 1

INTRODUCTION

The handling of large amounts of data efficiently has become a major challenge. Because sequential machines are designed only to process data piece by piece, parallel processing machines offer a very desirable advantage in many cases. If large amounts of data can be handled simultaneously with a speed comparable to the speed of processing one piece of data on conventional machines, great improvements in speed can be made.

The goal of this research is to present a new class of computers using parallel processing concepts, describe one such computer in detail, interpret the instruction set of this computer using a sequential machine, and describe some uses for this computer. The subject of this interpretation is a programmable tse computer proposed by Rickard and Bodenheimer [1]. The word "tse" is the transliteration of the Chinese word for pictograph character [2].

Interpretation is only one technique among several used for modeling certain aspects of computer systems. These techniques are presented here in more detail.

Methods of Modeling

The modeling of a computer instruction set is useful in several ways. When one computer installation is being replaced by another, a model that allows users to begin programming using the new instructions or language on the old facility eases the transition. Also, the modeling of the old instruction set to use on the new facility allows users to convert at

their own rate. Instruction sets for which compilers have not been written or for which the computer system has not been built can utilize models to study the behavior of the instructions [3].

There are several modeling techniques from which to choose. Simulation, emulation, translation, and interpretation are considered for the study of the programmable tse computer.

Simulation is the modeling of a system in order to learn more about its behavior. The simulation of a system consists of a model which seems to behave as the system does. In reality, the model may look quite different from the system, but the performance of the model gives results which are equivalent to the results which would be given by the system [4]. The simulator should resemble all aspects which require study in the particular system [5]. Given a set of circumstances, the system and the model should obtain equivalent results in a similar way. In this way, an actual system may be studied and analyzed.

Emulation, which is very similar to simulation, uses the micro language of a micro programmed computer to carry out the modeling of a system [3]. Hardware and software are combined to increase the efficiency of the model, whereas simulation only uses software [6]. This method eliminates some of the cost in terms of computer time involved in simulation [3].

Usually, interpretation is used for instructions for which a machine has not yet been built. Translation is very similar except that it keeps the translation of an instruction to use again each time that statement is encountered. The interpreter, however, translates an instruction and branches to the appropriate statements to carry out the instruction every

time it is encountered instead of each time the instruction appears in the program [7]. Even though time is saved, the translator is also more difficult to write and may be longer in terms of written code [3]. The interpreter can be described as a monitor for instructions not recognized by the host computer [8].

The most basic difference between simulation and interpretation is that simulation actually models how the instructions are carried out as closely as possible, whereas interpretation does not. Interpretation only models the final result of each instruction.

Interpretation of Tse Instructions

Interpretation is used in this modeling of the tse computer instruction set. Each instruction proposed for the computer carries out a specific action which is interpreted. However, the method by which the interpreter performs the instruction is usually totally different from the method used by the proposed tse computer. The machine used in this interpretation is the PDP 11/40 which is a sequential machine. Consequently, the actions performed on the arrays are done in sequence instead of in parallel.

CHAPTER 2

TSE COMPUTERS

Presented in this chapter are some facts about tse computers. Their development has occurred mainly as a result of the need for processing arrays of data. Proposed components are also introduced along with information on their uses.

The images processed by the programmable tse computer presented in Chapter 5 are simply binary images. Note, however, that the proposed class of tse computers is not limited to black and white images alone. Any gray scale image can be represented by several planes of binary images. Assume, for instance, that there are only four possible gray levels. An image containing all of these gray levels can be represented by two binary planes. By allowing the brightest to the darkest gray levels to be represented by the digits 00, 01, 10, and 11, then the most significant digit of each gray level could be placed in one plane at the position where that gray level occurred in the original image, and the least significant digit could be placed in the other plane.

History of Tse Computers

The processing of two-dimensional data is presently performed on conventional computers one bit at a time. The Goddard Space Flight Center (GSFC), however, is designing new components for a new class of computers which use black and white images consisting of thousands of elements to be processed simultaneously instead of only a single element as their basic computational entity [2]. This is one concept of parallel processing. This class of computers has been given the name tse computers.

The achievement of much greater speeds is a major consideration. The components being developed operate on arrays of 128 by 128 bits (16,384 bits). If this new class of computers can operate on each bit of this array simultaneously in the same amount of time as a conventional computer operates on one bit, then the overall speed which can be obtained is increased by a factor of 16,384. Essentially 16,384 computers which can each perform the operations of a conventional computer are combined to make one powerful machine [2].

Components of Tse Computers

The basic operations proposed are ANDing, ORing, negation, and sliding. At GSFC, components are being developed which will be able to operate on all bits of a binary array simultaneously, or, in the case of ANDing and ORing, the operation will occur between all corresponding bits of two arrays simultaneously. ANDing performs the AND of corresponding bits of two arrays, ORing performs the OR of corresponding bits of two arrays, negation negates all elements of an array, and sliding translates all bits of an array in the X and/or Y direction [2]. Assuming that a black image represents logic one elements, and a white image represents logic zero elements, Figure 2.1 shows the basic operations.

The desired components are being designed using fiber-optic, semiconductor, and electroluminescent technology. Also, analog to digital conversion on each element of the array simultaneously is used to input an image [2]. Basically, optic fibers are used to perform data operations such as sliding, while photosensitive elements perform operations such as negation. Because two images must be brought together during ANDing and ORing operations, the photosensitive element used to perform the

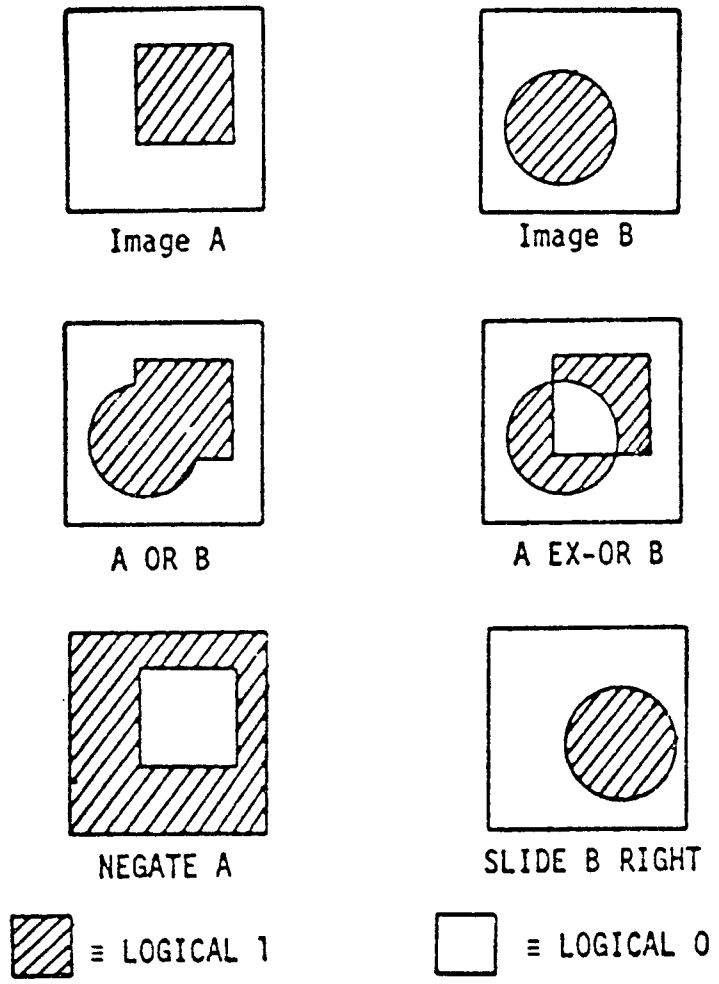


Figure 2.1. Examples of basic tse operations.

operation at each position in an array receives data from two optical fibers carrying data from elements of the two arrays in corresponding positions [2]. The desired operation is then performed and output. The device developed to bring corresponding elements of two arrays together in order to perform these operations is the interleaver (see Figure 2.2). Furthermore, by using the interleaver in reverse, a duplicator of images is created [2].

The sliding operation translates an image up, down, left, or right. After a slide down operation, the top rows are simply zero elements. The remaining rows consist of the original image shifted down with the bottom rows cut off. A loop slider takes the rows which would normally be cut off in a normal slide operation and inputs them into the top rows which would otherwise receive zeros. The device used in this operation is designed similarly to the interleaver [2].

Other components being proposed are an exchange gate, a reformatter, a cycler, a rotater, a vertical inverter, a horizontal inverter, a diagonal inverter, a horizontal sweeper, a vertical sweeper, a contractor, a spiller, a magnifier, and a demagnifier [1]. The exchange gate exchanges alternate rows of two images. This function is particularly useful in forming Golay neighbor planes as presented in Chapter 3. Another useful device is the contractor which outputs a logic one on a conventional logic line if any ones are present in the input image. The reformatter is simply used to adjust the output at certain gates to its proper form [1].

With the above components, combinations can be built to perform other operations. For example, the EXCLUSIVE-OR operation can be performed

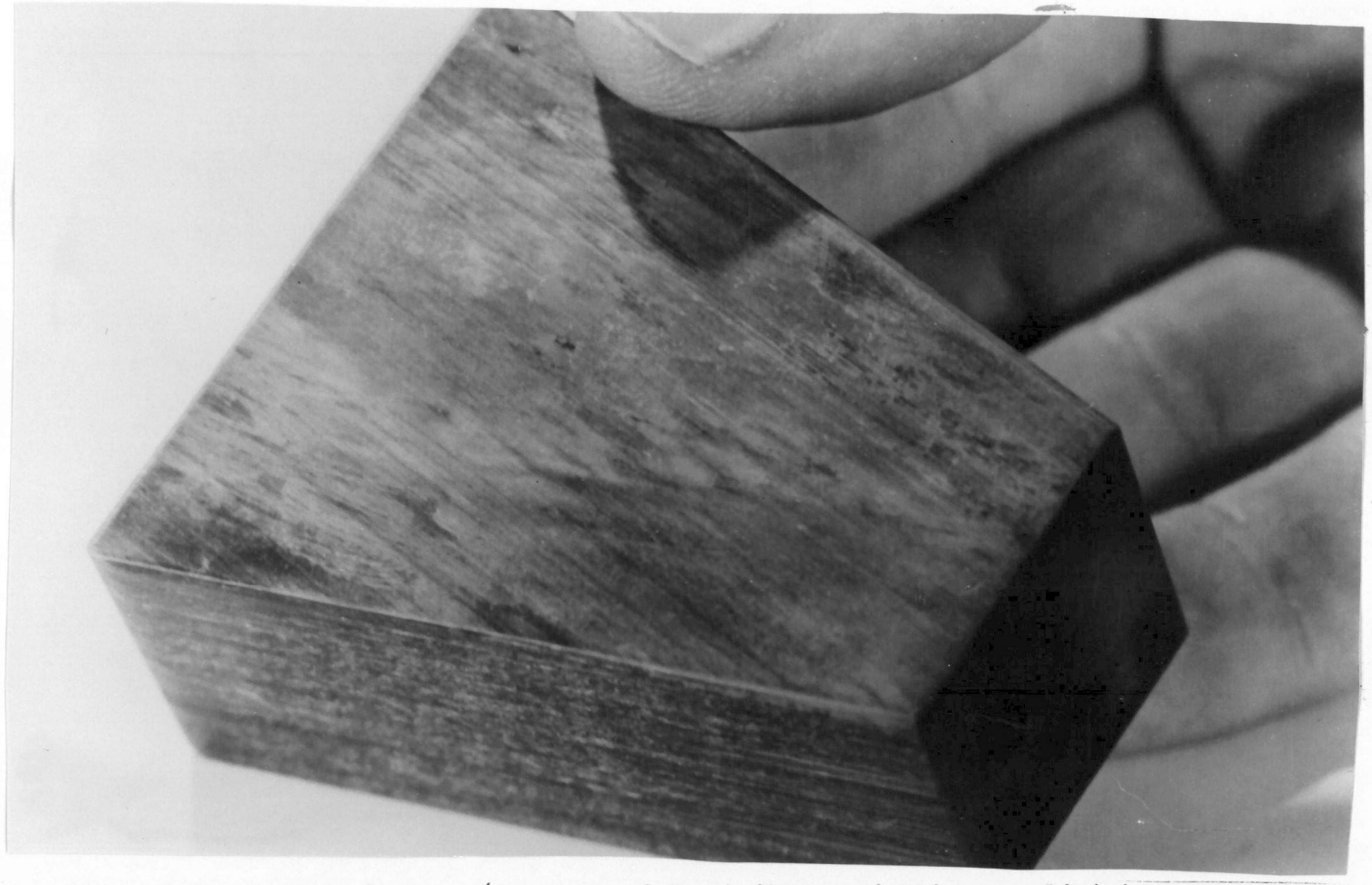


Figure 2.2. An interleaver, (Courtesy of Earth Observation Systems Division, Goddard Space Flight Center).

Source: David H. Schaefer and James P. Strong, III, "Tse Computers," Proc. IEEE, Vol. 65, p. 130, January 1977.

with a combination of negators, duplicators, combiners, and some other devices as seen in Figure 2.3 [2].

As stated previously, a tse computer can be considered to be a combination of thousands of computers which each operate on one point of an image. However, to be of use, these computers must be able to communicate with each other. This function is accomplished through devices such as the slide device which allows data from one position in an array to be passed to a new position in another array [2].

The hardware being proposed is bulky and consumes a relatively large amount of power, but consideration is being given to solving these problems, developing more complicated components, obtaining better speeds, and developing components which handle larger arrays of data [2].

Usefulness of Tse Computers

The most important advantage to be gained from the development of tse computers is high speed parallel computational ability [2]. In the next decade, NASA expects to receive as many as 50,000 images per day from spacecraft. The processing of this data along with images received from meteorological and planetary spacecraft is a formidable task [9]. Tse computers offer a faster method of processing this data. Matrix manipulation, which is often tedious on sequential machines, could also be facilitated by parallel processing machines [9]. Thus, tse computers offer great advantages over conventional computers simply by their uniqueness of design.

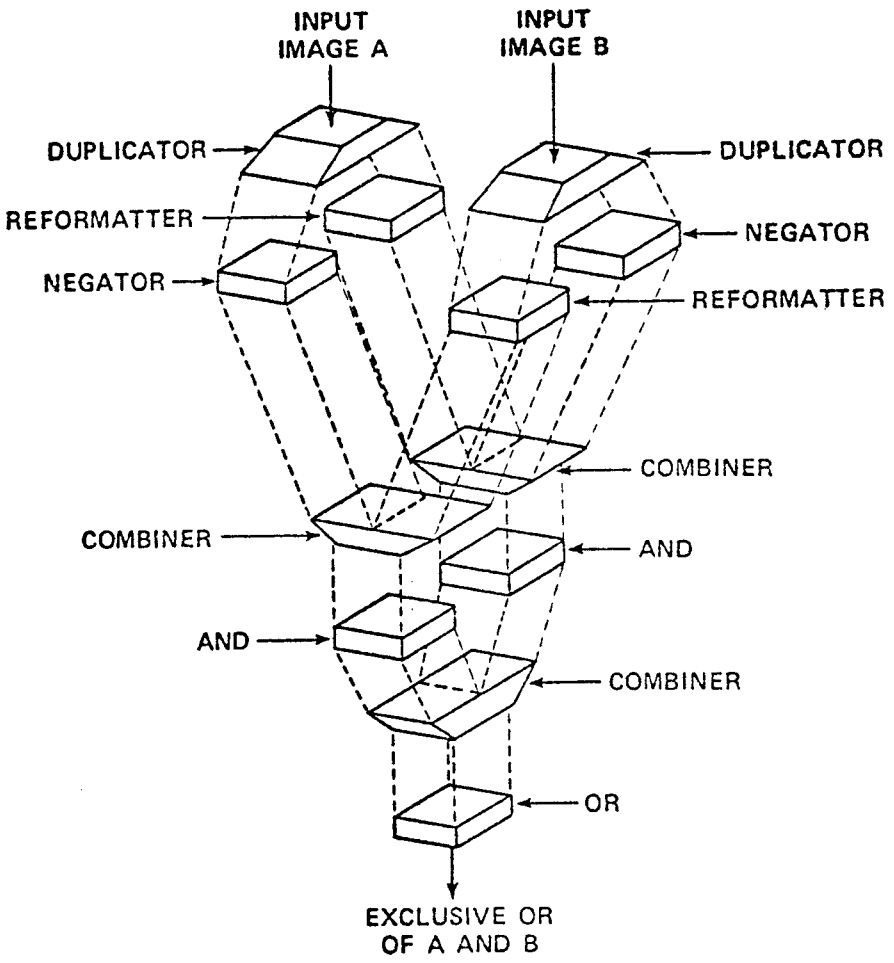


Figure 2.3. EXCLUSIVE-OR circuit.

Source: David H. Schaefer and James P. Strong, III, "Tse Computers," Proc. IEEE, Vol. 65, p. 138, January 1977.

CHAPTER 3

GOLAY TRANSFORMS

In this chapter Golay transforms are described in detail along with the process of hexagonal to rectangular array conversion. Golay's subfields, indexes, and neighbor planes, which are necessary ingredients for the implementation of Golay's functions, are also presented. Finally included are Golay's skeletonizing and swelling algorithms and information on how they work and their usefulness.

Square and Hexagonal Arrays

S. H. Unger first introduced the idea of parallel processing machines. The arrays he used were two-dimensional square arrays. Each element in a square array has a neighborhood of other elements surrounding it. If all rows and columns in an array are separated by a distance x, then each element has four closest neighbors at a distance x, four neighbors at a distance $x\sqrt{2}$, four neighbors at a distance 2x, eight neighbors at a distance $x\sqrt{5}$, etc. [10]. Figure 3.1 is an example of a square array.

The square array is useful in cases where the system to be studied is based on a rectangular set-up such as a chess game [10]. Unger also uses square arrays in some pattern recognition such as L-shaped figure detection, hole elimination in an otherwise solid image, and character recognition. Some other uses presented are filling in small notches and smoothing out small bumps on a straight edge [11]. However, in most pattern transformation cases certain problems arise. The transformation of a point is, of course, based on its neighborhood, but the question of whether to consider only the closest neighborhood when making the

U	N	J	O	V
M	E	B	F	P
I	A	P	C	K
T	H	D	G	Q
X	S	L	R	W

Figure 3.1. Square Array.

transformation on a point or whether to consider also the next closest neighborhood arises. If only the closest neighbors are considered, and the central element is transformed, then there may be a breakdown in their connectivity. To solve this problem, the next closest neighbors must also be considered. However, this would seem to make them equal in importance to the closest neighbors [12].

Golay [12] offers a solution that uses a hexagonal array (Figure 3.2) in which there are six equidistant neighbors spaced sixty degrees apart around the central point. These six neighbors contain enough information on which to base the transformation.

There is only one other type of array arrangement which is considered, and that is a triangular array (Figure 3.3). Problems similar to those which arise in the square array are present in pattern transformations using the triangular array [12].

The hexagonal array is the logical choice to use for pattern transformations because of its simplicity [12]. However, tse logic devices use rectangular arrays instead of hexagonal arrays. A method of modifying the hexagonal array to a form which can be easily manipulated in a rectangular arrangement must thus be considered [1].

Hexagonal to Rectangular Modification

Because tse logic devices use rectangular arrangements, the hexagonal pattern discussed must be modified to conform to rectangular patterns. The simplest way to achieve this is to shift every other row in Figure 3.4 (in this case the even rows) one-half the distance between horizontal elements to the left resulting in Figure 3.5 [1]. The neighborhood surrounding points E and H is indicated in both cases. The tse

B C

A P D

F E

Figure 3.2. Hexagonal Array.

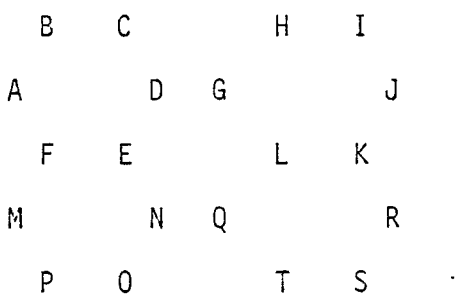


Figure 3.3. Triangular Array.

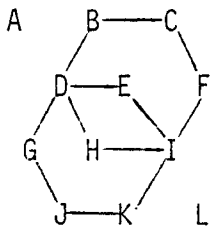


Figure 3.4. Hexangular arrangement of hexagonal pattern.

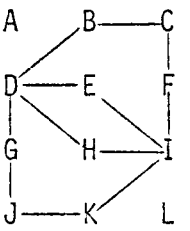


Figure 3.5. Rectangular arrangement of hexagonal pattern.

devices can now operate as planned using a rectangular arrangement. Furthermore, hexagonal pattern transformations may still be used providing the simplicity not found in rectangular or triangular pattern transformations.

Golay Subfields

In Golay pattern transformations, the state of a module can either be a one or a zero. Transformations on this module result as a function of the states of the six immediate neighbors called the surround of the module. Thus, two neighboring modules should not be dependent on each other at the time both are undergoing a transformation. If module one is being operated on as a function of module two, then module two must be stable during this operation. However, if module two is also undergoing a transformation, then confusion arises because the state of module two is unknown [12].

As a solution, Golay divides the hexagonal array into three subfields (Figure 3.6) which allow the operation on all modules of one subfield simultaneously. Each subfield is operated on in turn to take care of the whole array without confusion [12].

There is also the possibility of dividing the hexagonal array into four (Figures 3.7) or seven (Figure 3.8) independent subfields, but three subfields is the only case considered by Golay [12] or this thesis.

Golay Indexes

The whole basis of Golay's transformations is that the state of a module may be changed depending on the states of its six closest neighbors. A module may be either in state zero or state one with a module in state



Figure 3.6. Subdivision of hexagonal array into three subfields.

Source: Marcel J. E. Golay, "Hexagonal Parallel Pattern Transformations," IEEE Trans. Comp., Vol. C-18, p. 734, August 1969.



Figure 3.7. Subdivision of hexagonal array into four subfields.

Source: Marcel J. E. Golay, "Hexagonal Parallel Pattern Transformations," IEEE Trans. Comp., Vol. C-18, p. 735, August 1969.

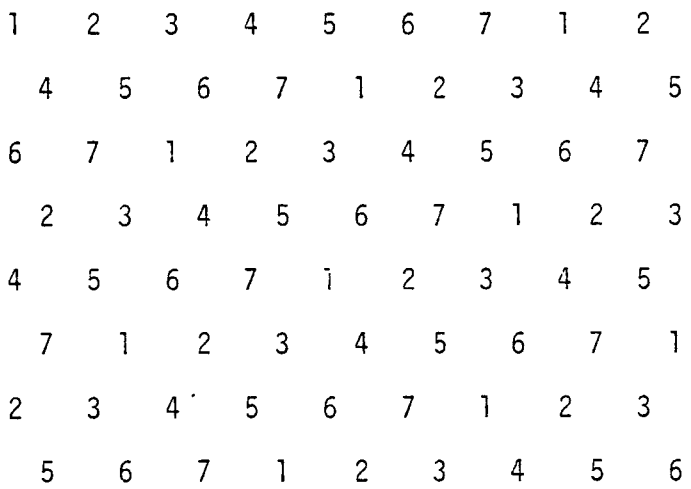


Figure 3.8. Subdivision of hexagonal array into seven subfields.

Source: Marcel J. E. Golay, "Hexagonal Parallel Pattern Transformations," IEEE Trans. Comp., Vol. C-18, p. 734, August 1969.

one being the subject of the transform and a module in state zero being the background.

Because each of the six neighbors of a module may be in either state zero or state one, 64 possible surrounds or patterns are formed. These 64 patterns are broken down first into two groups: one group containing only one set of adjacent ones and the other containing more than one set [12]. In the first group, a zero in the center module does not alter the connectivity of the ones of the surround. In other words, if two ones are connected (in the sense of being part of the same image) with a one in the center module, then they will still be connected with a zero in the center module [12].

These two groups can be further broken down into fourteen indexes. The weight of each index is the number of distinct surrounds which can be obtained from the rotation of the index's pattern in Table 3.1. Of course, the sum of the weights of all of the indexes totals 64. In this study of Golay transforms, two distinct patterns having the same index will be considered equal [12].

Golay Neighbor Planes

The function of Golay transforms is based on the six neighbors of a module $p_{i,j}$ of an array P. These six neighbors have varying positions in relation to $p_{i,j}$ depending on whether $p_{i,j}$ is in an even or an odd row. For example, if $p_{i,j}$ is in an even row, the six neighbors of $p_{i,j}$ are $p_{i,j-1}$, $p_{i-1,j}$, $p_{i-1,j+1}$, $p_{i,j+1}$, $p_{i+1,j+1}$, and $p_{i+1,j}$. However, if $p_{i,j}$ is in an odd row, the six neighbors are $p_{i,j-1}$, $p_{i-1,j-1}$, $p_{i-1,j}$, $p_{i,j+1}$, $p_{i+1,j}$, and $p_{i+1,j-1}$ as seen in Figure 3.9 [1].

TABLE 3.1

INDEXES OF GOLAY'S SURROUNDS

Pattern	Index	Weight
0 0 0 0 0 0	0	1
0 1 0 0 0 0	1	6
1 1 0 0 0 0	2	6
1 1 0 1 0 0	3	6
1 1 0 1 0 1	4	6
1 1 0 1 1 1	5	6
1 1 1 1 1 1	6	1
1 0 0 1 1 0	7	2
1 1 0 0 0 1	8	6
1 1 0 0 1 0	9	6
1 1 0 1 1 0	10	6
0 0 1 0 0 1	11	6
0 0 1 1 0 0	12	3
1 1 0 0 1 1	13	3

Source: Marcel J. E. Golay, "Hexagonal Parallel Pattern Transformations," IEEE Trans. Comp., Vol. C-18, p. 734, August 1969.

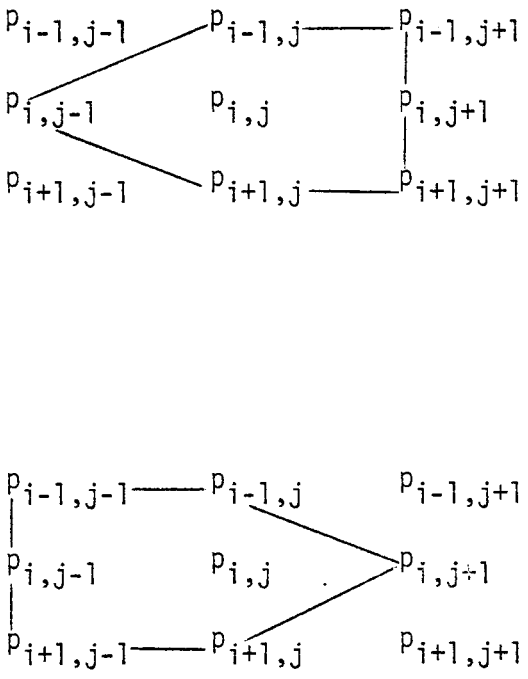


Figure 3.9. Neighbors of module $p_{i,j}$ in even and odd rows.

Using tse logic devices requires that each neighbor of a module $p_{i,j}$ can be accessed at position i,j. In order to process all six neighbors, six neighbor planes are required so that the kth (k being one of the six neighbors of $p_{i,j}$) neighbor plane, Q, contains the kth neighbor of $p_{i,j}$ in $q_{i,j}$. By labeling the six neighbors as in Figure 3.2 on page 14, plane A can be obtained by shifting P right one element in either the odd row or even row. Similarly, plane D is obtained by shifting P left one element [1].

Problems arise in creating planes B, C, E, and F. Consider the creation of plane B from the orientation of an even row. By shifting P one element down, plane B is created for only the even rows. Actually, plane C is created for the odd rows. In order to create the correct odd rows, P must be shifted down one element and right one element. The correct plane B will contain the odd rows of this array and the even rows of the previous array. Planes C, E, and F must be formed using similar logic [1]. The exchange gate facilitates this procedure very simply [12].

Golay's Skeletonizing and Swelling Algorithms

Golay [12] offers two algorithms used for the processing of binary arrays: the shrinking algorithm and the swelling algorithm.

In general, Golay's skeletonizing algorithm operates on a binary image in such a way that a group of ones is reduced to a single one, and a group of ones with simple holes made up of zeros is reduced to one or more loops with thicknesses of single ones. This is accomplished through a series of transformations and pattern matches.

The symbol

$$M^3_{1-3,n}\ (k=\overline{a_{i(k)}}\cdot k)$$

is used to define the skeletonizing operation. Whenever the surround of a module k has an index of one, two, or three (represented by the first subscript of M), then $a_{i(k)}=1$, $\overline{a_{i(k)}}=0$, and the transformation becomes $k=0\cdot k$ which changes the old state of k to the new state of zero. However, if any other index is recognized, then $a_{i(k)}=1$, and the module k is left in its previous state. The superscript of M represents the subfields which are to be operated on one at a time. One iteration consists of performing the transformation on each of the subfields once. When the result of the most recent iteration matches the result from the previous iteration, the process stops. The number of iterations required is represented by the second subscript of M. Observe that at the end of the process all modules have a surround with an index of zero or four through thirteen [12]. Figures 3.10 through 3.15 show steps of Golay transforms through the iterations required to obtain a result.

Another operation Golay introduces is the swelling operation defined by the symbol

$$M^3_{3-5,\infty}\ (k=a_{i(k)}+\overline{a_{i(k)}}\cdot k)\ .$$

The term $a_{i(k)}=1$ whenever the surround of module k has an index of three, four, or five. In any other case, $a_{i(k)}=0$. Basically, the operation smooths out the edges of a group of ones until the outline is convex and causes all holes consisting of a group of zeros surrounded by ones to reduce to a single zero [12].

```
0  0  0  0  0  0  0  0
  0  0  0  0  1  1  0  0
1  1  0  0  1  0  1  0
  1  1  0  1  0  1  1  0
1  1  0  0  1  1  1  0
  1  1  0  0  1  0  0  0
1  1  1  1  0  0  0  0
  0  0  0  0  0  0  0  0
```

Figure 3.10. Original image before Golay transform.

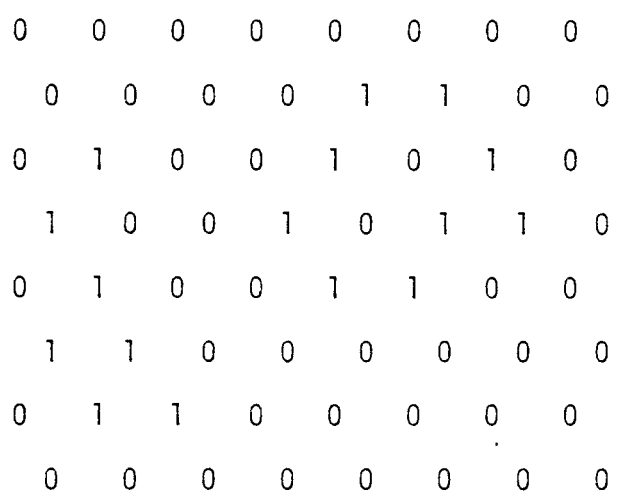


Figure 3.11. Resultant image after Golay transform on subfield one.

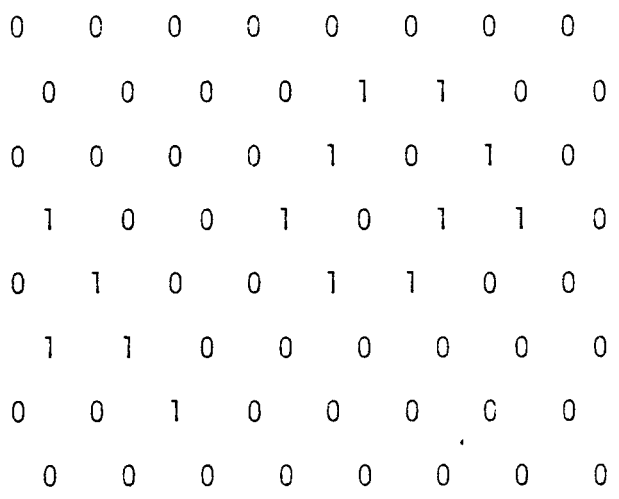


Figure 3.12. Resultant image after Golay transform on subfield two.

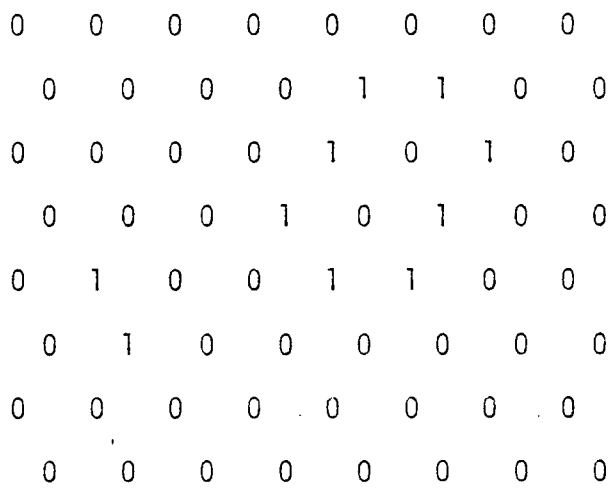


Figure 3.13. Resultant image after Golay transform on subfield three (first iteration).

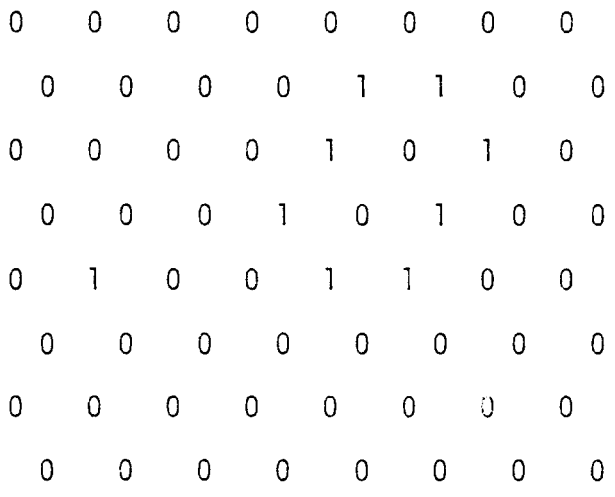


Figure 3.14. Resultant image after Golay transform on subfields one, two, and three (second iteration).

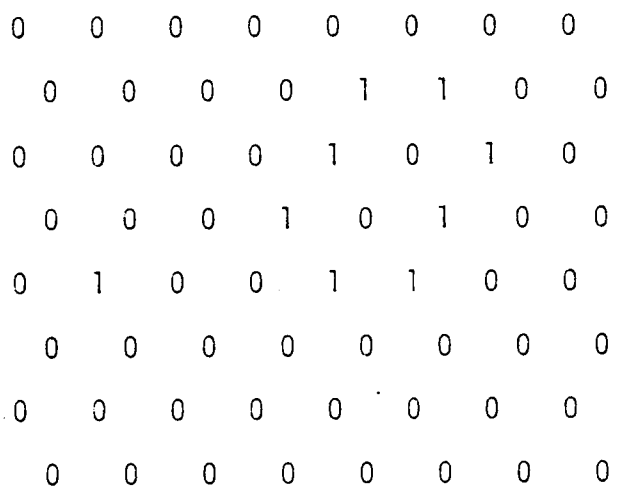


Figure 3.15. Resultant image after Golay transform on subfields one, two, and three (third and final iteration).

These types of functions together with other functions can be used to perform some very basic but useful operations. For example, consider an image containing blobs of ones, some of which have simple holes. The holes can be filled in by one operation, and then the blobs swollen and reduced to single ones. Thus the number of blobs can be obtained by merely counting the number of ones [12]. Other operations suggested by Golay include blob smoothing and skeletonizing of closed or open blobs [12].

CHAPTER 4

RELATIONSHIP BETWEEN TSE LOGIC DEVICES AND GOLAY TRANSFORMS

The goal of the research done on the programmable tse computer [1] is to produce tse processors suitable for implementing Golay transforms. In the work, several circuits consisting of tse components are designed to perform special functions such as generating Golay neighbor planes, recognizing certain Golay indexes, and performing Golay's skeletonizing and swelling algorithms.

Control, Masks, and Memory

The basic units consist of fiber optic bundles controlled by control images. These control images are true or false depending on whether they are all ones or all zeros [1].

Masks, which are necessary in providing the capability of processing only certain elements at a time, can be formed by several different methods. One method is to store a specific binary image in electro-optical devices and allowing this image to be ANDed or ORed to other images to obtain the desired result. Another method is to use a physical mask to allow data to flow along some fiber optic paths while inhibiting the flow in others [1].

Many tse operations are based on whether or not an image consists of all zero elements. The contractor and spiller devices [9] facilitate this type of image detection [1].

A more detailed description of the control of elementary tse processors is given in the research on the programmable tse computer.

Index Recognition Circuit Using Tse Components

Many special function processors can be designed using tse components. In fact, any logic function a conventional computer is capable of performing can be performed with tse components since the AND, OR, and negate tse components form a functionally complete set [1].

However, only one function is presented here. An iterative circuit which recognizes indexes one, two, and three from the six Golay neighborhood planes is designed to output an image with zeros designating the positions of the input image which have one of these surrounds. Each of the six cells of circuitry in Figure 4.1 recognizes one rotational orientation of indexes one, two, and three. See Appendix A for a definition of the symbols used. The outputs from the cells are ANDed together to form the final image [1].

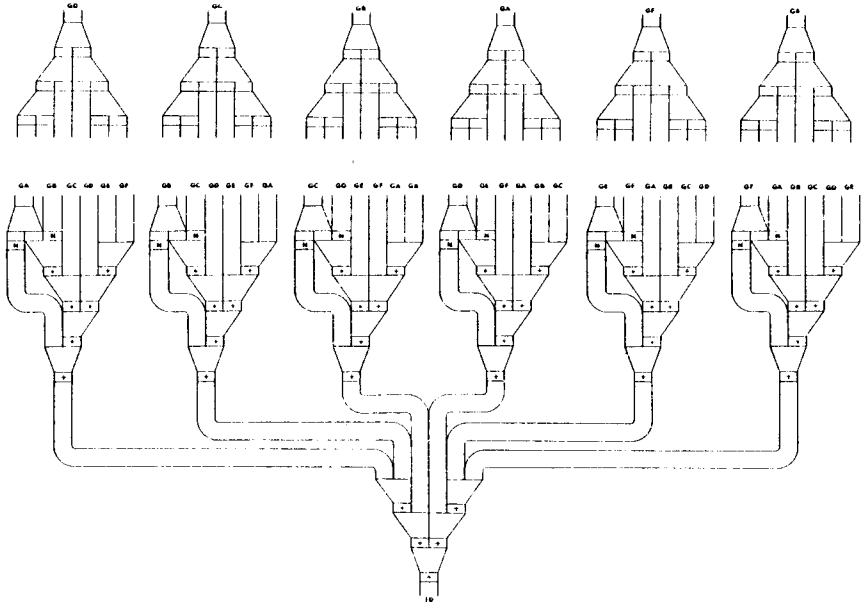


Figure 4.1. Space iterative combinational index recognition circuit for indexes one, two, and three.[1]

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1, The University of Tennessee, p. 70, September 1976.

CHAPTER 5

PROGRAMMABLE TSE COMPUTER

In previous examples, the iterative circuits which have been designed using tse components offer little or no flexibility. The functions they perform cannot be altered except by changing the hardware. However, research has been done on a programmable tse computer [1]. Much flexibility is achieved through the instruction set that has been especially tailored to perform Golay transforms. A description of this computer is presented along with details of the instruction set, information on the micro-computer based control unit, and programs which are written to perform Golay's skeletonizing and swelling algorithms [1].

Organization of Programmable Tse Computer

The computer performs Golay functions on images one subfield at a time. Five latches or registers are provided for manipulating images. Latch (A) is the accumulator, latches (B) and (C) are general purpose latches, latch (I) is used for index recognition, and latch (O) is used for output. The contractor is provided for detection of all zero images. Also included are an arithemetic logic unit (ALU) which performs AND, EXCLUSIVE-OR, and negate operations, an index recognition circuit, and a control unit. The ALU also provides a latch for temporary use when the source and destination registers are the same for an instruction [1]. Figure 5.1 shows the organization of these components.

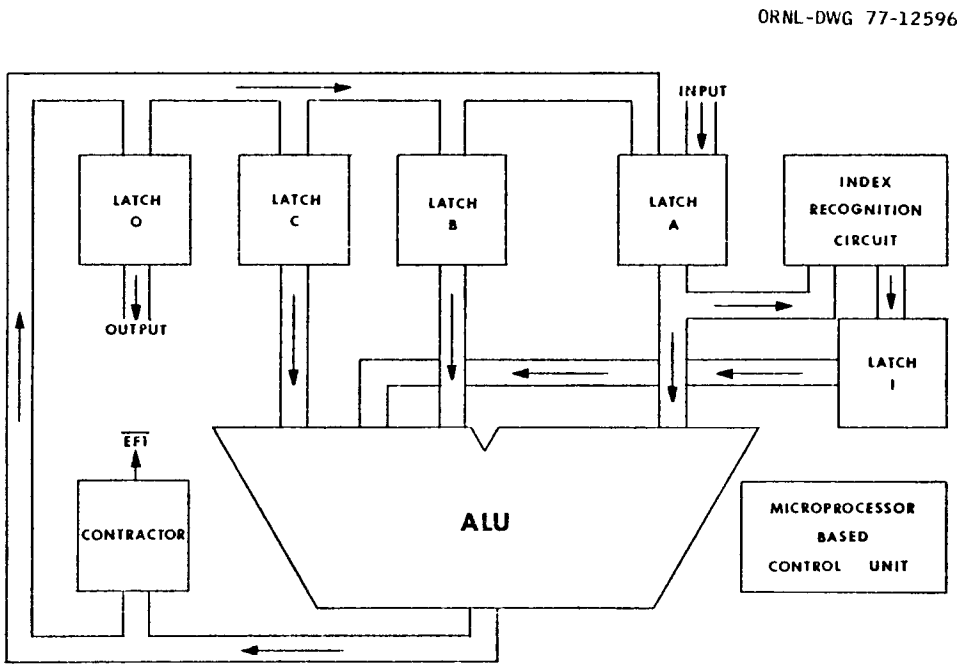


Figure 5.1. Organization of programmable tse computer.

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1, The University of Tennessee, p. 144, September 1976.

In general, latches (A), (B), and (C) are used as both source and destination registers. However, latch (I) is only used as a source register, and latch (O) is only used as a destination register [1].

The tse masks are used to operate on certain subfields of an array while not disturbing the remainder of the array. For example, if the instruction TAND B,I,M1 is executed, then subfield one of register (I) is ANDed to register (A) and the result stored in register (B). By ORing the complement of an array containing ones in subfield one (mask M1) with register (I), an array with register (I) in subfield one and ones in subfields two and three is obtained. This is then ANDed to register (A) without disturbing subfields two and three of register (A). Mask generators are contained in the ALU for this facility [1].

Also included in the organization are an index recognition circuit and a Golay neighborhood planes generator [1].

Figure 5.2 gives the logic of this tse processor. No extra memory is provided. If required, additional registers can be included [1].

Control Unit

RCA's CDP 1802 COSMAC microprocessor has been chosen as the control unit for this programmable tse computer. Control bytes are output to control the tse components of the computer. The correct timing is accomplished through timing loops. Monitors of the microprocessor detect all zero images, input requests, and external requests to halt the tse processing. Microinstructions are programmed to perform the desired control functions. In general, the COSMAC controls the execution of the tse computer [1].

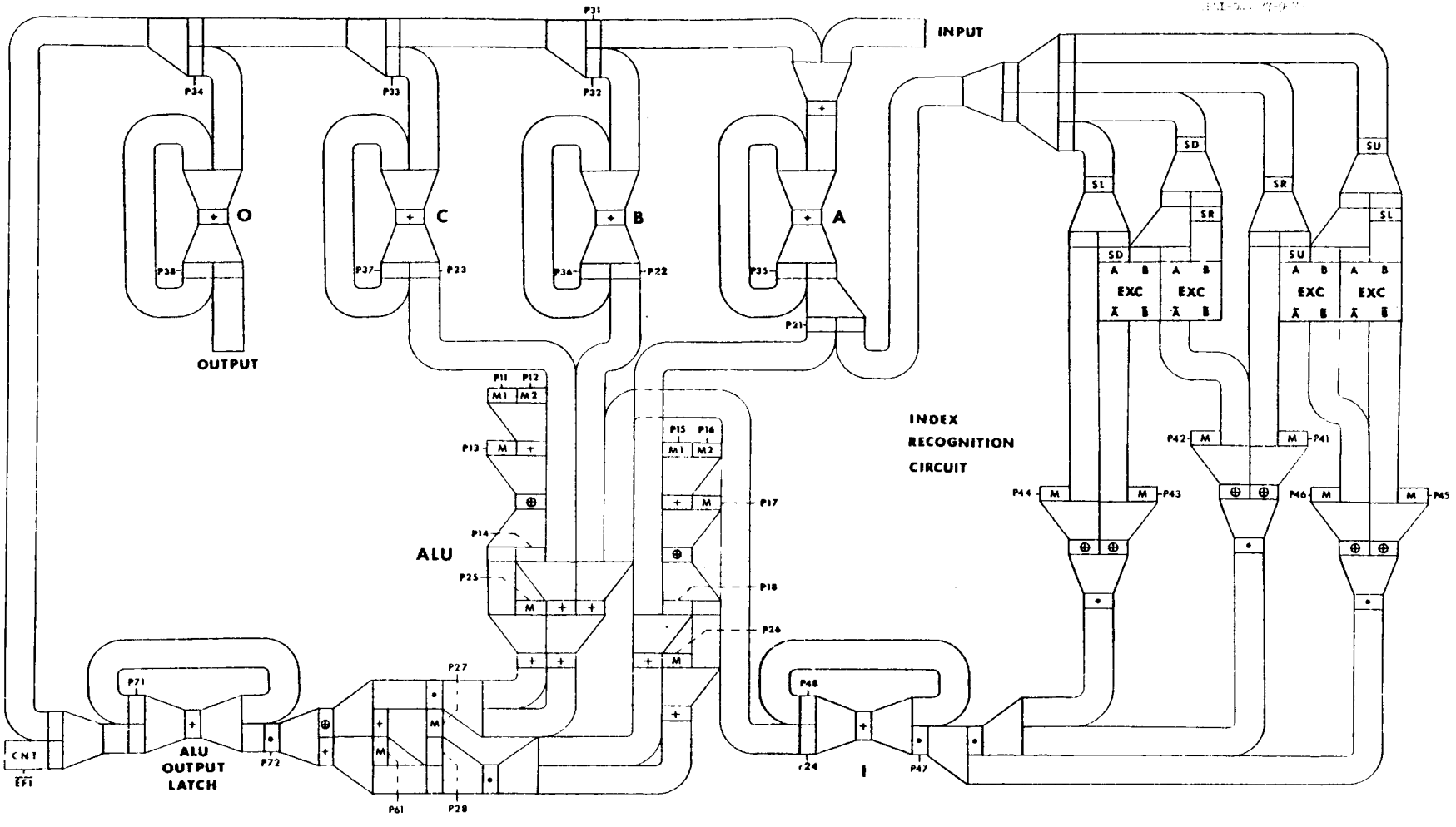


Figure 5.2. Logic of programmable tse computer.

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1, The University of Tennessee, p. 148, September 1976

Tse Computer Instruction Set

The tse computer instruction set presented here performs Golay functions such as index recognition with a great deal of flexibility. A list of the instructions is provided in Table 5.1 with the mnemonic and the result included. Some details about the instructions are also given.

In the table, the instructions are divided into six classes. The first class is the register operations class which provides internal manipulation of the data. The logic operations class performs ANDing, ORing, EXCLUSIVE-ORing and complementing operations on no, some, or all subfields of images depending on the mask that is specified. The index recognition class consists of only one special instruction. This is the index recognition function described previously. The class that performs compare operations provides the capability of testing and comparing images for certain logic values. For example, a certain function may be performed depending on whether or not subfield one of an image is all zeros. The register RF becomes important here because its value depends on the outcome of these instructions. This register is not part of the tse computer hardware, but of the COSMAC 1802 microcomputer used as the control unit of this machine. Branching, depending on the value of RF, is provided in another class. The final class is that of input and output operations. An input image is fed into latch (A), and a designated output image is moved into latch (O) [1].

TABLE 5.1

TSE INSTRUCTIONS

Instruction	Mnemonic	Operation
REGISTER OPERATIONS		
MOVE REGISTER TO REGISTER (SEE TOUT)	TMOV REG1,REG2	(REG2)→REG1
MOVE IMMEDIATE TO REGISTER	TMVI REG,MASK	(MASK)→REG
CLEAR REGISTER I	TCLRI	0→I
LOGIC OPERATIONS		
AND REGISTER TO A	TAND REG1, REG2, MASK	$[(\overline{MASK})+(REG2)]\cdot(A)\rightarrow REG1$
AND A TO REGISTER	TANA REG1,REG2,MASK	$[(\overline{MASK})+(A)]\cdot(REG2)\rightarrow REG1$
AND IMMEDIATE TO REGISTER	TANI REG1,REG2,MASK	$(MASK)\cdot(REG2)\rightarrow REG1$
AND $\overline{\text{REGISTER}}$ TO A	TANDN REG1,REG2,MASK	$[\overline{(MASK)\cdot(REG2)}]\cdot(A)\rightarrow REG1$
AND $\overline{A}$ TO REGISTER	TANAN REG1,REG2,MASK	$[\overline{(MASK)\cdot(A)}]\cdot(REG2)\rightarrow REG1$
OR REGISTER TO A	TOR REG,MASK	$[(MASK)\cdot(REG)]+(A)\rightarrow A$
OR A TO REGISTER	TORA REG,MASK	$[(MASK)\cdot(A)]+(REG)\rightarrow REG$
OR IMMEDIATE TO REGISTER	TORI REG,MASK	$(MASK)+(REG)\rightarrow REG$
OR $\overline{\text{REGISTER}}$ TO A	TORN REG,MASK	$\overline{[(\overline{MASK})+(REG)]}+(A)\rightarrow A$
OR $\overline{A}$ TO REGISTER	TORAN REG, MASK	$\overline{[(\overline{MASK})+(A)]}+(REG)\rightarrow REG$
EXCLUSIVE-OR REGISTER TO A	TXOR REG1,REG2,MASK	$[(MASK)\cdot(REG2)]\oplus(A)\rightarrow REG1$
EXCLUSIVE-OR A TO REGISTER	TXRA REG1,REG2,MASK	$[(MASK)\cdot(A)]\oplus(REG2)\rightarrow REG1$
EXCLUSIVE-OR IMMEDIATE TO REGISTER COMPLEMENT REGISTER	TXRI REG1,REG2,MASK TCMR REG1,REG2,MASK	$(MASK)\oplus(REG2)\rightarrow REG1$
INDEX RECOGNITION		
IDENTIFY BASIS POINTS WITH A SURROUND OF INDEX X	TIDA X	(I)+(ID)→I Where ID is a tse with 1's in positions which correspond to basis points of A which have a surround of index X.

TABLE 5.1 (Continued)

Instruction	Mnemonic	Operation
COMPARE OPERATIONS		
CONTRACT REGISTER	TCNT REG,MASK	(MASK)·(REG);IF=0, 0→RF IF≠0, 1→RF
TEST REGISTER	TTEST REG,MASK	$(\overline{MASK})$+(REG);IF=0, 0→RF IF≠0, 1→RF
COMPARE REGISTER	TCMP REG,MASK	[(MASK)·(REG)]⊕[(MASK)·(A)]; IF RESULT=0, 0→RF IF RESULT≠0, 1→RF
COMPARE IMMEDIATE	TCPI REG,MASK	(MASK)⊕(REG); IF(REG)=(MASK), 0→RF IF(REG)≠(MASK), 1→RF
TSE BRANCH OPERATIONS		
TSE SHORT BRANCH ON ZERO	TBZ ADR	IF RF=0,M(R(P))→R(P).0 ELSE R(P)+1
TSE SHORT BRANCH ON NO ZERO	TBNZ ADR	IF RF≠0,M(R(P))→R(P).0 ELSE R(P)+1
TSE LONG BRANCH ON ZERO	TLBZ ADR	IF RF=0,M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
TSE LONG BRANCH ON NO ZERO	TLBNZ ADR	IF RF=1,M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
INPUT/OUTPUT OPERATIONS		
INPUT TSE	TIN	INPUT TSE→A
OUTPUT TSE (SEE TMOV)	TOUT REG	(REG)→O

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol. 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1, The University of Tennessee, pp. 154–55, September 1976.

The arguments REG1, REG2, REG, and MASK are replaced by the various symbols appearing in Table 5.2. The restrictions given before for source and destination registers must be kept in mind, however.

In the register operations class, the instruction TMOV A,B moves the data in latch (B) to latch (A) without destroying the contents of latch (B). TCLRI simply places an all zero image in latch (I).

The logic operations instructions perform several varied operations. For example, instructions TAND and TANA seem to perform the same logical AND operation. However, on closer inspection, TAND C,B,M1 ANDs a certain portion of latch (B) to latch (A), whereas TANA C,B,M1 ANDs a certain portion of latch (A) to latch (B). Both place the result in latch (C). The ANDing of a mask to a register is another instruction. Also the ANDing of the complement of a register to latch (A) and the ANDing of the complement of latch (A) to a register are included. Similar ORing and EXCLUSIVE-ORing operations give additional capabilities. Note that the two instructions TXRI and TCMR perform exactly the same operation. After the execution of these and other instructions, register RF contains a zero if the resultant image is all zeros, and it contains a one otherwise. However, because of the hardware structure, ORing operations do not necessarily set the value of this register [1].

The argument for the TIDA instruction is replaced by the number of an index to be identified. By executing a series of TIDA instructions, latch (I) will contain ones in positions that have surrounds of any index given as an argument in those instructions.

TABLE 5.2

REGISTER AND MASK CONSTANT DEFINITIONS

Name	Symbol
Tse Registers	
Register A	A
Register B	B
Register C	C
Register O	O
Register I	I
Tse Masks	
All Zero Mask	MO
All One Mask	M
Subfield One Mask	M1
Subfield Two Mask	M2
Subfield Three Mask	M3
Subfields One and Two	M12
Subfields One and Three	M13
Subfields Two and Three	M23

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1, p. 157, September 1976.

Short and long branches are both provided. Short branches are limited to the current memory page, whereas long branches have no limit [1].

Programs to Perform Golay's Skeletonizing and Swelling Algorithms

Using the instruction set which is described previously, a program is written (see Table 5.3) which takes an input image and performs Golay's skeletonizing algorithm [1]. The program produces as output an image with single ones representing the original blobs of ones and single or multiple rings of ones representing the original blobs of ones containing simple holes.

In general, the program identifies positions with surround indexes of one, two, and three in each of the three subfields individually. Those positions identified in the original image are changed to logic zero if they are not already logic zero. The logic then branches through the same steps on this new image. The process repeats until the new resultant image is the same as the previous resultant image.

Similarly, Table 5.4 gives a program which performs Golay's swelling algorithm. This program identifies positions with surround indexes of three, four, and five on each of the subfields. Those positions identified become logic one if not already logic one. All other positions remain unchanged. The number of iterations necessary to obtain the result is determined in the same way as for the skeletonizing algorithm.

TABLE 5.3

PROGRAM TO PERFORM GOLAY'S SKELETONIZING ALGORITHM

Instruction			Comment
Start:	TIN		;input image for skeletonizing
Another:	TMOV	C,A	;save image in register C
	TCLRI		;clear index register
	TIDA	1	;recognize index one
	TIDA	2	;recognize index two
	TIDA	3	;recognize index three
	TANDN	A,I,M1	;perform Golay function operation
	TCLRI		;for the first subfield clear ;index register
	TIDA	1	;recognize index one
	TIDA	2	;recognize index two
	TIDA	3	;recognize index three
	TANDN	A,I,M2	;perform Golay function
	TCLRI		;for the second subfield clear ;index register
	TIDA	1	;recognize index one
	TIDA	2	;recognize index two
	TIDA	3	;recognize index three
	TANDN	A,I,M3	;perform Golay function operation
	TCMP	C,M	;for the third subfield compare ;iteration result with image
	TBNZ	Another	;saved in register C ;branch to perform another iteration
	TOUT	A	;if different ;otherwise output the ;skeleton image

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol. 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1. The University of Tennessee, p. 247, September 1976.

TABLE 5.4

PROGRAM TO PERFORM GOLAY'S SWELLING ALGORITHM

Instruction			Comment
Start:	TIN		;input image for swelling
Another:	TMOV	C,A	;save image in register C
	TCLRI		;clear index register
	TIDA	3	;recognize index three
	TIDA	4	;recognize index four
	TIDA	5	;recognize index five
	TANDN	A,I,M1	;perform Golay function operation
	TOR	I,M1	;on the first subfield in two ;operations
	TCLRI		;clear index register
	TIDA	3	;recognize index three
	TIDA	4	;recognize index four
	TIDA	5	;recognize index five
	TANDN	A,I,M2	;perform Golay function operation
	TOR	I,M2	;on the second subfield in two ;operations
	TCLRI		;clear index register
	TIDA	3	;recognize index three
	TIDA	4	;recognize index four
	TIDA	5	;recognize index five
	TANDN	A,I,M3	;perform Golay function operation
	TOR	I,M3	;on the third subfield in two ;operations
	TCMP	C,M	;compare result to image stored in ;register C
	TBNZ	Another	;branch to perform another iteration ;if deferrent
	TOUT	A	;if no difference output the swelled ;image

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol. 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1. The University of Tennessee, p. 252, September 1976.

CHAPTER 6

INTERPRETATION OF TSE COMPUTER INSTRUCTION SET

The interpretation of the instruction set in Table 5.1 on page 42 is carried out on the PDP 11/40 computer using Fortran and Macro which is the assembly language for the PDP 11/40. In general, each tse instruction causes the program's execution to branch to the appropriate set of statements to provide results identical to the final results of that instruction.

Outline of the Interpreter Program

The program actually consists of two parts. The main program, which is written in Fortran, controls the overall execution (see Appendix B). The secondary program, which is written in Macro, consists of many small subroutines which perform special functions (see Appendix C).

The main program begins by setting up storage for all arrays needed for tse masks, tse registers, and temporary results. The size of the image to be processed is read in. By calling special subroutines, the masks used in Golay transforms are defined. Next, two words at a time are read in from the memory used to store the encoded tse instructions. The proper fields are decoded giving enough information for the main program to know which instruction is to be interpreted and which tse registers to process. Depending on which instruction is being performed, the program branches to different sections of code. For example, if the instruction TMOV C,A is encountered, the main program moves image (or array) A to image (or array) C.

The subroutine program consists of special purpose functions called for when needed by the main program. When the main program is ANDing two arrays of data, the ITAND routine is called to AND two corresponding words from each array. Another important function of this secondary program is storing the tse program written to perform Golay transforms. Many macros are defined which recognize each tse instruction and set up two words of four half word fields containing encoded forms of the instruction, addresses, masks, and/or registers used in that particular instruction. As stated before, these fields contain values which are decoded by the main program giving it the necessary information to carry out the instruction.

Interpreter Registers, Masks, and Storage

Within the main program, arrays, which are mentioned before, are set up to correspond to each of the tse latches or registers. An image of size thirty-two by thirty-two is contained in an array dimensioned to thirty-two by two words. Since each word of the PDP 11/40 contains sixteen bits, each bit of the array represents one distinct point of the image. The option of using a sixteen by sixteen bit image is also available to the user.

Each mask defined in Table 5.2 on page 45 is stored in an array which is immediately available to the program as needed.

Figure 6.1 shows how the secondary program stores the encoded form of the tse instructions. Depending on which instruction is stored, the fields contain various values. For example, if the instruction TMOV C,A is stored, field four contains the lower byte of the value of the address of label LABEL in the instruction TBZ LABEL.

First Word

REG2 or REG (field one)	OPCODE (field two)

Second Word

MASK or UPPER BYTE OF ADDRESS (field three)	REG2 or LOWER BYTE OF ADDRESS (field four)

Figure 6.1. Two word encoded form of tse instructions.

Input and Output

Extra capabilities are provided in the input and output instructions to aid the user in displaying his data. Using special routines [13], input to latch (A) during a TIN instruction is received from a file. A Vititar camera displays a black and white image on a screen (Tektronix 632 Picture Monitor). By means of a sampling program [14], the image is put into a file on the storage device being used on the PDP 11/40. The main interpreter program then processes the data contained in this file. The output image in latch (0) during a TOUT instruction is stored in the same file. This file can then be displayed on the screen by the user through the use of a display program [14].

Figure 6.2 shows the input image to the interpreter program. The tse program contained in the memory of the interpreter program is Golay's skeletonizing algorithm. The result of this skeletonization is pictured in Figure 6.3. If the memory is changed to contain Golay's swelling algorithm, and Figure 6.4 is the input to the interpreter program, then Figure 6.5 is the result.

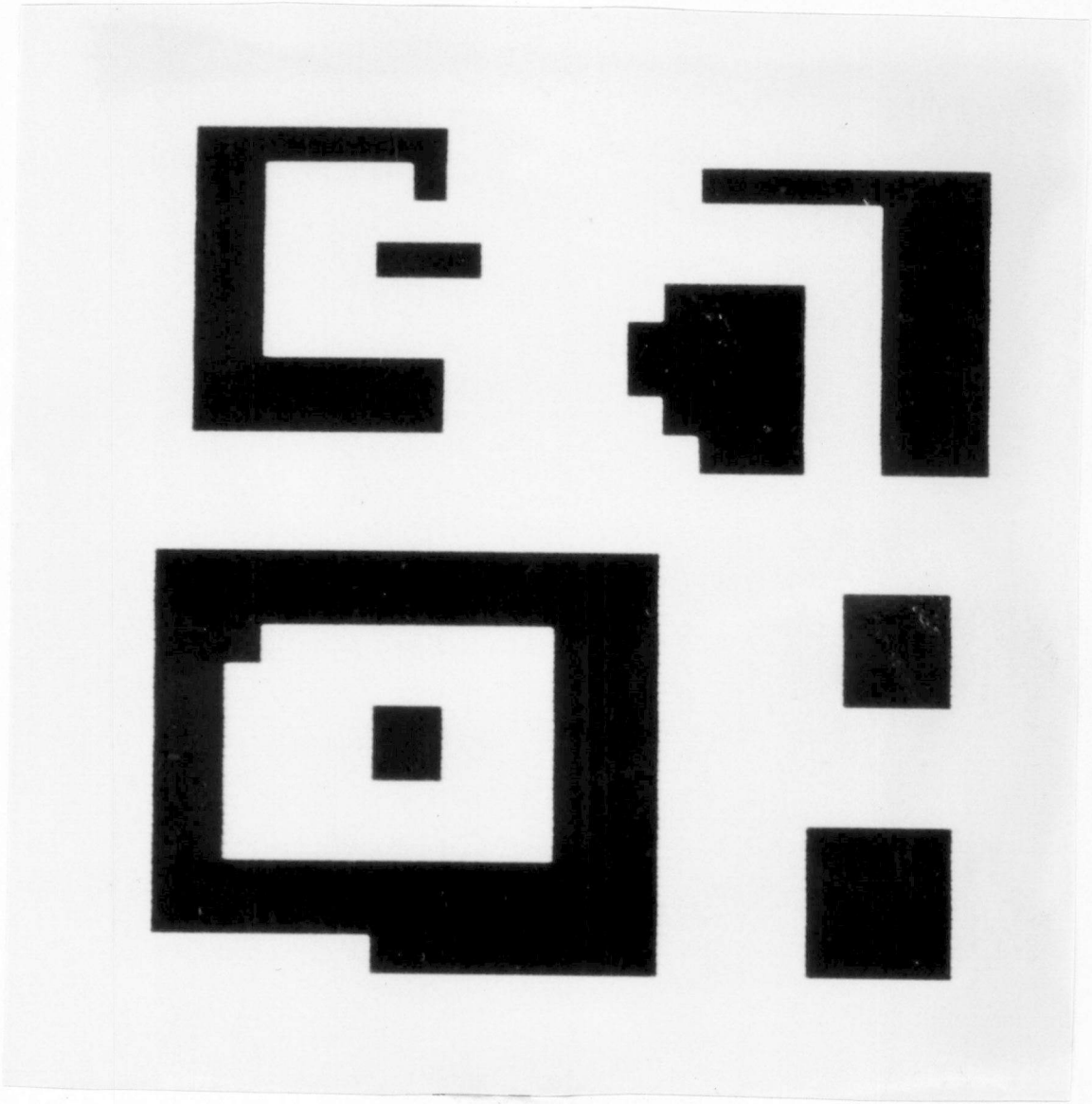


Figure 6.2. Input image to Golay's skeletonizing algorithm.

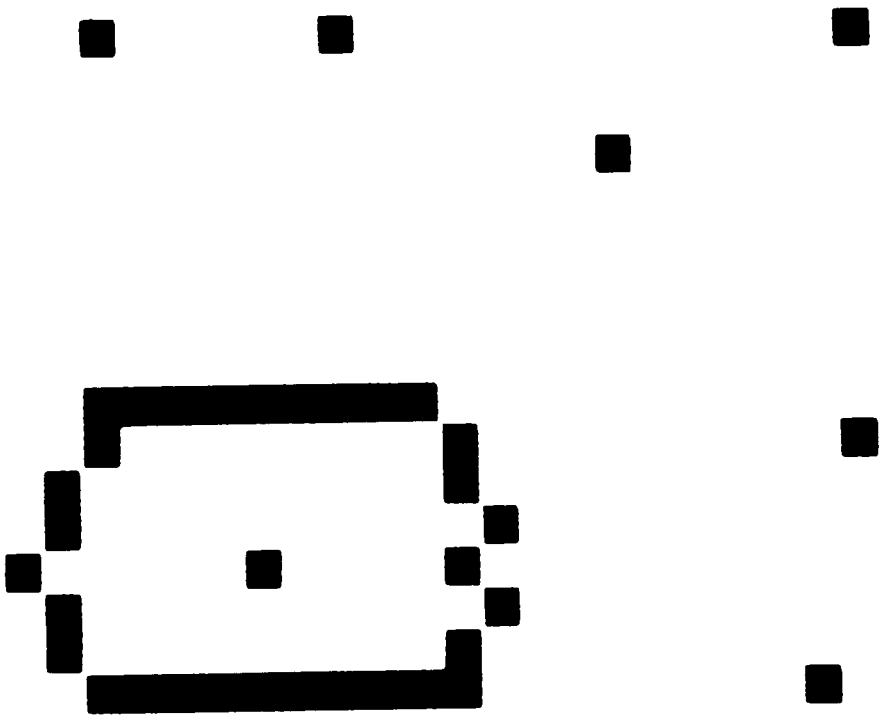


Figure 6.3. Output image from Golay's skeletonizing algorithm.

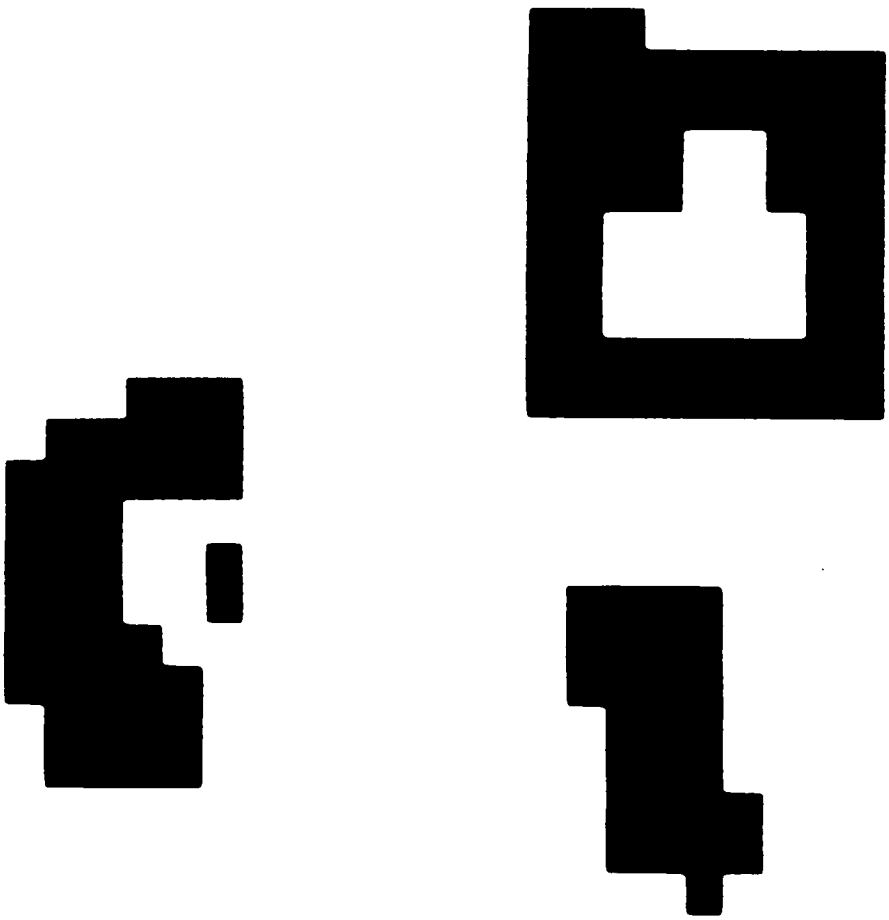


Figure 6.4. Input image to Golay's swelling algorithm.

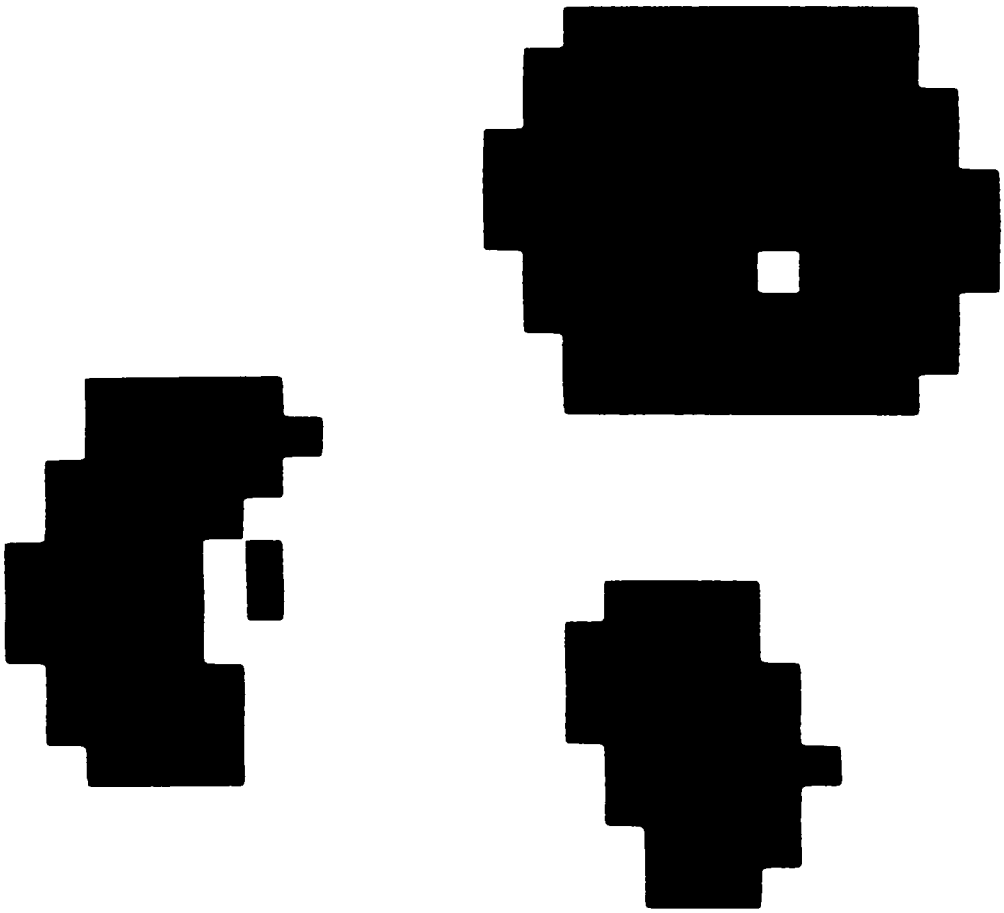


Figure 6.5. Output image from Golay's swelling algorithm.

CHAPTER 7

CONCLUSION

The interpreter for the programmable tse computer instruction set is useful in studying the actions of the computer. Even though the computer itself has not been built, the interpreter allows the instruction set to be used. Algorithms such as Golay's skeletonizing and swelling algorithm which operate on binary images can be implemented and the results studied.

Golay's transforms accept binary images as input. Blobs of ones are reduced to a single one, and blobs of ones containing simple holes are reduced to one or more rings consisting of single layers of ones in the skeltonizing algorithm. In the swelling algorithm, the edge of a blob of ones is smoothed out, and holes contained in a blob of ones are reduced to single zeros.

The programmable tse computer performs functions which are particularly useful in performing Golay operations. For example, the skeletonizing algorithm depends on identifying all elements of an image which have surrounds of indexes one, two, and three. The tse computer performs the function of forming an image with ones in positions corresponding to elements of the input image whose surround has a specified index. Tse components being designed at GSFC are used in the design of this computer. However, since the hardware necessary to construct this computer is only in a developmental stage, results using the tse instructions are not obtainable. The beauty of the interpretation of the instruction set is that image data can be processed, giving results identical to those that

would be given by the tse computer if it were constructed. By using conventional sequential machines, this advantage of parallel processing is not available; however, for the purpose of studying image processing results, this method is suitable.

Preston [15] suggests the use of Golay transforms to identify different types of human blood cells. One example he presents is that of distinguishing between lymphocyte and neutrophil white blood cells. He defines Glol statements which contract the nuclear images of the cells. The two classes of cells can be then distinguished by comparing the total number of binary ones per nucleus and the total number which have surround indexes one through five. The Glol statements used for contracting the images use Golay transform procedures which are available using the programmable tse computer. By adding extra instructions, the counting procedure for totaling binary ones in an image could also be included in the tse computer allowing this valuable function to biomedical science to be implemented.

Because the interpreter can perform all functions of the programmable tse computer which can in turn program Golay transforms, much image processing can be done by the interpreter. Since very simple images such as different types of white blood cells can be distinguished, certainly much more complicated images can be identified using a more powerful tse computer. Even the simple skeletonizing and swelling algorithms could be used to process very large and complicated images.

Because of the possibility of obtaining very high speed parallel processing capabilities, tse computers may become very important in processing the high volume of images expected by NASA from spacecraft [9]. Also, the processing of matrices may be facilitated by the development of tse computers. The interpreter allows algorithms for implementing these functions to be studied even before the high speed parallel processor is available.

The input to the interpreter program is formatted into either sixteen by sixteen or thirty-two by thirty-two bit images. These sizes are sufficient for this research to study some Golay transform results and the actions of the instruction set. The memory size of the system used for the interpretation imposes a limit on how much data can be stored. The thirty-two by thirty-two case is the largest case which can be contained in memory along with all the other data needed for the interpretation. Much larger images could be processed by either breaking the input image up into smaller images, or by using a system with a larger memory.

In conclusion, the interpreter for the programmable tse computer offers a workable method of obtaining results using the proposed instruction set. Even though the computer has not yet been built, the results from the interpreter are identical to those for which the computer is designed.

LIST OF REFERENCES

LIST OF REFERENCES

[1] D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol. 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1, The University of Tennessee, September 1976.

[2] David H. Schaefer and James P. Strong, III, "Tse Computers," Proc. IEEE, Vol. 65, pp. 129–138, January 1977.

[3] C. William Gear, Computer Organization and Programming, New York, McGraw-Hill, 1969.

[4] Robert E. Stephenson, Computer Simulation for Engineers, New York, Harcourt Brace Jovanovich, Inc., 1971.

[5] John McLeod, Simulation, New York, McGraw-Hill, 1968.

[6] Samir S. Husson, Microprogramming: Principles and Practices, Englewood Cliffs, N. J., Prentice Hall, Inc., 1970.

[7] Herbert Hellerman, Digital Computer System Principles, New York, McGraw-Hill, 1967.

[8] William H. Desmonde, Computers and Their Uses, Englewood Cliffs, N. J., Prentice Hall, Inc., 1964.

[9] David H. Schaefer and James P. Strong, III, Tse Computers, X-943-75-14, Goddard Space Flight Center, 1975.

[10] S. H. Unger, "A Computer Oriented Toward Spatial Problems," Proc. IRE, Vol. 46, pp. 1744–1750, October 1958.

[11] S. H. Unger, "Pattern Detection and Recognition," Proc. IRE, Vol. 47, pp. 1737–1752, October 1959.

[12] Marcel J. E. Golay, "Hexagonal Parallel Pattern Transformations," IEEE Trans. Comp., Vol. C-18, pp. 733–740, August 1969.

[13] RT-11 System Reference Manuel, Digital Equipment Corporation, Maynard, Mass., 1975.

[14] Image Processing Laboratory, Department of Electrical Engineering, University of Tennessee, Knoxville, Tennessee.

[15] Kendall Preston, Jr., "Feature Extraction by Golay Hexagonal Pattern Transforms," IEEE Trans. Comp., Vol. C-20, pp. 1007–1014, September 1971.

APPENDIXES

APPENDIX A

SCHEMATIC SYMBOLS FOR TSE LOGIC DEVICES

Table A.1 is a listing of all the symbols used in various figures throughout this thesis. Note that the interleaver is represented by three different symbols. This is done to simplify the figures.

TABLE A.1

SCHEMATIC SYMBOLS FOR TSE LOGIC DEVICES

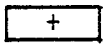
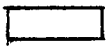
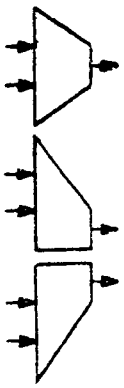
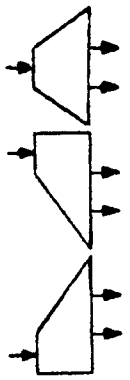
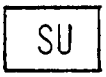
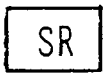
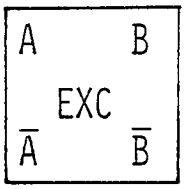
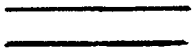
Device	Symbol
Active Devices	
AND	
OR	
NEGATE	
EXCLUSIVE-OR	
REFORMAT	
TOTAL SPILLER	
CONTRACTOR	
ROM	
Passive Devices	
INTERLEAVER AS A COMBINER	

TABLE A.1 (Continued)

Device	Symbol
INTERLEAVER AS A DUPLICATOR	
SLIDE UP	
SLIDE DOWN	
SLIDE RIGHT	
SLIDE LEFT	
EXCHANGE	
IMAGE BUS-LONG	
IMAGE BUS-SHORT	

Source: D. A. Rickard and R. E. Bodenheimer, "Architecture and Data Processing," Vol 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures, Technical Report TR-EE/CS-76-1, The University of Tennessee, pp. 187–89, September 1976.

APPENDIX B

MAIN PROGRAM

Table B.1 is the source listing of the main program for the interpreter written in Fortran. Arrays A, B, C, O and IND correspond to the latches of the programmable tse computer. The masks used in the instructions are represented by arrays MM, MO, M1, M2, M3, M12, M13, and M23.

TABLE B.1

MAIN PROGRAM

```
C
C  MAIN PROGRAM
C
C  PROGRAMMABLE TSE COMPUTER INTERPRETER
C
C
C  DEFINE STORAGE FOR TSE REGISTERS, TEMPORARY REGISTERS, MASKS,
C  NEIGHBOR PLANE REGISTERS, AND COMPLEMENT NEIGHBOR PLANES.
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION M(32,2),N(32,2),P(32,2),A(32,2),B(32,2),C(32,2),
     C   O(32,2),IND(32,2)
      DIMENSION MO(32,2),MM(32,2),M1(32,2),M2(32,2),M3(32,2),
     C   M12(32,2),M13(32,2),M23(32,2)
      DIMENSION AX(32,2),BX(32,2),CX(32,2),DX(32,2),EX(32,2),
     C   FX(32,2)
      DIMENSION NAX(32,2),NBX(32,2),NCX(32,2),NDX(32,2),
     C   NEX(32,2),NFX(32,2)
C  SET UP ARRAYS AND VARIABLES FOR DISPLAY OUTPUT.
      LOGICAL*1 LPIC(128,128),LP1(16384)
      INTEGER*2 FILSPC(39),DEFEXT(4)
      DATA DEFEXT/4*3RPIC/
      EQUIVALENCE (LPIC(1),LP1(1))
C
C  READ IN THE SIZE OF THE IMAGE (16 BY 16 OR 32 BY 32).
C
      WRITE (7,4)
4     FORMAT (1X,'INPUT THE SIZE OF THE IMAGE.'/2X,
     C  'TYPE A BLANK FOLLOWED BY 16 OR 32')
      READ (7,1) SIZE
1     FORMAT (I3)
      PSIZE=SIZE/16
C
C  SET UP THE TSE MASKS.
C
C  SET UP MO
      DO 30 I=1,SIZE
      DO 31 J=1,PSIZE
31    MO(I,J)=0
30    CONTINUE
C  SET UP MM
      CALL DEFMM(ROW)
      DO 32 I=1,SIZE
      DO 33 J=1,PSIZE
33    MM(I,J) = ROW
32    CONTINUE
C  SET UP M1
      CALL DEFM1(ROW1,ROW2)
      CALL DEFINE(SIZE,PSIZE, M1,ROW1,ROW2)
C  SET UP M2
      CALL DEFM2(ROW1,ROW2)
      CALL DEFINE(SIZE,PSIZE,M2,ROW1,ROW2)
C  SET UP M3
      CALL DEFM3(ROW1,ROW2)
      CALL DEFINE(SIZE,PSIZE,M3,ROW1,ROW2)
C  SET UP M12
      CALL DEFM12(ROW1,ROW2)
      CALL DEFINE(SIZE,PSIZE,M12,ROW1,ROW2)
```

TABLE B.1 (Continued)

```
C  SET UP M13
        CALL DEFM13(ROW1,ROW2)
        CALL DEFINE(SIZE,PSIZE,M13,ROW1,ROW2)
C  SET UP M23
        CALL DEFM23(ROW1,ROW2)
        CALL DEFINE(SIZE,PSIZE,M23,ROW1,ROW2)
C
C  INITIALIZE THE COUNTROL COUNTER (CC).
C
        CC=0
2       MAR=CC
C
C  READ IN THE ENCODED FORM OF A TSE INSTRUCTION FROM MEMORY INTO
C  MDR1 AND MDR2.
C
        CALL READ(MAR,MDR1,MDR2)
C
C  UPDATE CC.
C
        CC=CC+4
C
C  DECODE MDR1 AND MDR2 INTO THE FIELDS INSTRU,
C  FLD1, FLD2, AND FLD3.
C
        CALL DCODE(MDR1,MDR2,INSTRU,FLD1,FLD2,FLD3)
C
C  BRANCH TO PROPER SET OF STATEMENTS TO EXECUTE CURRENT TSE
C  INSTRUCTION.
C
        IF (INSTRU .EQ. 18) GO TO 300
        IF ((INSTRU .GE. 23) .AND. (INSTRU .LE. 26)) GO TO 400
        IF (INSTRU .EQ. 27) GO TO 500
        IF (INSTRU .EQ. 29) GO TO 600
        IF (FLD1. EQ. 0) GO TO 110
C
C  ARRAYS M,N, AND P ARE TEMPORARY STORAGE FOR THE IMAGES TO BE
C  OPERATED UPON DEPENDING ON THE CURRENT TSE INSTRUCTION.
C
        DO 100 I=1,SIZE
        DO 101 J=1,PSIZE
        GO TO (102,103,104,105,106), FLD1
102     M(I,J) = A(I,J)
        GO TO 101
103     M(I,J)=B(I,J)
        GO TO 101
104     M(I,J)=C(I,J)
        GO TO 101
105     M(I,J)=D(I,J)
        GO TO 101
106     M(I,J)=IND(I,J)
101     CONTINUE
100     CONTINUE
110     IF (FLD2 .EQ. 0) GO TO 120
        DO 111 I=1,SIZE
        DO 112 J=1,PSIZE
        GO TO (113,114,115,116,117), FLD2
113     N(I,J)=A(I,J)
        GO TO 112
```

TABLE B.1 (Continued)

```
114      N(I,J)=B(I,J)
         GO TO 112
115      N(I,J)=C(I,J)
         GO TO 112
116      N(I,J)=D(I,J)
         GO TO 112
117      N(I,J)=IND(I,J)
112      CONTINUE
111      CONTINUE
120      IF (FLD3 .EQ. 0) GO TO 140
         DO 121 I=1,SIZE
         DO 122 J=1,PSIZE
         GO TO (123,124,125,126,127,128,129,130), FLD3
123      P(I,J)=M0(I,J)
         GO TO 122
124      P(I,J)=MM(I,J)
         GO TO 122
125      P(I,J)=M1(I,J)
         GO TO 122
126      P(I,J)=M2(I,J)
         GO TO 122
127      P(I,J)=M3(I,J)
         GO TO 122
128      P(I,J)=M12(I,J)
         GO TO 122
129      P(I,J)=M13(I,J)
         GO TO 122
130      P(I,J)=M23(I,J)
122      CONTINUE
121      CONTINUE
C
C  BRANCH TO PROPER SET OF STATEMENTS TO EXECUTE THE CURRENT
C  TSE INSTRUCTION.
C
140      GO TO (150,160,170,180,190,210,220,230,240,250,260,
     C  270,280,290,310,320,330,300,360,370,380,390,
     C  400,400,400,400,500,600,800), INSTRU
C
C
C  TSE INSTRUCTIONS
C
C
C
C  TMOV INSTRUCTION
C
150      DO 151 I=1,SIZE
         DO 152 J=1,PSIZE
152      CALL ITMVI(M(I,J),N(I,J))
151      CONTINUE
         CALL RFSUB(M,SIZE,PSIZE,RF)
         GO TO 550
C
C  TMVI INSTRUCTION
C
```

TABLE B.1 (Continued)

```
160     DO 161 I=1,SIZE
        DO 162 J=1,PSIZE
162     CALL ITMVI(M(I,J),P(I,J))
161     CONTINUE
        CALL RFSUB(M,SIZE,PSIZE,RF)
        GO TO 550
C
C  TCLRI INSTRUCTION
C
170     DO 201 I=1,SIZE
        DO 202 J=1,PSIZE
202     IND(I,J)=0
201     CONTINUE
        GO TO 2
C
C  TAND INSTRUCTION
C
180     DO 181 I=1,SIZE
        DO 182 J=1,PSIZE
182     CALL ITAND(A(I,J),M(I,J),N(I,J),P(I,J))
181     CONTINUE
        CALL RFSUB(M,SIZE,PSIZE,RF)
        GO TO 550
C
C  TANA INSTRUCTION
C
190     DO 191 I=1,SIZE
        DO 192 J=1,PSIZE
192     CALL ITAND(N(I,J),M(I,J),A(I,J),P(I,J))
191     CONTINUE
        CALL RFSUB(M,SIZE,PSIZE,RF)
        GO TO 550
C
C  TANI INSTRUCTION
C
210     DO 211 I=1,SIZE
        DO 212 J=1,PSIZE
212     CALL ITANI(M(I,J),N(I,J),P(I,J))
211     CONTINUE
        CALL RFSUB(M,SIZE,PSIZE,RF)
        GO TO 550
C
C  TANDN INSTRUCTION
C
220     DO 221 I=1,SIZE
        DO 222 J=1,PSIZE
222     CALL ITANDN(A(I,J),M(I,J),N(I,J),P(I,J))
221     CONTINUE
        CALL RFSUB(M,SIZE,PSIZE,RF)
        GO TO 550
C
C  TANAN INSTRUCTION
C
230     DO 231 I=1,SIZE
        DO 232 J=1,PSIZE
232     CALL ITANDN(N(I,J),M(I,J),A(I,J),P(I,J))
231     CONTINUE
        CALL RFSUB(M,SIZE,PSIZE,RF)
        GO TO 550
```

TABLE B.1 (Continued)

```
C
C  TOR INSTRUCTION
C
240      DO 241 I=1,SIZE
         DO 242 J=1,PSIZE
         IF ((I .EQ. 1) .AND. (J .EQ. 1)) RF=0
242      CALL ITOR(A(I,J),M(I,J),P(I,J),RF)
241      CONTINUE
         GO TO 2
C
C  TORA INSTRUCTION
C
250      DO 251 I=1,SIZE
         DO 252 J=1,PSIZE
         IF ((I .EQ. 1) .AND. (J .EQ. 1)) RF=0
252      CALL ITOR(M(I,J),A(I,J),P(I,J),RF)
251      CONTINUE
         GO TO 550
C
C  TORI INSTRUCTION
C
260      DO 261 I=1,SIZE
         DO 262 J=1,PSIZE
262      CALL ITORI(M(I,J),P(I,J))
261      CONTINUE
         CALL RFSUB(P,SIZE,PSIZE,RF)
         GO TO 550
C
C  TORN INSTRUCTION
C
270      DO 271 I=1,SIZE
         DO 272 J=1,PSIZE
         IF ((I .EQ. 1) .AND. (J .EQ. 1)) RF=0
272      CALL ITORN(A(I,J),M(I,J),P(I,J),RF)
271      CONTINUE
         GO TO 2
C
C  TORAN INSTRUCTION
C
280      DO 281 I=1,SIZE
         DO 282 J=1,PSIZE
         IF ((I .EQ. 1) .AND. (J .EQ. 1)) RF =0
282      CALL ITORN(M(I,J),A(I,J),P(I,J),RF)
281      CONTINUE
         GO TO 550
C
C  TXOR INSTRUCTION
C
290      DO 291 I=1,SIZE
         DO 292 J=1,PSIZE
292      CALL ITXOR(A(I,J),M(I,J),N(I,J),P(I,J))
291      CONTINUE
         CALL RFSUB(M,SIZE,PSIZE,RF)
         GO TO 550
```

TABLE B.1 (Continued)

```
C
C  TXRA INSTRUCTION
C
310      DO 311 I=1,SIZE
         DO 312 J=1,PSIZE
312      CALL ITXOR(N(I,J),M(I,J),A(I,J),P(I,J))
311      CONTINUE
         CALL RFSUB(M,SIZE,PSIZE,RF)
         GO TO 550
C
C  TXRI INSTRUCTION
C
320      DO 321 I=1,SIZE
         DO 322 J=1,PSIZE
322      CALL ITXRI(M(I,J),N(I,J),P(I,J))
321      CONTINUE
         CALL RFSUB(M,SIZE,PSIZE,RF)
         GO TO 550
C
C  TCMR INSTRUCTION
C
330      DO 331 I=1,SIZE
         DO 332 J=1,PSIZE
332      CALL ITXRI(M(I,J),N(I,J),P(I,J))
331      CONTINUE
         CALL RFSUB(M,SIZE,PSIZE,RF)
         GO TO 2
C
C  TCNT INSTRUCTION
C
360      DO 361 I=1,SIZE
         DO 362 J=1,PSIZE
362      CALL ITCNT(M(I,J),N(I,J),P(I,J))
361      CONTINUE
         CALL RFSUB(N,SIZE,PSIZE,RF)
         GO TO 2
C
C  TTEST INSTRUCTION
C
370      DO 371 I=1,SIZE
         DO 372 J=1,PSIZE
372      CALL ITTEST(M(I,J),N(I,J),P(I,J))
371      CONTINUE
         CALL RFSUB(N,SIZE,PSIZE,RF)
         GO TO 2
C
C  TCMP INSTRUCTION
C
380      DO 381 I=1,SIZE
         DO 382 J=1,PSIZE
382      CALL ITCMP(M(I,J),N(I,J),P(I,J),A(I,J))
381      CONTINUE
         CALL RFSUB(N,SIZE,PSIZE,RF)
         GO TO 2
```

TABLE B.1 (Continued)

```
C
C  TCPI INSTRUCTION
C
390      DO 391 I=1,SIZE
         DO 392 J=1,PSIZE
392      CALL ITCPI(M(I,J),N(I,J),P(I,J))
391      CONTINUE
         CALL RFSUB(N,SIZE,PSIZE,RF)
         GO TO 2
550      DO 551 I=1,SIZE
         DO 552 J=1,PSIZE
         GO TO (553,554,555,556,557), FLD1
553      A(I,J)=M(I,J)
         GO TO 552
554      B(I,J)=M(I,J)
         GO TO 552
555      C(I,J)=M(I,J)
         GO TO 552
556      O(I,J)=M(I,J)
         GO TO 552
557      IND(I,J)=M(I,J)
552      CONTINUE
551      CONTINUE
         GO TO 2
C
C  INDEX INSTRUCTION
C
C  POSITION 'X' CORRESPOUNDS TO THE XTH POSITION
C  OF THE EIGHT NEIGHBORS OF MODULE A(I,J).
C     1 2 3
C
C     8 A 4
C
C     7 6 5
C
C  NEIGHBORHOOD PLANES A,B,C,D,E, AND F ARE SET UP.
C
C  GET POSITION 2 AND PUT IN NAX
300      DO 301 I=2,SIZE
         DO 302 J=1,PSIZE
302      NAX(I,J)=A(I-1,J)
301      CONTINUE
         DO 303 J=1,PSIZE
303      NAX(1,J)=0
C  GET POSITION 3 AND PUT IN NBX
         DO 304 I=1,SIZE
         DO 305 J=1,PSIZE
         NBX(I,J)=NAX(I,J)*2
         IF (J .EQ. PSIZE) GO TO 305
         IF (NAX(I,J+1) .LT. 0) NBX(I,J)=NBX(I,J)+1
305      CONTINUE
304      CONTINUE
C  GET POSITION 4 AND PUT IN DX (THIS SETS UP NEIGHBORHOOD D)
         DO 306 I=1,SIZE
         DO 307 J=1,PSIZE
         DX(I,J)=A(I,J)*2
         IF (J .EQ. PSIZE) GO TO 307
         IF (A(I,J+1) .LT. 0) DX(I,J)=DX(I,J)+1
```

TABLE B.1 (Continued)

```
307      CONTINUE
306      CONTINUE
C  GET POSITION 5 AND PUT IN NCX
         DO 308 I=1,SIZE-1
         DO 309 J=1,PSIZE
309      NCX(I,J)=DX(I+1,J)
308      CONTINUE
         DO 313 J=1,PSIZE
313      NCX(SIZE,J)=0
C  GET POSITION 6 AND PUT IN NEX
         DO 314 I=1,SIZE-1
         DO 315 J=1,PSIZE
315      NEX(I,J)=A(I+1,J)
314      CONTINUE
         DO 316 J=1,PSIZE
316      NEX(SIZE,J)=0
C  GET POSITION 8 AND PUT IN AX (THIS SET UP NEIGHBORHOOD A)
         DO 317 I=1,SIZE
         CARRY =0
         DO 318 J=1,PSIZE
318      CALL RSHFT (A(I,J),AX(I,J),CARRY)
317      CONTINUE
C  GET POSITION 1 AND PUT IN NFX
         DO 319 I=2,SIZE
         DO 340 J=1,PSIZE
340      NFX(I,J)=AX(I-1,J)
319      CONTINUE
         DO 341 J=1,PSIZE
341      NFX(1,J)=0
C  GET POSITION 7 AND PUT IN NDX
         DO 324 I=1,SIZE-1
         DO 323 J=1,PSIZE
323      NDX(I,J)=AX(I+1,J)
324      CONTINUE
         DO 342 J=1,PSIZE
342      NDX(SIZE,J)=0
C  SET UP BX (NEIGHBORHOOD B)
         DO 325 I=1,SIZE,2
         DO 326 J=1,PSIZE
         BX(I,J)=NFX(I,J)
326      BX(I+1,J)=NAX(I+1,J)
325      CONTINUE
C  SET UP CX (NEIGHBORHOOD C)
         DO 333 I=1,SIZE,2
         DO 327 J=1,PSIZE
         CX(I,J)=NAX(I,J)
327      CX(I+1,J)=NBX(I+1,J)
333      CONTINUE
C  SET UP EX (NEIGHBORHOOD E)
         DO 328 I=1,SIZE,2
         DO 329 J=1,PSIZE
         EX(I,J)=NEX(I,J)
329      EX(I+1,J)=NCX(I+1,J)
328      CONTINUE
C  SET UP FX (NEIGHBORHOOD F)
         DO 334 I=1,SIZE,2
         DO 335 J=1,PSIZE
         FX(I,J)=NDX(I,J)
335      FX(I+1,J)=NEX(I+1,J)
334      CONTINUE
C  GET COMPLEMENT OF AX,BX,CX,DX,EX,FX
```

TABLE B.1 (Continued)

```
650      DO 651 I=1,SIZE
         DO 652 J=1,PSIZE
         CALL NEG(AX(I,J),NAX(I,J))
         CALL NEG(BX(I,J),NBX(I,J))
         CALL NEG(CX(I,J),NCX(I,J))
         CALL NEG(DX(I,J),NDX(I,J))
         CALL NEG(EX(I,J),NEX(I,J))
652      CALL NEG(FX(I,J),NFX(I,J))
651      CONTINUE
         INDEX=FLD3+1
         DO 721 I=1,SIZE
         DO 722 J=1,PSIZE
C
C  THE FOLLOWIING BLOCKS OF STATEMENTS RECOGNIZE
C  THE INDICATED INDEX OF THE SURROUND OF MODULE
C  A(I,J).
C
         GO TO (700,701,702,703,704,705,706,707,708,709,710,711,712,
     C  713), INDEX
C  INDEX 0
700      CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C  NFX(I,J),M(I,J))
         CALL ITORI(IND(I,J),M(I,J))
         GO TO 722
C  INDEX 1
701      CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C  NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C  NFX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C  NFX(I,J),M(1,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C  NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C  NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C  FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 2
702      CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C  NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C  NFX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C  NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C  NFX(I,J),M(1,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C  FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C  FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
```

TABLE B.1 (Continued)

```
C  INDEX 3
703      CALL RECOG(AX(I,J),BX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C   NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C   NFX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(1,J))
         CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 4
704      CALL RECOG(AX(I,J),BX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C   NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   NFX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),BX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 5
705      CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),BX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),BX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(1,J))
         CALL RECOG(AX(I,J),BX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
```

TABLE B.1 (Continued)

```
C  INDEX 6
706      CALL RECOG(AX(I,J),BX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(IND(I,J),M(I,J))
         GO TO 722
C  INDEX 7
707      CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 8
708      CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C   NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   NFX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(1,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 9
709      CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C   FX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
```

TABLE B.1 (Continued)

```
C  INDEX 10
710      CALL RECOG(AX(I,J),BX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),DX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 11
711      CALL RECOG(NAX(I,J),NBX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C   NFX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),NDX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 12
712      CALL RECOG(AX(I,J),NBX(I,J),NCX(I,J),DX(I,J),NEX(I,J),
     C   NFX(I,J),M(I,J))
         CALL RECOG(NAX(I,J),BX(I,J),NCX(I,J),NDX(I,J),EX(I,J),
     C   NFX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(NAX(I,J),NBX(I,J),CX(I,J),NDX(I,J),NEX(I,J),
     C   FX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
         GO TO 722
C  INDEX 13
713      CALL RECOG(NAX(I,J),BX(I,J),CX(I,J),NDX(I,J),EX(I,J),
     C   FX(I,J),M(I,J))
         CALL RECOG(AX(I,J),NBX(I,J),CX(I,J),DX(I,J),NEX(I,J),
     C   FX(I,J),N(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL RECOG(AX(I,J),BX(I,J),NCX(I,J),DX(I,J),EX(I,J),
     C   NFX(I,J),M(I,J))
         CALL ITORI(N(I,J),M(I,J))
         CALL ITORI(IND(I,J),N(I,J))
```

TABLE B.1 (Continued)

```
722     CONTINUE
721     CONTINUE
        GO TO 2
C  END OF INDEX INSTRUCTION.
C
C
C
C  TBZ, TBNZ, TLBZ, TLBNZ INSTRUCTIONS
C
400     ADDR=MDR2
        INSTRU2=INSTRU-22
        GO TO (401, 402, 401, 402), INSTRU2
401     IF (RF .EQ. 0) CC=ADDR
        GO TO 2
402     IF (RF .NE. 0) CC=ADDR
        GO TO 2
C
C  TIN INSTRUCTION
C
C  READ IN AN ARRAY INTO A FROM A FILE
C  READ IN A COMMAND STRING TO IDENTIFY INPUT AND
C  OUTPUT FILES.
C
500     CALL PRINT ('ENTER COMMAND STRING')
        CALL ICSI(FILSPC, DEFEXT, , , 0)
        IF (IFETCH(FILSPC(16))+IFETCH(FILSPC(1)).NE.0) STOP 'FETCH'
        IF (IENTER(1, FILSPC(1), 40).NE. 40) STOP 'ENTER'
        IF (LOOKUP(2, FILSPC(16)).NE. 40) STOP 'LOOKUP'
        IF (IREADW(2048, LP1, 0, 2).NE. 2048) STOP 'READ'
C  SET UP FILENAME TO BE DISPLAYED ON THE SCREEN.
        LP1(4)="123
        LP1(9)="40
        LP1(10)="40
        LP1(11)="40
        LP1(12)="124
        LP1(13)="123
        LP1(14)="105
        LP1(15)="115
        IF (IWRITW(2048, LP1, 0, 1).NE. 2048) STOP 'WRITE'
        IF (IREADW(8192, LP1, 8, 2).NE. 8192) STOP 'READ2'
C  SET UP ARRAY A FROM INPUT FILE.
        DO 580 I=1, SIZE
        DO 581 K=1, PSIZE
        WORK=0
        DO 582 J=1, 64, 4
        WORK=WORK*2
        IF (LPIC(J+(K-1)*64, I*4-3) .NE. "177) WORK=WORK+1
582     CONTINUE
581     A(I, K)=WORK
580     CONTINUE
        GO TO 2
C
C  TOUT INSTRUCTION
C
C  WRITE OUT ARRAY O INTO A FILE
600     DO 601 I=1, SIZE
        DO 602 J=1, PSIZE
        O(I, J)=M(I, J)
602     CONTINUE
601     CONTINUE
        WRITE (7, 6) ((O(I, J), J=1, PSIZE), I=1, SIZE)
6       FORMAT (1X, 4I6)
```

TABLE B.1 (Continued)

```
C  INSERT THE SETUP FOR LP1 TO OUTPUT TO A FILE
        DO 560 I=1,32
        DO 561 J=1,2
        WORK=M(I,J)
        IF ((I .GT. SIZE) .OR. (J .GT. PSIZE)) WORK=0
        DO 564 L=1,16
        DO 562 K=I*4-3,I*4
        DO 563 KK=(J-1)*64+L*4-3,(J-1)*64+L*4
        LPIC(KK,K)="177
        IF (WORK .LT. 0) LPIC(KK,K)="200
563     CONTINUE
562     CONTINUE
564     WORK=WORK*2
561     CONTINUE
560     CONTINUE
        IF (IWRITW(8192,LP1,8,1).NE.8192) STOP 'WRITE2'
        CALL CLOSEC(1)
        CALL CLOSEC(2)
        GO TO 2
C
C  ENDD INSTRUCTION
C
800     STOP
        END
C
C  THIS SUBROUTINE CHECKS ARRAY Q FOR ALL ZEROS.  IF Q=0,  RF=0.
C  OTHERWISE RF=1
C
        SUBROUTINE RFSUB(Q,SIZE,PSIZE,RF)
        IMPLICIT INTEGER (A-Z)
        DIMENSION Q(32,2)
        RF=0
        DO 40 I=1,SIZE
        DO 41 J=1,PSIZE
        IF (Q(I,J) .NE. 0) RF=1
41      CONTINUE
40      CONTINUE
        RETURN
        END
C
C  THIS SUBROUTINE DEFINES THE TSE MASKS.
C
        SUBROUTINE DEFINE(SIZE,PSIZE,MASK,ROW1,ROW2)
        IMPLICIT INTEGER (A-Z)
        DIMENSION MASK(32,2)
        DO 34 I=1,SIZE,2
        SAV1=ROW1
        SAV2=ROW2
        DO 35 J=1,PSIZE
        MASK(I,J)=SAV1
        MASK(I+1,J)=SAV2
35      CALL ROT(SAV1,SAV2)
34      CONTINUE
        RETURN
        END
```

APPENDIX C

SECONDARY PROGRAM

Table C.1 is a source listing of the subroutine program for the interpreter. These subroutines are called from the main program to perform special operations which can best be executed using Macro.

TABLE C.1

SECONDARY PROGRAM

```
;
;   SECONDARAY PROGRAM
;
;   INTERPRETOR SUBROUTINES
;
;
;   THIS SUBROUTINE IS USED TO READ TWO WORDS FROM THE
;   TSE MEMORY INTO THE SPECIFIED OPERANDS.
;
        .TITLE READ
        .MCALL ..V2...,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL MEMORY
        .GLOBL READ
READ:   MOV     #MEMORY,R2      ;VALUE OF ADDR OF MEMORY TO R2
        ADD     @2(R5),R2       ;ADD 1ST OP (MAR) TO R2
        MOV     (R2),@4(R5)     ;MOVE PART OF RESULT TO MDR1
        ADD     #2,R2           ;INCREMENT R2
        MOV     (R2),@6(R5)     ;MOVE REMAINDER OF RESULT TO MDR2
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO SEPARATE AND MANIPULATE
;   THE FOUR FIELDS OF THE TWO WORDS READ FROM THE TSE
;   MEMORY.
;
        .TITLE DCODE
        .MCALL ..V2...,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL DCODE
DCODE:  MOV     #^B1111111100000000,R4  ;R4 CLEARS HIGH BYTE
        MOVB    @2(R5),R2       ;MOVE FIELD TO R2
        BIC     R4,R2           ;CLEAR HIGH BYTE OF R2
        MOV     R2,@6(R5)       ;PUT OPCODE IN INSTRU
        SWAB    @2(R5)          ;SWAP THE TWO FIELDS
        MOVB    @2(R5),R2       ;MOVE FIELD TO R2
        BIC     R4,R2           ;CLEAR HIGH BYTE OF R2
        MOV     R2,@10(R5)      ;PUT FLD1 IN OPERAND
        MOVB    @4(R5),R2       ;MOVE FIELD TO R2
        BIC     R4,R2           ;CLEAR HIGH BYTE OF R2
        MOV     R2,@12(R5)      ;PUT FLD2 IN OPERAND
        SWAB    @4(R5)          ;SWAP BYTES OF MDR2
        MOVB    @4(R5),R2       ;MOVE FIELD TO R2
        BIC     R4,R2           ;CLEAR HIGH BYTE OF R2
        MOV     R2,@14(R5)      ;  PUT FLD3 IN OPERAND
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE DEFINES MACROS WHICH SET UP THE TWO-WORD
;   ENCODED FORM OF THE TSE INSTRUCTIONS.   IT ALSO CONTAINS
;   THE MEMORY FOR THE TSE PROGRAM.
;
;   IND IS LATCH I OF THE TSE COMPUTER, MM IS MASK M OF THE
;   TSE COMPUTER, AND ENDD IS THE DELIMITER FOR THE END
;   OF THE TSE COMPUTER MEMORY.
;
```

TABLE C.1 (Continued)

```
.TITLE TSE MEMORY
.MCALL  ..V2..,.REGDEF
..V2..
.REGDEF
.GLOBL  MEMORY
.RADIX  10
A=1
B=2
C=3
O=4
IND=5
MO=1
MM=2
M1=3
M2=4
M3=5
M12=6
M13=7
M23=8
.RADIX
.MACRO  TMOV    REG1,REG2
.NLIST  SRC
.BYTE   ^D1,REG1,REG2,0
.LIST   SRC
.ENDM
.MACRO  TMVI    REG,MASK
.NLIST  SRC
.BYTE   ^D2,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TCLRI
.NLIST  SRC
.BYTE   ^D3,0,0,0
.LIST   SRC
.ENDM
.MACRO  TAND    REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D4,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TANA    REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D5,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TANI    REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D6,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TANDN   REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D7,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TANAN   REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D8,REG1,REG2,MASK
.LIST   SRC
```

TABLE C.1 (Continued)

```
.ENDM
.MACRO  TOR     REG,MASK
.NLIST  SRC
.BYTE   ^D9,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TORA    REG,MASK
.NLIST  SRC
.BYTE   ^D10,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TORI    REG,MASK
.NLIST  SRC
.BYTE   ^D11,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TORN    REG,MASK
.NLIST  SRC
.BYTE   ^D12,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TORAN   REG,MASK
.NLIST  SRC
.BYTE   ^D13,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TXOR    REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D14,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TXRA    REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D15,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TXRI    REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D16,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TCMR    REG1,REG2,MASK
.NLIST  SRC
.BYTE   ^D17,REG1,REG2,MASK
.LIST   SRC
.ENDM
.MACRO  TIDA    X
.NLIST  SRC
.BYTE   ^D18,0,0,X
.LIST   SRC
.ENDM
.MACRO  TCNT    REG,MASK
.NLIST  SRC
.BYTE   ^D19,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TTEST   REG,MASK
.NLIST  SRC
```

TABLE C.1 (Continued)

```
.BYTE   ^D20,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TCMP    REG,MASK
.NLIST  SRC
.BYTE   ^D21,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TCPI    REG,MASK
.NLIST  SRC
.BYTE   ^D22,REG,0,MASK
.LIST   SRC
.ENDM
.MACRO  TBZ     ADDR
.NLIST  SRC
.BYTE   ^D23,0,0,<ADDR-MEMORY>
.LIST   SRC
.ENDM
.MACRO  TBNZ    ADDR
.NLIST  SRC
.BYTE   ^D24,0,0,<ADDR-MEMORY>
.LIST   SRC
.ENDM
.MACRO  TLBZ    ADDR
.NLIST  SRC
.BYTE   ^D25,0
.WORD   <ADDR-MEMORY>
.LIST   SRC
.ENDM
.MACRO  TLBNZ   ADDR
.NLIST  SRC
.BYTE   ^D26,0
.WORD   <ADDR-MEMORY>
.LIST   SRC
.ENDM
.MACRO  TIN
.NLIST  SRC
.BYTE   ^D27,0,0,0
.LIST   SRC
.ENDM
.MACRO  TOUT    REG
.NLIST  SRC
.BYTE   ^D28,REG,0,0
.LIST   SRC
.ENDM
.MACRO  ENDD
.NLIST SRC
.BYTE   ^D29,0,0,0
.LIST   SRC
.ENDM
.LIST   MEB
.EVEN
```

TABLE C.1 (Continued)

```
;
;   THIS IS THE TSE MEMORY.
;
MEMORY: TIN
ATHER:  TMOV    C,A
        TCLRI
        TIDA    1
        TIDA    2
        TIDA    3
        TANDN   A,IND,M1
        TCLRI
        TIDA    1
        TIDA    2
        TIDA    3
        TANDN   A,IND,M2
        TCLRI
        TIDA    1
        TIDA    2
        TIDA    3
        TANDN   A,IND,M3
        TCMP    C,MM
        TBNZ    ATHER
        TOUT    A
        ENDD
;
;   THIS SUROUTINE IS USED TO PERFORM THE TMOV AND
;   TMVI TSE INSTRUCTIONS.
;
        .TITLE TMVI
        .MCALL ..V2...REGDEF
        ..V2..
        .REGDEF
        .GLOBL ITMVI
ITMVI:  MOV     @4(R5),@2(R5)   ;MOVE 2ND OP TO 1ST OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TAND AND
;   TANA TSE INSTRUCTIONS.
;
        .TITLE TAND
        .MCALL  ..V2...REGDEF
        ..V2..
        .REGDEF
        .GLOBL ITAND
ITAND:  MOV     @10(R5),R1      ;MOVE MASK TO R1
        COM     R1              ;COMPLEMENT MASK
        BIS     @6(R5),R1       ;MASK OR 3RD OP
        MOV     @2(R5),R2       ;MOV 1ST OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,R1           ;MASK OR 3RD OP AND A
        MOV     R1,@4(R5)       ;MOVE RESULT TO 2ND OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TANI TSE INSTRUCTION.
;
        .TITLE  TANI
        .MCALL  ..V2...REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITANI
```

TABLE C.1 (Continued)

```
ITANI:  MOV     @6(R5),R1       ;MOVE MASK TO R1
        MOV     @4(R5),R2       ;MOVE 2ND OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,R1           ;MASK AND 2ND OP
        MOV     R1,@2(R5)       ;MOVE RESULT TO 1ST OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUROUTINE IS USED TO PERFORM THE TANDN AND
;   TANAN TSE INSTRUCTIONS.
;
        .TITLE  TANDN
        .MCALL  ..V2...,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITANDN
ITANDN: MOV     @10(R5),R1      ;MOVE MASK TO R1
        MOV     @6(R5),R2       ;MOVE 3RD OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,R1           ;MASK AND 3RD OP
        COM     R1              ;COMPLEMENT RESULT
        MOV     @2(R5),R2       ;MOVE 1ST OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,R1           ;RESULT AND 1ST OP
        MOV     R1,@4(R5)       ;MOVE THIS RESULT TO 2ND OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TOR AND
;   TORA TSE INSTRUCTIONS.
;
        .TITLE  TOR
        .MCALL  ..V2...,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITOR
ITOR:   MOV     @6(R5),R1       ;MOVE MASK TO R1
        MOV     @4(R5),R2       ;MOVE 2ND OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,R1           ;MASK AND 2ND OP
        BEQ     SKIP            ;SET RF IF RESULT DOES
        MOV     #^D1,@10(R5)    ;NOT EQUAL ZERO.
SKIP:   BIS     R1,@2(R5)       ;RESULT OR 1ST OP TO 1ST OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TORI TSE INSTRUCTION.
;
        .TITLE  TORI
        .MCALL  ..V2...,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITORI
ITORI:  MOV     @4(R5),R1       ;MOVE MASK TO R1
        BIS     R1,@2(R5)       ;MASK OR REG TO REG
        RTS     PC              ;RETURN CONTROL
```

TABLE C.1 (Continued)

```
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TORN AND
;   TORAN TSE INSTRUCTIONS.
;
        .TITLE  TORN
        .MCALL  ..V2.,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITORN
ITORN:  MOV     @6(R5),R1       ;MOVE MASK TO R1
        COM     R1              ;COMPLEMENT MASK
        BIS     @4(R5),R1       ;RESULT OR 2ND OP
        COM     R1              ;COMPLEMENT THIS RESULT
        BEQ     SKIP2           ;SET RF IF RESULT DOES
        MOV     #^D1,@10(R5)    ;NOT EQUAL ZERO.
SKIP2:  BIS     R1,@2(R5)       ;THIS RESULT OR 1ST OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TXOR AND
;   TXRA TSE INSTRUCTIONS.
;
        .TITLE  TXOR
        .MCALL  ..V2.,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITXOR
ITXOR:  MOV     @10(R5),R1      ;MOVE MASK TO R1
        MOV     @6(R5),R2       ;MOVE 3RD OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,R1           ;MASK AND 3RD OP
        MOV     @2(R5),R2       ;MOVE 1ST OP TO R2
        XOR     R1,R2           ;PREVIOUS RESULT EXOR 1ST OP
        MOV     R2,@4(R5)       ;MOVE THIS RESULT TO 2ND OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TXRI AND
;   TCMR TSE INSTRUCTIONS.
;
        .TITLE  TXRI
        .MCALL  ..V2.,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITXRI
ITXRI:  MOV     @6(R5),R1       ;MOVE MASK TO R1
        MOV     @4(R5),R2       ;MOVE 2ND UP TO R2
        XOR     R1,R2           ;MASK EXOR 2ND OP
        MOV     R2,@2(R5)       ;MOVE RESULT TO 1ST OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TCNT TSE INSTRUCTION.
;
        .TITLE  TCNT
        .MCALL  ..V2.,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITCNT
ITCNT:  MOV     @2(R5),@4(R5)   ;PUT REG IN N
        MOV     @6(R5),R2       ;MOVE 3RD OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,@4(R5)       ;AND MASK TO N
        RTS     PC              ;RETURN CONTROL
```

TABLE C.1 (Continued)

```
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TTEST TSE INSTRUCTION.
;
        .TITLE  TTEST
        .MCALL  ..V2..,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITTEST
ITTEST: MOV     @6(R5),@4(R5)   ;MOVE MASK TO N
        COM     @4(R5)          ;COMPLEMENT MASK
        BIS     @2(R5),@4(R5)   ;MASK OR REG
        COM     @4(R5)          ;COMPLEMENT RESULT
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TCMP TSE INSTRUCTION.
;
        .TITLE  TCMP
        .MCALL  ..V2..,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITCMP
ITCMP:  MOV     @2(R5),@4(R5)   ;MOVE REG TO N
        MOV     @6(R5),R2       ;MOVE 3RD OP TO R2
        COM     R2              ;COMPLEMENT R2
        BIC     R2,@4(R5)       ;MASK AND REG
        MOV     @10(R5),R2      ;MOVE A TO R2
        MOV     @6(R5),R3       ;MOVE 3RD OP TO R3
        COM     R3              ;COMPLEMENT R3
        BIC     R3,R2           ;MASK AND A
        XOR     R2,@4(R5)       ;RESULT EXOR RESULT
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO PERFORM THE TCPI TSE INSTRUCTION.
;
        .TITLE  TCPI
        .MCALL  ..V2..,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  ITCPI
ITCPI:  MOV     @2(R5),R2       ;MOVE REG TO R2
        MOV     @6(R5),@4(R5)   ;MOVE MASK TO N
        XOR     R2,@4(R5)       ;MASK EXOR REG
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUROUTINE IS USED TO DEFINE TSE MASK M.
;
        .TITLE  DEFMM
        .MCALL  ..V2..,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  DEFMM
DEFMM:  MOV     #^B1111111111111111,@2(R5)      ;MOVE BIT PATTERN TO OP
        RTS     PC              ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO DEFINE TSE MASK M1.
;
        .TITLE  DEFM1
        .MCALL  ..V2..,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  DEFM1
```

TABLE C.1 (Continued)

```
DEFM1:  MOV     #^B1001001001001001,@2(R5)      ;MOVE BIT PATTERN TO OP
        MOV     #^B0100100100100100,@4(R5)      ;MOVE BIT PATTERN TO OP
        RTS     PC                  ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO DEFINE TSE MASK M2.
;
        .TITLE  DEFM2
        .MCALL  ..V2...,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  DEFM2
DEFM2:  MOV     #^B0100100100100100,@2(R5)      ;MOVE BIT PATTERN TO OP
        MOV     #^B0010010010010010,@4(R5)      ;MOVE BIT PATTERN TO OP
        RTS     PC                  ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO DEFINE TSE MASK M3.
;
        .TITLE  DEFM3
        .MCALL  ..V2...,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  DEFM3
DEFM3:  MOV     #^B0010010010010010,@2(R5)      ;MOVE BIT PATTERN TO OP
        MOV     #^B1001001001001001,@4(R5)      ;MOVE BIT PATTERN TO OP
        RTS     PC                  ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO DEFINE TSE MASK M12.
;
        .TITLE  DEFM12
        .MCALL  ..V2...,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  DEFM12
DEFM12: MOV     #^B1101101101101101,@2(R5)      ;MOVE BIT PATTERN TO OP
        MOV     #^B0110110110110110,@4(R5)      ;MOVE BIT PATTERN TO OP
        RTS     PC                  ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO DEFINE TSE MASK M13.
;
        .TITLE  DEFM13
        .MCALL  ..V2...,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  DEFM13
DEFM13: MOV     #^B1011011011011011,@2(R5)      ;MOVE BIT PATTERN TO OP
        MOV     #^B1101101101101101,@4(R5)      ;MOVE BIT PATTERN TO OP
        RTS     PC                  ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO DEFINE TSE MASK M23.
;
        .TITLE  DEFM23
        .MCALL  ..V2...,..REGDEF
        ..V2..
        .REGDEF
        .GLOBL  DEFM23
```

TABLE C.1 (Continued)

```
DEFM23: MOV     #^B0110110110110110,@2(R5)      ;MOVE BIT PATTERN TO OP
        MOV     #^B1011011011011011,@4(R5)      ;MOVE BIT PATTERN TO OP
        RTS     PC                  ;RETURN CONTROL
;
;  THIS SUROUTINE IS USED TO ROTATE A WORD LEFT TO
;  HELP SET UP THE TSE MASKS.
;
        .TITLE ROTATE
        .MCALL ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL ROT
ROT:    ASL     @2(R5)              ;SHIFT SAV1 LEFT
        BMI     SET1                ;BRANCH IF RESULT IS MINUS
        BR      NEXT                ;OTHERWISE SET UP SAV2
SET1:   ADD     #1,@2(R5)           ;SET LOW BIT OF SAV1
NEXT:   ASL     @4(R5)              ;SHIFT SAV2 LEFT
        BMI     SET2                ;BRANCH IF RESULT IS MINUS
        BR      LAST                ;OTHERWISE RETURN
SET2:   ADD     #1,@4(R5)           ;SET LOW BIT OF SAV2
LAST:   RTS     PC                  ;RETURN CONTROL
;
;  THIS SUBROUTINE IS USED TO DESIGNATE THE POSITIONS WHICH
;  HAVE SURROUNDS OF THE INDEX BEING CHECKED FOR.
;
        .TITLE  RECOG
        .MCALL ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL RECOG
RECOG:  MOV     @2(R5),R2           ;MOVE 1ST OP TO R2
        MOV     @4(R5),R3           ;MOVE 2ND OP TO R3
        COM     R3                  ;COMPLEMENT 2ND OP
        BIC     R3,R2               ;AND 1ST OP TO 2ND OP
        MOV     @6(R5),R3           ;MOVE 3RD OP TO R3
        COM     R3                  ;COMPLEMENT 3RD OP
        BIC     R3,R2               ;AND 2ND OP TO 3RD OP
        MOV     @10(R5),R3          ;MOVE 4TH OP TO R3
        COM     R3                  ;COMPLEMENT 4TH OP
        BIC     R3,R2               ;AND 3RD OP TO 4TH OP
        MOV     @12(R5),R3          ;MOVE 5TH OP TO R3
        COM     R3                  ;COMPLEMENT 5TH OP
        BIC     R3,R2               ;AND 4TH OP TO 5TH OP
        MOV     @14(R5),R3          ;MOVE 6TH OP TO R3
        COM     R3                  ;COMPLEMENT 6TH OP
        BIC     R3,R2               ;AND 5TH OP TO 6TH OP
        MOV     R2,@16(R5)          ;MOV RESULT TO 7TH OP
        RTS     PC                  ;RETURN CONTROL
;
;  THIS SUBROUTINE IS USED TO SHIFT A WORD OF DATA RIGHT ONE
;  PLACE.  IT IS USED TO HELP SET UP THE NEIGHBOR PLANES.
;
        .TITLE  RSHFT
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  RSHFT
```

TABLE C.1 (Continued)

```
RSHFT:  MOV     @2(R5),R2           ;MOVE 1ST OP TO R2
        ASR     R2                  ;SHIFT R2 RIGHT
        BCC     CLEAR               ;BRANCH IF NEW CARRY IS CLEAR
        MOV     #1,R3               ;OTHERWISE SET NEW CARRY
        BR      CONT                ;BRANCH TO CONT
CLEAR:  MOV     #0,R3               ;CLEAR NEW CARRY
CONT:   CMP     #0,@6(R5)           ;SEE IF PREVIOUS CARRY IS 0
        BNE     CLEAR2              ;BRANCH IF PREVIOUS CARRY NE 0
        BIC     #^B1000000000000000,R2  ;CLEAR 1ST BIT OF R2
        BR      GOON                ;BRANCH TO GOON
CLEAR2: BIS     #^B1000000000000000,R2  ;SET 1ST BIT OF R2
GOON:   MOV     R3,@6(R5)           ;PUT NEW CARRY IN 3RD OP
        MOV     R2,@4(R5)           ;PUT RESULT IN 2ND OP
        RTS     PC                  ;RETURN CONTROL
;
;   THIS SUBROUTINE IS USED TO COMPLEMENT A WORD OF DATA.
;
        .TITLE NEG
        .MCALL  ..V2..,.REGDEF
        ..V2..
        .REGDEF
        .GLOBL  NEG
NEG:    MOV     @2(R5),R2           ;MOVE 1ST OP TO R2
        COM     R2                  ;COMPLEMENT FIRST OP
        MOV     R2,@4(R5)           ;MOVE RESULT TO 2ND OP
        RTS     PC                  ;RETURN CONTROL
        .END READ
```

VITA

Dianne Elizabeth Horner McCollough was born on December 19, 1954 in Knoxville, Tennessee. She attended elementary schools in Clinton, Tennessee for three years. The remainder of her school years were spent in Oak Ridge, Tennessee where she attended elementary school, junior high, and was graduated from Oak Ridge High School in June 1972. Immediately following graduation she began work on a Bachelor of Science degree in Mathematics at Tennessee Technological University and received that degree in August 1975. In September of 1975 she accepted a non-service fellowship for one year at The University of Tennessee and began work toward the Master of Science degree in Computer Science. This degree was completed in August 1977.

During her study she accepted a position at Oak Ridge National Laboratory as a Computing Analyst.

She is a member of Alpha Psi Upsilon, Kappa Mu Epsilon, and Phi Kappa Phi honor societies.

Married to Norman Darnell McCollough, Jr., she is living in Oak Ridge, Tennessee.