A cellular computer to implement the Kalman filter algorithm
by Lynn Elliot Cannon

A thesis submitted to the Graduate Faculty in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY in Electrical Engineering
Montana State University

Abstract:
The subject of this thesis is the development of the design for a specially-organized, general-purpose computer which performs matrix operations efficiently.

The content of the thesis is summarized as follows: First, a review of the relevant work which has been done with microcellular and macrocellular techniques is made, Second, the discrete Kalman filter is described as an example of the type of problem for which this computer is efficient. Third, a detailed design for a cellular, array-structured computer is presented. Fourth, a computer program which simulates the cellular computer is described. Fifth, the recommendation is made that one cell and the associated control circuits be constructed to determine the feasibility of producing a hardware realization of the entire computer.

A CELLULAR COMPUTER TO IMPLEMENT THE KALMAN FILTER ALGORITHM

by

LYNN ELLIOT CANNON

A thesis submitted to the Graduate Faculty in partial fulfillment of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Head, Major Department

Chairman, Examining Committee

Graduate Dean

MONTANA STATE UNIVERSITY
Bozeman, Montana

August, 1969

ACKNOWLEDGMENT

The author wishes to thank Professor R. C. Minnick for his encouragement and helpful suggestions during the course of this graduate work and thesis research.

The financial support of the graduate work provided by the Electrical Engineering Department and by a National Science Foundation Traineeship is gratefully acknowledged, as is the thesis support furnished by the Office of Naval Research Contract No. N00014-67-C-0477.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The subject of this thesis is the development of the design for a specially-organized, general-purpose computer which performs matrix operations efficiently.

The content of the thesis is summarized as follows: First, a review of the relevant work which has been done with microcellular and macrocellular techniques is made. Second, the discrete Kalman filter is described as an example of the type of problem for which this computer is efficient. Third, a detailed design for a cellular, array-structured computer is presented. Fourth, a computer program which simulates the cellular computer is described. Fifth, the recommendation is made that one cell and the associated control circuits be constructed to determine the feasibility of producing a hardware realization of the entire computer.

Chapter 1

INTRODUCTION AND REVIEW OF CELLULAR RESEARCH

1.1 Introduction

Designers of computers are often confronted with the development of a computer for a particular application such as vehicle navigation, signal processing or the like. Many of these applications require that computations be performed as rapidly as possible and that the computer be as small as possible. In order to meet the requirements for speed, cost and physical size, the designer will sometimes resort to the development of a special purpose computer. The disadvantages of this approach are the large initial design costs and the possible necessity of a complete redesign to account for a change in the computation algorithm. On the other hand, general purpose computers, while they do not have to be redesigned to account for algorithm changes, do have disadvantages. Those general purpose computers which are fast enough for some applications are usually not small enough; and those which are small enough are usually not fast enough or are too expensive. These arguments suggest that some thought should be given to the design of general purpose computers which are capable of performing certain types of operations efficiently. Although efficiency may be gained by using faster components, this method is usually expensive and leads to other complications, such as poor reliability. Another method of obtaining greater efficiency for certain operations is to build a computer which takes advantage of the structure inherent in these operations. This type of machine can be called a specially organized, general purpose computer. A computer of this type could be highly efficient for some

types of operations as is a special purpose computer, but would be easy to adapt to changes in computation algorithms as is a general purpose computer.

One class of problems whose inherent structure may be incorporated into the design of a computer is that class which deals with operations on matrices and vectors. An example of problems of this type is the discrete Kalman filter.[3] For certain applications of the Kalman filter many operations must be performed on matrices as rapidly as possible. Thus, the basis for this research is to develop a design for a small specially organized, general purpose computer which is particularly efficient for performing matrix operations.

Incorporating matrix operations into the structure of a computer suggests the use of arrays of logic elements. Therefore, it is natural to consider some of the work which has been done with arrays of such elements.

Arrays of similar or identical logic circuits together with some interconnection structure constitute cellular arrays. If the cells have a low complexity, the arrays of these cells are referred to as microcellular. If not, they are referred to as macrocellular. Low-complexity cells contain no more than a few gates. In either case, the logic circuits, or cells as they are called, are usually arranged in some rectangular fashion and the interconnections between the cells usually forms a uniform pattern.

In the next section some of the work which has been done on micro-

cellular arrays will be discussed, and in the following section some of the pertinent work on macrocellular arrays will be considered.

1.2 Survey of Microcellular Work

One of the earliest arrays which used identical cells and a uniform interconnection was originated in 1961 by Brooking[4,8] at the Air Force Cambridge Research Laboratories (AFCRL). The cells in this array are eight-input NOR gates with each input of a cell being connected to the output of one of the eight neighboring cells in the array. The array is known as an eight-neighbor array. Each cell in the array has inputs from outside the array which are used to provide an electrical disconnect for any of the eight inputs. Several variations of the eight-neighbor arrays were produced at AFCRL by Giusti.

Methods for the logical design of eight-neighbor cellular arrays of NAND cells were developed by Spandorfer and Murphy[21] in 1963. These workers also developed methods of avoiding faulty cells in an array.

Much work has been done on arrays whose cells may perform any one of several different functions. In 1962, Maitra[11] considered one-dimensional arrays in which each cell can be any of the 16 functions of two variables. Such an array is called a Maitra cascade. Additional work on Maitra cascades eventually led to Minnick's cutpoint array[16] in 1964. This array contains cells which can produce any of eight combinational switching functions or an S-R flip-flop. Four parameters

specify the function to be produced for each cell. In a later array by Minnick[15], called the cobweb array, cell parameters specify the interconnection as well as the cell function. Techniques have been developed for the logical design of both cutpoint and cobweb arrays[14-16].

For a much more complete survey of the development of microcellular work the reader is referred to reference (10).

1.3 Survey of Macrocellular Work

Arrays of large or complex cells are known as macrocellular arrays. Most computers which have been designed with an array structure use some form of a macrocellular array. In this section, some of the work which has been done in the design of array structured computers is described.

1.3.1 Unger's Machine

In 1958 Unger[23] described a stored-program computer which was specially organized for pattern detection. The computer consists of a master control and a rectangular array of logical modules as shown in Figure 1.1. The master control contains a clock, decoding circuits and a random access memory. It accesses instructions from memory, decodes them and issues commands which go simultaneously to all of the modules.

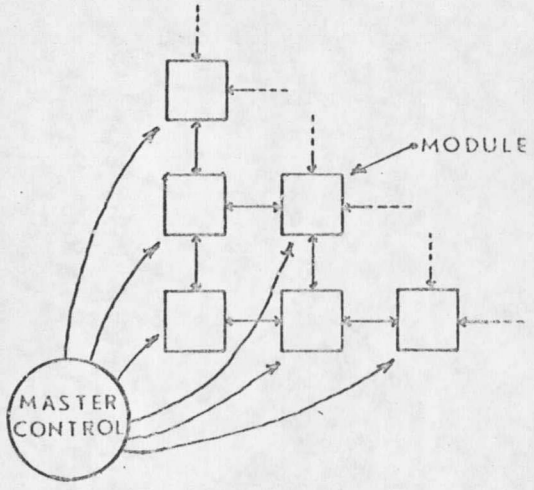


Figure 1.1. Unger's Machine

Each module (or macro-cell in the nomenclature of Section 1.1) contains a one-bit accumulator, six one-bit words of random access memory and associated logic. Each has inputs from the master control and from the accumulators of its four immediate neighbors. Each accumulator also has an input from outside the machine.

A logical OR circuit with inputs from each of the accumulators informs the master control if all of the accumulators contain zeros. This information is used for a transfer-on-zero command which is the only decision-dependent command used.

Other commands for Unger's machine are shown in Table 1.1. These commands are described in detail in reference (23).

Table 1.1. Command Table for Unger's Machine

Command	Meaning
tr x	Transfer to instruction x.
trz x	Transfer on zero to instruction x.
in	Logical invert (NOT).
add	Logical add (OR).
mpy	Logical multiply (AND).
adm	Logical add to memory (OR).
mpm	Logical multiply to memory (AND).
st	Store.
wr	Write (actually a "LOAD" command).
sL(sR,sU,sD)	Shift left (right, up, down).
Add ref	Add reference.
Link	Link.
exp	Expand.
sA	Shift around.

Inputs to the array can be made in parallel by associating an input device such as a photocell with each module or by using the shift around command.

A typical program for Unger's machine might locate lower-left corners of a pattern. Lower-left corners are located by placing a 1 in all cells (here a cell refers to the accumulator of a module)

which satisfy the conditions: (a) there is a 1 in the cell, (b) there are 1's in the cells immediately above and immediately to the right of the cell, and (c) there are 0's in the cells immediately to the left, immediately below and diagonally adjacent to the lower left. The shaded cell of Figure 1.2 satisfies these conditions.

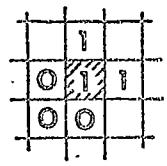


Figure 1.2. A Lower-Left Corner.

In order to illustrate Unger's machine, a program to find lower-left corners is given below. The first command of this program loads the original pattern into the a-register of all cells. In the following steps, the operands R and U refer to the a-register of adjacent cells to the right and above the cell in question, respectively. In Figure 1.3 an example set of data are shown in order to make the execution of the program more clear. From an examination of Figure 1.3(a) it is clear that cells (5,2) and (4,5) are the only two lower-left corners.

1) st a — Result in a-register is Figure 1.3(a).

2) mpm R,U,a — Result in a-register is Figure 1.3(b).

3) in — Result in accumulator is Figure 1.3(c).

4) mpy R,U — Result in accumulator is Figure 1.3(d).

5) sU — Result is not shown.

6) sR — Result in accumulator is Figure 1.3(e).

7) mpy a — Result in accumulator is Figure 1.3(f).

It should be noted that the result, shown in Figure 1.3(f), has 1's at the locations which are lower-left corners in the original pattern of Figure 1.3(a).

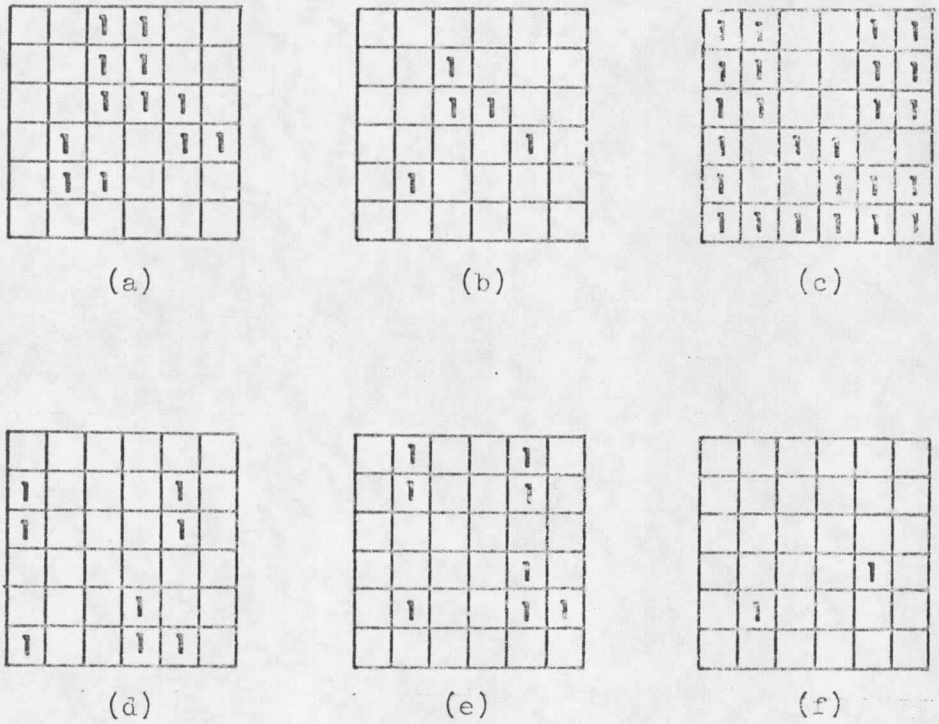


Figure 1.3. Lower-Left Corner Program Results.

Several programs have been written for pattern detection and recognition using the Unger machine. Some of these are described in reference (24).

1.3.2 Holland's Machine

In 1959 Holland[7] described a machine organization to provide a basis for investigations in computability and the theory of automata.

The machine is a two-dimensional array of modules. Each module contains a storage register, some associated logic and some auxiliary registers. At a given time a module may be active or inactive. If the module is active it treats the content of its storage register as an instruction and executes the instruction. After a module has executed its instruction it passes its active status on to its successor, which may be any of its four nearest neighbors in the array. Thus, sequences of instructions are arranged spatially throughout the array of modules, with an arbitrary number of sequences being executed at any given time.

Each cycle of the machine consists of three phases. In the first phase, module storage registers may be set to values provided by an external source. In the second phase, active modules determine the locations of their operands by causing paths to be gated open to them. In the third phase, the instructions in the storage registers of all active modules are executed.

In general, the Holland machine requires a large amount of hardware to accomplish reasonable computing tasks. It is extremely difficult to program the machine so that more than a very small percentage of the

modules are active at one time. In light of these problems a modified version of the Holland machine was described by Comfort[2] in 1963.

1.3.3 Comfort's Machine

Comfort's version had basically the same organization as the Holland machine. The machine described by Comfort is a fixed-size rectangular array of modules. At one side of the array are a set of arithmetic units called A-boxes. The array modules provide storage for data and instructions, communication with and between the A-boxes, and sequencing of instructions. The A-boxes do all arithmetic and logical computations. There is no central control unit; each module executes its own instruction when it is placed in the active state. The execution of a sequence of instructions causes the activation and deactivation of successive modules as in the Holland machine.

Comfort concluded that compared with Holland's machine, his organization results in:

1. Programmability improved by several orders of magnitude.
2. Reduction in amount of hardware by a factor of five.
3. Hardware utilization improved by a factor of three.
4. System performance degraded somewhat. (Because only one program sequence per A-box can be executed at one time.)

1.3.4 Gonzales' Machine

Another modified Holland machine was proposed by Gonzales[5]. It consists of three layers of modules with each layer being a

rectangular array similar to Holland's. A program layer stores data and instructions, a control layer decodes instructions and a computing layer performs arithmetical, logical and geometrical operations.

The programming is basically the same as the Holland machine with each instruction specifying the module containing the next instruction. While the computing layer is executing an instruction, the control and program layers can be working on the next two instructions, respectively.

As in the Holland machine, programming is difficult and hardware is not efficiently utilized.

1.3.5 The SOLOMON Computer

The SOLOMON (Simultaneous Operation Linked Ordinal Modular Network) is a parallel computer which was introduced by Slotnick, Borck and McReynolds[19] in 1962.

The SOLOMON consists basically of an array of processing elements and a central control as shown in Figure 1.4.

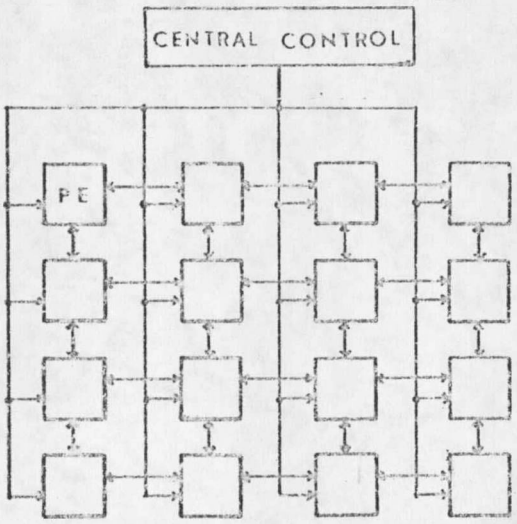


Figure 1.4. The SOLOMON Computer

Each of the processing elements has 4096 bits of core storage and the capabilities required to perform serial-by-bit arithmetic and logic. Words in the processing elements may be varied from one to 128 bits. A processing element may transfer data serially to and from its four nearest neighbors in the array.

A later version of the SOLOMON[17] had a faster clock rate, faster multiply logic and a fixed word length. The changes provided more computing capability for about the same cost.

1.3.6 The ILLIAC IV Computer

A direct descendent of the SOLOMON, the ILLIAC IV computer[1], consists of 256 processing elements arranged in four SOLOMON-type arrays of 64 processors each. Each processing element has 2048 words of 64-bit memory and extensive arithmetic capability. There is a common control unit which decodes instructions and generates control signals for all processing elements in the array.

The ILLIAC IV is supervised by a large general purpose computer which controls the execution of array programs, conducts input and output processes and performs independent data processing tasks.

The ILLIAC IV is currently under construction by the Burroughs Corporation. A large number of reports which describe various aspects of the ILLIAC IV have been issued by the University of Illinois where much of the design work is being done.

1.4 Relevance of Previous Work

The specially-organized computer to be described in this thesis draws both on the microcellular work and the macrocellular work which has been done.

Previous microcellular work is responsible for the concept of incorporating as much as possible of the structure of the problems, for which the computer is designed, into the array interconnection. The use of parameters to modify the interconnection structure is suggested by Minnick's cobweb array.

The contributions due to the macrocellular work are somewhat more obvious than those for the microcellular. Unger's machine suggests an array structure controlled by a central control unit. However, Unger's cells had limited arithmetic capability since his machine was intended for pattern detection. Holland introduced the idea of local control in the cells. While some local control seems advantageous, the use of purely local control has not proved to be very effective.

The basic structure of the proposed computer is similar to that of the SOLOMON with an array of processing cells under the control of a central control unit. Differences arise in the types of cells used, the interconnection structure for the array and the size of the system. The SOLOMON and its descendent the ILLIAC IV are large-scale systems. The designers of the ILLIAC IV, for example, are now considering the use of one of the largest computers currently available (the CDC 6600) for their central control unit. Although the ideas presented for

the proposed computer may be extended to much larger systems, the computer is intended to be roughly the size of some of the smallest computers now available.

1.5 Organization of Remaining Chapters

Chapter 2 discusses the Kalman filter. This discussion is presented to acquaint the reader with the types of operations which a specially organized computer should be able to perform efficiently. No new material is presented on the Kalman filter because it is intended to serve only as an example. Also included in this chapter are algorithms for the multiplication and inversion of matrices within an array-structured computer.

Chapter 3 presents a detailed design of a specially organized, general-purpose computer which has an array structure so that it is capable of performing matrix operations efficiently.

A computer program is given in Chapter 4 which provides a detailed simulation of the computer described in Chapter 3.

The concluding Chapter 5 gives a comparison of the computer described in Chapter 3 to more conventional computers.

Chapter 2

THE KALMAN FILTER

2.1 The Estimation Problem

A uniformly sampled discrete model of a linear dynamic system without control or noise inputs is given by

$$x(k+1) = \phi(k+1,k)x(k) \tag{2.1}$$

where $x(k)$ is a state vector representing the state of the system at time kT, T being the sampling period, $\phi(k+1,k)$ is the system transition matrix and $x(k+1)$ represents the state of the system at time $(k+1)T$.

Equation (2.1) represents the case of perfect predictability. The state $x(k+n)$ at time $(k+n)T$ may be determined exactly based only on the state vector $x(k)$ and the transition matrix $\phi(k+n,k)$. No information about states preceeding or following $x(k)$ is required.

If external additive noise is applied to the system, equation (2.1) must be modified to

$$x(k+1) = \phi(k+1,k)x(k) + \Gamma(k+1,k)w(k) \tag{2.2}$$

where $w(k)$ is a random vector sequence representing noise and $\Gamma(k+1,k)$ is a known matrix indicating how $w(k)$ affects each component of $x(k+1)$.

In most systems the state vector $x(k)$ is not directly observed. Instead, a measurement $z(k)$ which is related to $x(k)$ by the measuring system $H(k)$ is observed. The relation between $z(k)$ and $x(k)$ is described by

$$z(k) = H(k)x(k) \tag{2.3}$$

where $H(k)$ is a known matrix describing the operation of the measuring system.

Since no measuring system is free of noise, equation (2.3) must be modified to

$$z(k) = H(k)x(k) + v(k) \qquad (2.4)$$

where $v(k)$ is the noise vector sequence associated with the measurement of $x(k)$.

The problem is: Given the system and measurement models, (equations (2.2) and (2.4)), some statistical properties of the noise and the past history of measurements, what is the best estimate of the state at some given time? If some past state is to be estimated, the process is called smoothing. If the present state is being estimated, it is called filtering. Estimation of the future is called predicting.

2.2 The Discrete Kalman Filter

2.2.1 Introduction

The Kalman or related filters such as least squares, maximum likelihood of Bayesian estimators, are often used for state estimation of a dynamic system. Since all reduce to the Kalman form under assumptions of gaussianess of random sequences and first order Markovian properties, attention will be restricted to the Kalman filter. Also, since smoothing is not often done and the filtering or prediction problems involve an identical algorithm, that algorithm will be employed.

Very breifly, the state of a system is a vector quantity which encodes all of the system history that needs to be known for a particular purpose (?2).

In general, it cannot be determined or predicted exactly due to measurement noise, random inputs to the system or incomplete knowledge of the system parameters. To avoid the identification problem and its exposition, system parameters are assumed to be known exactly in this discussion.

Although the state cannot be precisely determined, it is possible to estimate it in some optimal fashion by the appropriate processing of available data. Thus, the problem is that of estimating the state of a system based on the available measurements and on knowledge about system parameters and noise statistics.

2.2.2 The System Model

The discrete system model is described by

$$x(k+1) = \phi(k+1,k)x(k) + \Gamma(k+1,k)w(k) \tag{2.5}$$

and

$$z(k) = H(k)x(k) + v(k) \tag{2.6}$$

where $x(k)$ is the n-dimensional state vector, $\phi(k+1,k)$ is the system transition matrix, $z(k)$ is the m-dimensional observation vector, $m \leq n$, $H(k)$ is a matrix describing the operation of the measuring process, and $v(k)$ and $w(k)$ are vector Gaussian noise sequences.

The means and covariances of v and w are

$$E[v(k)] = 0 \quad \text{for all } k \tag{2.7}$$

$$E[w(k)] = 0 \quad \text{for all } k \tag{2.8}$$

$$\text{cov}[w(k)] = E[w(k)w'(n)] = \delta(k,n)Q(k) \quad \text{for all } k,n \tag{2.9}$$

$$\text{cov}[v(k)] = E[v(k)v'(n)] = \delta(k,n)R(k)$$

$$\text{for all } k,n \qquad (2.10)$$

where $\delta(k,n)$ is the Kronecker delta.

It is usually assumed that no correlation exists between noise driving the plant and noise in the observation system. Should such correlation exist, a covariance matrix is given as

$$\text{cov}[v(k)w(n)] = E[v(k)w'(k)] = \delta(k,n)C(k)$$

$$\text{for all } k,n. \qquad (2.11)$$

Appropriate filter equations for non-zero $C(k)$ can be found in reference (3). For the discussion and example to follow, no correlation is assumed and $C(k) = 0$. Also $\phi(k,k-1)$, $\Gamma(k,k-1)$, $H(k)$, $Q(k)$, and $R(k)$ are known since the identification problem is not to be dealt with.

Suppose that a sequence of observations $z(0)$, $z(1)$, ...$z(k)$ is made. Based on these observations and a knowledge of the system and noise parameters, it is desired to make an estimate of the state vector $x(n)$. Notationally this is shown as $\hat{x}(n|k)$ and is read as "the optimal estimate of $x(n)$ given observations up through time kT". For filtering, $n=k$; for prediction, $n>k$, most usually $n=k+1$. The prediction problem for $n=k+1$ (which is computationally identical to the filtering problem) will be discussed.

The estimation error is defined as

$$\tilde{x}(k+1|k) = x(k+1) - \hat{x}(k+1|k) \qquad (2.12)$$

and the measure of performance is the mean squared error, which is the sum of variances in the case of an unbiased estimator such as the Kalman filter.

$$\overline{e^2} = E[\tilde{\hat{x}}'(k+1|k)\tilde{\hat{x}}(k+1|k)].$$

In terms of the covariance matrix of $\tilde{\hat{x}}$,

$$\Sigma(k+1|k) = E[\tilde{\hat{x}}(k+1|k)\tilde{\hat{x}}'(k+1|k)] \tag{2.14}$$

the mean squared error can be expressed as

$$\overline{e^2} = tr\Sigma(k+1|k). \tag{2.15}$$

2.2.3 The Solution

Kalman (3) has solved the problem posed in Section 2.2.2 by using the projection theorem from the theory of linear spaces. The equations describing the optimal filter are

$$\hat{x}(k+1|k) = \phi(k+1,k)\hat{x}(k|k) \tag{2.16}$$

$$\hat{x}(k|k) = \hat{x}(k|k-1) - G(k)[H(k)\hat{x}(k|k-1) - z(k)] \tag{2.17}$$

$$G(k) = \Sigma(k|k-1)H'(k)[H(k)\Sigma(k|k-1)H'(k) + R(k)]^{-1} \tag{2.18}$$

$$\Sigma(k|k-1) = \phi(k,k-1)\Sigma(k-1|k-1)\phi'(k,k-1) + \Gamma(k,k-1)Q(k-1)\Gamma'(k,k-1) \tag{2.19}$$

$$\Sigma(k|k) = [I - G(k)H(k)]\Sigma(k|k-1) \tag{2.20}$$

where G(k) is a matrix called the optimal filter gain matrix. The filtered estimate of x(k) is $\hat{x}(k|k)$ and the predicted estimate is $\hat{x}(k+1|k)$. It is worthwhile to interpret 2.17 as showing the filtered value to be the predicted value modified by the gain matrix times the difference between predicted and actual observation.

As indicated in the equations, $\hat{x}(k+1|k)$ is a processed version of $\hat{x}(k|k-1)$, updated by the most recent observation z(k). It is not necessary, therefore, to store the sequence of previous observations, z(0), z(1), ..., z(k-1). All relevant information contained in these

observations is contained in the vector $\hat{x}(k|k-1)$ and matrix $\Sigma(k|k-1)$. It is this recursive nature that makes the Kalman and related filters computationally attractive.

2.2.4 Computation

To start the computation, initial value $\hat{x}(0|0)$ for the state vector and $\Sigma(0|0)$ for the covariance matrix are assumed. These choices having been made, $\Sigma(1|0)$ is calculated according to equation (2.20) and $G(1)$ is calculated according to equation (2.19). The estimation $\hat{x}(1|0)$ is then determined using equations (2.17) and (2.18). $\Sigma(1|0)$ and $\hat{x}(1|0)$ are then used along with the next observation $z(1)$ to determine $\Sigma(2|1)$ and $\hat{x}(2|1)$, and so on.

The computations involve several matrix additions, subtractions and multiplications and the inversion of one matrix for each observation time. For this reason a machine which is used to perform the Kalman filter algorithm should be able to do these operations on matrices efficiently. The next section describes an algorithm for the multiplication of two matrices using an array-structured computer. An algorithm for inverting a matrix in an array-structured computer is given in the succeeding section.

2.3 Matrix Multiplication Algorithm

In this section an algorithm for the multiplication of two matrices using an array-structured computer is given.

It is desired to multiply two n x n matrices using the conventional definition of matrix multiplication. Assume, for the sake of demonstrating the multiplying scheme, that two 3 x 3 matrices A and B are defined by

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & b_4 & b_7 \\ b_2 & b_5 & b_8 \\ b_3 & b_6 & b_9 \end{bmatrix}$$

where the numbering scheme for the elements was chosen to provide a matrix product whose elements show some form of symmetry as will be seen later.

By the conventional definition of matrix product, if A is multiplied by B, the result, call it C, is given by

$$C = AxB = \begin{bmatrix} a_1b_1+a_2b_2+a_3b_3 & a_1b_4+a_2b_5+a_3b_6 & a_1b_7+a_2b_8+a_3b_9 \\ a_4b_1+a_5b_2+a_6b_3 & a_4b_4+a_5b_5+a_6b_6 & a_4b_7+a_5b_8+a_6b_9 \\ a_7b_1+a_8b_2+a_9b_3 & a_7b_4+a_8b_5+a_9b_6 & a_7b_7+a_8b_8+a_9b_9 \end{bmatrix}.$$

The symmetry of this product can be seen by comparing the ij^{th} element with the ji^{th} element and noticing that one is obtained from the other by inter-changing the subscripts on the a's and the b's.

The first steps in the multiplication scheme are to clear all registers and load the matrices A and B into the array. After the matrices are loaded into the array they are shifted according to the following rules.

A. 1. The first row of A is left alone.

2. The second row of A is shifted left one column.

3. The third row of A is shifted left two columns.

(Note, in general the i^{th} row of A is shifted left i-1 columns for i = 1, ..., n).

B. 1. The first column of B is left alone.

2. The second column of B is shifted up one row.

3. The third column of B is shifted up two rows.

(Note, in general the j^{th} column of B is shifted up j-1 rows for j = 1, ..., n)

Once the registers have been shifted the multiplication process can begin. The contents of the A register of a cell are multiplied by the contents of the B register of the cell and the result is added to the contents of the C register and stored there. This operation will be called element multiplication. The contents of the A and B registers remain unchanged. Now the contents of all A registers are shifted right one cell. In a similar fashion the contents of all B registers are shifted down one cell. The cycle is now repeated. A is element multiplied by B and the result is added to C. A is shifted right one cell, B is shifted down one cell. The cycle is repeated n times in general (three in the example).

The operation can be seen more clearly as the 3 x 3 example is gone through. The registers are first loaded with A and B

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & b_4 & b_7 \\ b_2 & b_5 & b_8 \\ b_3 & b_6 & b_9 \end{bmatrix}.$$

A and B are shifted using the rules given. Note that A, B and C represent the contents of the A, B, and C registers respectively.

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_5 & a_6 & a_4 \\ a_9 & a_7 & a_8 \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & b_5 & b_9 \\ b_2 & b_6 & b_7 \\ b_3 & b_4 & b_8 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

A is element multiplied by B and the result is added to C.

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_5 & a_6 & a_4 \\ a_9 & a_7 & a_8 \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & b_5 & b_9 \\ b_2 & b_6 & b_7 \\ b_3 & b_4 & b_8 \end{bmatrix}$$

$$C = \begin{bmatrix} a_1b_1 & a_2b_5 & a_3b_9 \\ a_5b_2 & a_6b_6 & a_4b_7 \\ a_9b_3 & a_7b_4 & a_8b_8 \end{bmatrix}$$

A is shifted right and B is shifted down one cell.

$$A = \begin{bmatrix} a_3 & a_1 & a_2 \\ a_4 & a_5 & a_6 \\ a_8 & a_9 & a_7 \end{bmatrix} \qquad B = \begin{bmatrix} b_3 & b_4 & b_8 \\ b_1 & b_5 & b_9 \\ b_2 & b_6 & b_7 \end{bmatrix}$$

$$C = \begin{bmatrix} a_1b_1 & a_2b_5 & a_3b_9 \\ a_5b_2 & a_6b_6 & a_4b_7 \\ a_9b_3 & a_7b_4 & a_8b_8 \end{bmatrix}$$

A is element multiplied by B and the result is added to C.

$$A = \begin{bmatrix} a_3 & a_1 & a_2 \\ a_4 & a_5 & a_6 \\ a_8 & a_9 & a_7 \end{bmatrix} \qquad B = \begin{bmatrix} b_3 & b_4 & b_8 \\ b_1 & b_5 & b_9 \\ b_2 & b_6 & b_7 \end{bmatrix}$$

$$C = \begin{bmatrix} a_1b_1+a_3b_3 & a_2b_5+a_1b_4 & a_3b_9+a_2b_8 \\ a_5b_2+a_4b_1 & a_6b_6+a_5b_5 & a_4b_7+a_6b_9 \\ a_9b_3+a_8b_2 & a_7b_4+a_9b_6 & a_8b_8+a_7b_7 \end{bmatrix}$$

A is shifted right and B is shifted down one cell.

$$A = \begin{bmatrix} a_2 & a_3 & a_1 \\ a_6 & a_4 & a_5 \\ a_7 & a_8 & a_9 \end{bmatrix} \qquad B = \begin{bmatrix} b_2 & b_6 & b_7 \\ b_3 & b_4 & b_8 \\ b_1 & b_5 & b_9 \end{bmatrix}$$

$$C = \begin{bmatrix} a_1b_1+a_3b_3 & a_2b_5+a_1b_4 & a_3b_9+a_2b_8 \\ a_5b_2+a_4b_1 & a_6b_6+a_5b_5 & a_4b_7+a_6b_9 \\ a_9b_3+a_8b_2 & a_7b_4+a_9b_6 & a_8b_8+a_7b_7 \end{bmatrix}$$

A is element multiplied by B and the result is added to C.

$$A = \begin{bmatrix} a_2 & a_3 & a_1 \\ a_6 & a_4 & a_5 \\ a_7 & a_8 & a_9 \end{bmatrix} \qquad B = \begin{bmatrix} b_2 & b_6 & b_7 \\ b_3 & b_4 & b_8 \\ b_1 & b_5 & b_9 \end{bmatrix}$$

$$C = \begin{bmatrix} a_1b_1+a_3b_3+a_2b_2 & a_2b_5+a_1b_4+a_3b_6 & a_3b_9+a_2b_8+a_1b_7 \\ a_5b_2+a_4b_1+a_6b_3 & a_6b_6+a_5b_5+a_4b_4 & a_4b_7+a_6b_9+a_5b_8 \\ a_9b_3+a_8b_2+a_7b_1 & a_7b_4+a_9b_6+a_8b_5 & a_8b_8+a_7b_7+a_9b_9 \end{bmatrix}$$

C now contains the matrix product of the original A and B matrices.

2.4 Matrix Inversion Algorithm

In this section an algorithm is presented for obtaining the inverse of a matrix. The algorithm is Gauss's algorithm modified to make efficient use of an array processor. The modifications consist of performing operations on all of the elements of a row, column or matrix simultaneously wherever possible.

The algorithm is presented in the form of an APL program. The program assumes a machine with two memory registers (M), a right-shifting matrix register (A) with an augmented column vector register (R), a down-shifting matrix register (B) with an augmented row vector register (C), an adder-subtractor with three matrix registers (AA, AB and AC) and a multiplier with three matrix registers (MA, MB and MC). In the above context a "matrix register" means an array of registers used to store a matrix. The "vector register" means a string of registers used to store a vector.

The program, which is shown in Figure 2.1, is intended to demonstrate the algorithm, not to produce an optimum realization of it. A numerical example which demonstrates the algorithm follows.

```
      ∇ INVERT
[  1] N←(⍴MATRIX)[1]
[  2] C←N⍴0
[  3] M←(2,N,N)⍴0
[  4] M[1;;]←MATRIX
[  5] K←1
[  6] LOOP:A←M[1;;]
[  7] R←N⍵1
[  8] T←(0 1+⍴A)⍴0
[  9] T[;1]←R
[ 10] T[;1+⍳N]←A
[ 11] A←T[;⍳N]
[ 12] R←T[;N+1]
[ 13] R[N]←1÷R[N]
[ 14] M[1;;]←A
[ 15] A←R∘.×N⍴1
[ 16] M[2;;]←A
[ 17] MA←M[1;;]
[ 18] MB←M[2;;]
[ 19] MC←MA×MB
[ 20] M[1;N;]←MC[N;]
[ 21] B←M[1;;]
[ 22] T←(1 0+⍴B)⍴0
[ 23] T[1;]←C
[ 24] T[1+⍳N;]←B
[ 25] B←T[⍳N;]
[ 26] C←T[N+1;]
[ 27] B←(N,N)⍴C
[ 28] MA←M[2;;]
[ 29] M[2;;]←B
[ 30] MB←M[2;;]
[ 31] MC←MA×MB
[ 32] M[2;;]←MC
[ 33] AA←M[1;;]
[ 34] AB←M[2;;]
[ 35] AC←AA-AB
[ 36] M[1;⍳N-1;]←AC[⍳N-1;]
[ 37] B←M[1;¯1↑⍳N;]
[ 38] M[1;;]←B
[ 39] K←K+1
[ 40] →(K≤N)/LOOP
[ 41] INVERSE←M[1;;]
      ∇
```

Figure 2.1. Parallel Matrix Inversion Algorithm

As an example of obtaining the inverse of a matrix consider Gauss's algorithm modified for the array structure. A 3x3 array is used for brevity.

Find the inverse of the matrix

$$\begin{bmatrix} 3.0 & 0.5 & 1.0 \\ 0.5 & 1.0 & 0.5 \\ 1.0 & 0.5 & 0.5 \end{bmatrix}.$$

Load the matrix into the routing A register and set the row buffers as indicated giving

$$\left[\begin{array}{c|ccc} 0.0 & 3.0 & 0.5 & 1.0 \\ 0.0 & 0.5 & 1.0 & 0.5 \\ 1.0 & 1.0 & 0.5 & 0.5 \end{array}\right].$$

Rotate the array, including the row buffers, one cell to the right to get

$$\left[\begin{array}{c|ccc} 1.0 & 0.0 & 3.0 & 0.5 \\ 0.5 & 0.0 & 0.5 & 1.0 \\ 0.5 & 1.0 & 1.0 & 0.5 \end{array}\right]$$

Invert the last-row element of the row buffers giving

$$\left[\begin{array}{c|ccc} 1.0 & 0.0 & 3.0 & 0.5 \\ 0.5 & 0.0 & 0.5 & 1.0 \\ 2.0 & 1.0 & 1.0 & 0.5 \end{array}\right].$$

Store the routing A register in memory location 1. Location 1 now contains

$$\begin{bmatrix} 0.0 & 3.0 & 0.5 \\ 0.0 & 0.5 & 1.0 \\ 1.0 & 1.0 & 0.5 \end{bmatrix} .$$

Broadcast (by routing) the row buffers across the routing A register to get

$$\left[\begin{array}{c|ccc} 1.0 & 1.0 & 1.0 & 1.0 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 2.0 & 2.0 & 2.0 & 2.0 \end{array}\right] .$$

Store the routing A register in memory location 2. Location 2 now contains

$$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 0.5 & 0.5 & 0.5 \\ 2.0 & 2.0 & 2.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the multiply A register and the contents of memory location 2 into the multiply B register. Multiply, and store the last row only back into memory location 1. Location 1 now contains

$$\begin{bmatrix} 0.0 & 3.0 & 0.5 \\ 0.0 & 0.5 & 1.0 \\ 2.0 & 2.0 & 1.0 \end{bmatrix}$$

Load the contents of memory location 1 into the routing B register. Rotate the array, including the column buffers, down one

cell to place the bottom row in the column buffers. Then broadcast (by routing) the column buffers down the routing B register to get

$$\begin{bmatrix} 2.0 & 2.0 & 1.0 \\ 2.0 & 2.0 & 1.0 \\ 2.0 & 2.0 & 1.0 \end{bmatrix} .$$

Load the contents of memory location 2 into the multiply A register. Store the routing B register in memory location 2. Location 2 now contains

$$\begin{bmatrix} 2.0 & 2.0 & 1.0 \\ 2.0 & 2.0 & 1.0 \\ 2.0 & 2.0 & 1.0 \end{bmatrix} .$$

Load the contents of memory location 2 into the multiply B register. Multiply and store the result back in memory location 2. Location 2 now contains

$$\begin{bmatrix} 2.0 & 2.0 & 1.0 \\ 1.0 & 1.0 & 0.5 \\ 4.0 & 4.0 & 4.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the adder A register. Load the contents of memory location 2 into the adder B register. Subtract, and store all but the last row of the result back in memory location 1. Location 1 now contains

$$\begin{bmatrix} -2.0 & 1.0 & -0.5 \\ -1.0 & -0.5 & 0.5 \\ 2.0 & 2.0 & 1.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the routing B register. Rotate the array down one cell giving

$$\begin{bmatrix} 2.0 & 2.0 & 1.0 \\ -2.0 & 1.0 & -0.5 \\ -1.0 & -0.5 & 0.5 \end{bmatrix} .$$

Store the routing B register back into memory location 1. Location 1 now contains

$$\begin{bmatrix} 2.0 & 2.0 & 1.0 \\ -2.0 & 1.0 & -0.5 \\ -1.0 & -0.5 & 0.5 \end{bmatrix} .$$

Load the contents of memory location 1 into the routing A register and set the row buffers as indicated giving

$$\left[\begin{array}{c|ccc} 0.0 & 2.0 & 2.0 & 1.0 \\ 0.0 & -2.0 & 1.0 & -0.5 \\ 1.0 & -1.0 & -0.5 & 0.5 \end{array}\right] .$$

Rotate the array, including the row buffers, one cell to the right to get

$$\left[\begin{array}{c|ccc} 1.0 & 0.0 & 2.0 & 2.0 \\ -0.5 & 0.0 & -2.0 & 1.0 \\ 0.5 & 1.0 & -1.0 & -0.5 \end{array}\right] .$$

Invert the last-row element of the row buffers giving

$$\left[\begin{array}{c|ccc} 1.0 & 0.0 & 2.0 & 2.0 \\ -0.5 & 0.0 & -2.0 & 1.0 \\ 2.0 & 1.0 & -1.0 & -0.5 \end{array}\right] .$$

Store the routing A register in memory location 1. Location 1 now contains

$$\begin{bmatrix} 0.0 & 2.0 & 2.0 \\ 0.0 & -2.0 & 1.0 \\ 1.0 & -1.0 & -0.5 \end{bmatrix} .$$

Broadcast (by routing) the row buffers across the routing A register to get

$$\left[\begin{array}{c|ccc} 1.0 & 1.0 & 1.0 & 1.0 \\ -0.5 & -0.5 & -0.5 & -0.5 \\ 2.0 & 2.0 & 2.0 & 2.0 \end{array}\right] .$$

Store the routing A register in memory location 2. Location 2 now contains

$$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ -0.5 & -0.5 & -0.5 \\ 2.0 & 2.0 & 2.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the multiply A register and the contents of memory location 2 into the multiply B register. Multiply, and store the last row only back into memory location 1. Location 1 now contains

$$\begin{bmatrix} 0.0 & 2.0 & 2.0 \\ 0.0 & -2.0 & 1.0 \\ 2.0 & -2.0 & -1.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the routing B register. Rotate the array, including the column buffers, down one cell to place the bottom row in the column buffers. Then broadcast (by routing) the column buffers down the routing B register to get

$$\begin{bmatrix} 2.0 & -2.0 & -1.0 \\ 2.0 & -2.0 & -1.0 \\ 2.0 & -2.0 & -1.0 \end{bmatrix} .$$

Load the contents of memory location 2 into the multiply A register. Store the routing B register in memory location 2. Location 2 now contains

$$\begin{bmatrix} 2.0 & -2.0 & -1.0 \\ 2.0 & -2.0 & -1.0 \\ 2.0 & -2.0 & -1.0 \end{bmatrix} .$$

Load the contents of memory location 2 into the multiply B register. Multiply and store the result back in memory location 2. Location 2 now contains

$$\begin{bmatrix} 2.0 & -2.0 & -1.0 \\ -1.0 & 1.0 & 0.5 \\ 4.0 & -4.0 & -2.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the adder A register. Load the contents of memory location 2 into the adder B register. Subtract, and store all but the last row of the result back in memory location 1. Location 1 now contains

$$\begin{bmatrix} -2.0 & 4.0 & 3.0 \\ 1.0 & -3.0 & 0.5 \\ 2.0 & -2.0 & -1.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the routing B register. Rotate the array down one cell giving

$$\begin{bmatrix} 2.0 & -2.0 & -1.0 \\ -2.0 & 4.0 & 3.0 \\ 1.0 & -3.0 & 0.5 \end{bmatrix} .$$

Store the routing B register back into memory location 2. Location 2 now contains

$$\begin{bmatrix} 2.0 & -2.0 & -1.0 \\ -2.0 & 4.0 & 3.0 \\ 1.0 & -3.0 & 0.5 \end{bmatrix} .$$

Load the contents of memory location 1 into the routing A register and set the row buffers as indicated giving

$$\left[\begin{array}{c|ccc} 0.0 & 2.0 & -2.0 & -1.0 \\ 0.0 & -2.0 & 4.0 & 3.0 \\ 1.0 & 1.0 & -3.0 & 0.5 \end{array}\right] .$$

Rotate the array, including the row buffers, one cell to the right to get

$$\left[\begin{array}{c|ccc} -1.0 & 0.0 & 2.0 & -2.0 \\ 3.0 & 0.0 & -2.0 & 4.0 \\ 0.5 & 1.0 & 1.0 & -3.0 \end{array}\right] .$$

Invert the last-row element of the row buffers giving

$$\left[\begin{array}{c|ccc} -1.0 & 0.0 & 2.0 & -2.0 \\ 3.0 & 0.0 & -2.0 & 4.0 \\ 2.0 & 1.0 & 1.0 & -3.0 \end{array}\right] .$$

Store the routing A register in memory location 1. Location 1 now contains

$$\begin{bmatrix} 0.0 & 2.0 & -2.0 \\ 0.0 & -2.0 & 4.0 \\ 1.0 & 1.0 & -3.0 \end{bmatrix} .$$

Broadcast (by routing) the row buffers across the routing A register to get

$$\left[\begin{array}{c|ccc} -1.0 & -1.0 & -1.0 & -1.0 \\ 3.0 & 3.0 & 3.0 & 3.0 \\ 2.0 & 2.0 & 2.0 & 2.0 \end{array}\right] .$$

Store the routing A register in memory location 2. Location 2 now contains

$$\begin{bmatrix} -1.0 & -1.0 & -1.0 \\ 3.0 & 3.0 & 3.0 \\ 2.0 & 2.0 & 2.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the multiply A register and the contents of memory location 2 into the multiply B register. Multiply, and store the last row only back into memory location 1. Location 1 now contains

$$\begin{bmatrix} 0.0 & 2.0 & -2.0 \\ 0.0 & -2.0 & 4.0 \\ 2.0 & 2.0 & -6.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the routing B register. Rotate the array, including the column buffers, down one cell to place the bottom row in the column buffers. Then broadcast (by routing) the column buffers down the routing B register to get

$$\begin{bmatrix} 2.0 & 2.0 & -6.0 \\ 2.0 & 2.0 & -6.0 \\ 2.0 & 2.0 & -6.0 \end{bmatrix} .$$

Load the contents of memory location 2 into the multiply A register. Store the routing B register in memory location 2. Location 2 now contains

$$\begin{bmatrix} 2.0 & 2.0 & -6.0 \\ 2.0 & 2.0 & -6.0 \\ 2.0 & 2.0 & -6.0 \end{bmatrix} .$$

Load the contents of memory location 2 into the multiply B register. Multiply and store the result back in memory location 2. Location 2 now contains

$$\begin{bmatrix} -2.0 & -2.0 & 6.0 \\ 6.0 & 6.0 & -18.0 \\ 4.0 & 4.0 & -12.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the adder A register. Load the contents of memory location 2 into the adder B register.

Subtract, and store all but the last row of the result back in memory location 1. Location 1 now contains

$$\begin{bmatrix} 2.0 & 4.0 & -8.0 \\ -6.0 & -8.0 & 22.0 \\ 2.0 & 2.0 & -6.0 \end{bmatrix} .$$

Load the contents of memory location 1 into the routing B register. Rotate the array down one cell giving

$$\begin{bmatrix} 2.0 & 2.0 & -6.0 \\ 2.0 & 4.0 & -8.0 \\ -6.0 & -8.0 & 22.0 \end{bmatrix} .$$

The routing B register now contains the inverse of the original matrix.

2.5 Kalman Filter Example

To provide an example of a use of the Kalman filter algorithm, a problem which has been solved by Kolb and Hollister [9], using the algorithm is considered.

The problem concerns the estimation of the motion of a target in a plane, given only bearing measurements corrupted by zero-mean Gaussian noise. The observer is assumed to be moving in the samp plane as the target. Sequential observations of the bearing of the target from the observer are made and the problem is to estimate the position and velocity of the target. The observations are assumed to be disturbed by zero-mean Gaussian distributed errors. While the target velocity is assumed constant, the Kalman theory allows the system being estimated to be excited by a ramdon sequence with known mean and variance. Therefore, the filter is able to tolerate some maneuvering of the target.

The target motion relative to the observer is approximated by

$$x(k+1) = \phi x(k) + \Gamma[w(k) - u(k)], \qquad (2.21)$$

where ϕ is the state transition matrix, $w(k)$ is a vector of random disturbances with covariance matrix Q, Γ is the distribution matrix and $u(k)$ is a vector of deterministic accelerations of the observer.

An observation $z(k)$ is determined by

$$z(k) = H(k)x(k) + v(k) \qquad (2.22)$$

where H(k) describes the measurement system and v(k) represents the measurement noise and has covariance matrix R.

The solutions given in (9) can be written as

$$\hat{x}(k|k-1) = x^{*}(k-1|k-1) - \Gamma u(k) \qquad (2.23)$$

$$\Sigma(k|k-1) = \phi\Sigma(k-1|k-1)\phi' + \Gamma Q\Gamma' \qquad (2.24)$$

$$G(k) = \Sigma(k|k-1)H'(k)[H(k)\Sigma(k|k-1)H'(k) + R]^{-1} \qquad (2.25)$$

$$\Sigma(k|k) = [I - G(k)H(k)]\Sigma(k|k-1) \qquad (2.26)$$

$$\hat{z}(k|k) = H(k)\hat{x}(k|k-1) \qquad (2.27)$$

$$x^{*}(k|k) = G(k)[z(k) - \hat{z}(k|k-1)] + \hat{x}(k|k-1) \qquad (2.28)$$

where $\hat{x}(k|k-1)$ is the estimate of the state x(k) based on k-1 observations, $x^{*}(k|k)$ is the estimate of x(k) based on k observations, $\Sigma(k|k-1)$ is the error covariance matrix for $\hat{x}(k|k-1)$, $\Sigma(k|k)$ is the error covariance matrix for $x^{*}(k|k)$, and G(k) is the Kalman gain matrix. The identity matrix is represented by I.

A FORTRAN IVH computer program was written to perform the Kalman filter algorithm as used in this problem. A listing of this program is given in Appendix A and a flow chart describing the program is shown in Figure 2.2. Table 2.1 shows the correspondence between the identifiers used in the program and those used in Equations 2.21 through 2.28.

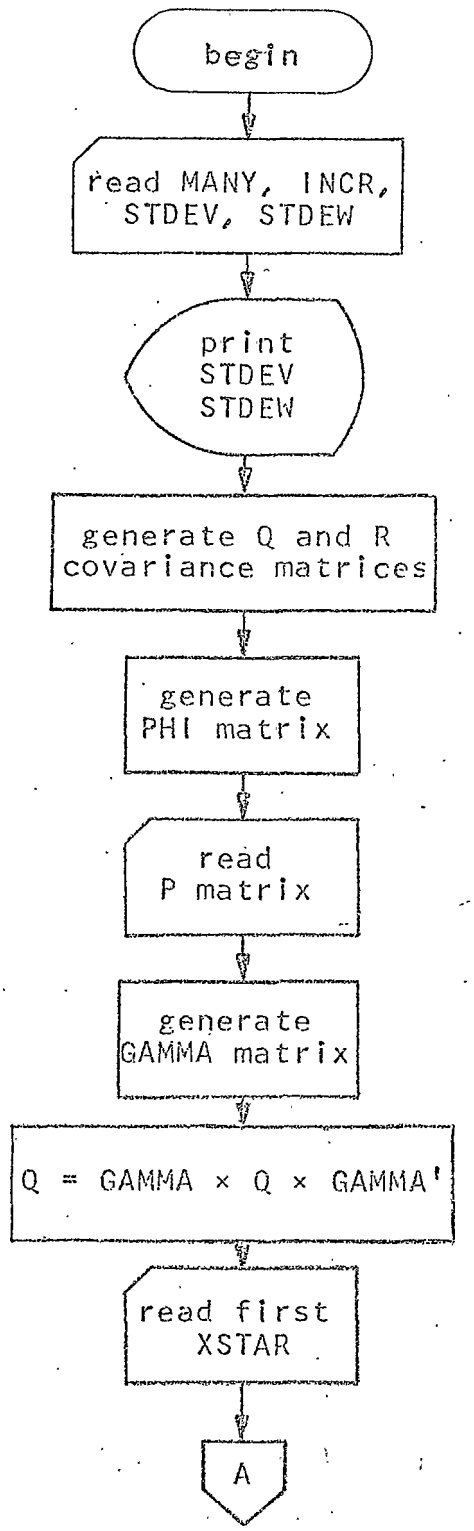


Figure 2.2. Flow Chart for Example Kalman Filter Program

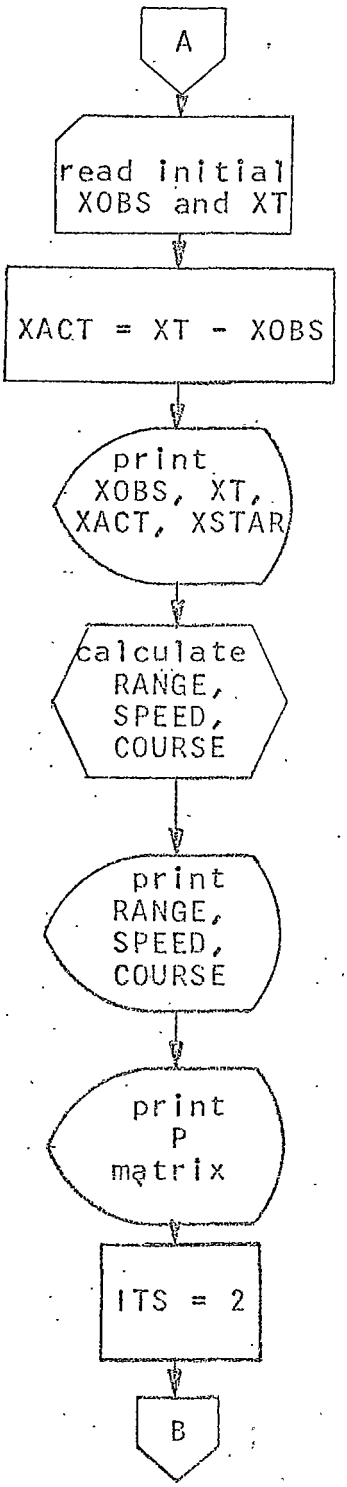


Figure 2.2. (Continued)

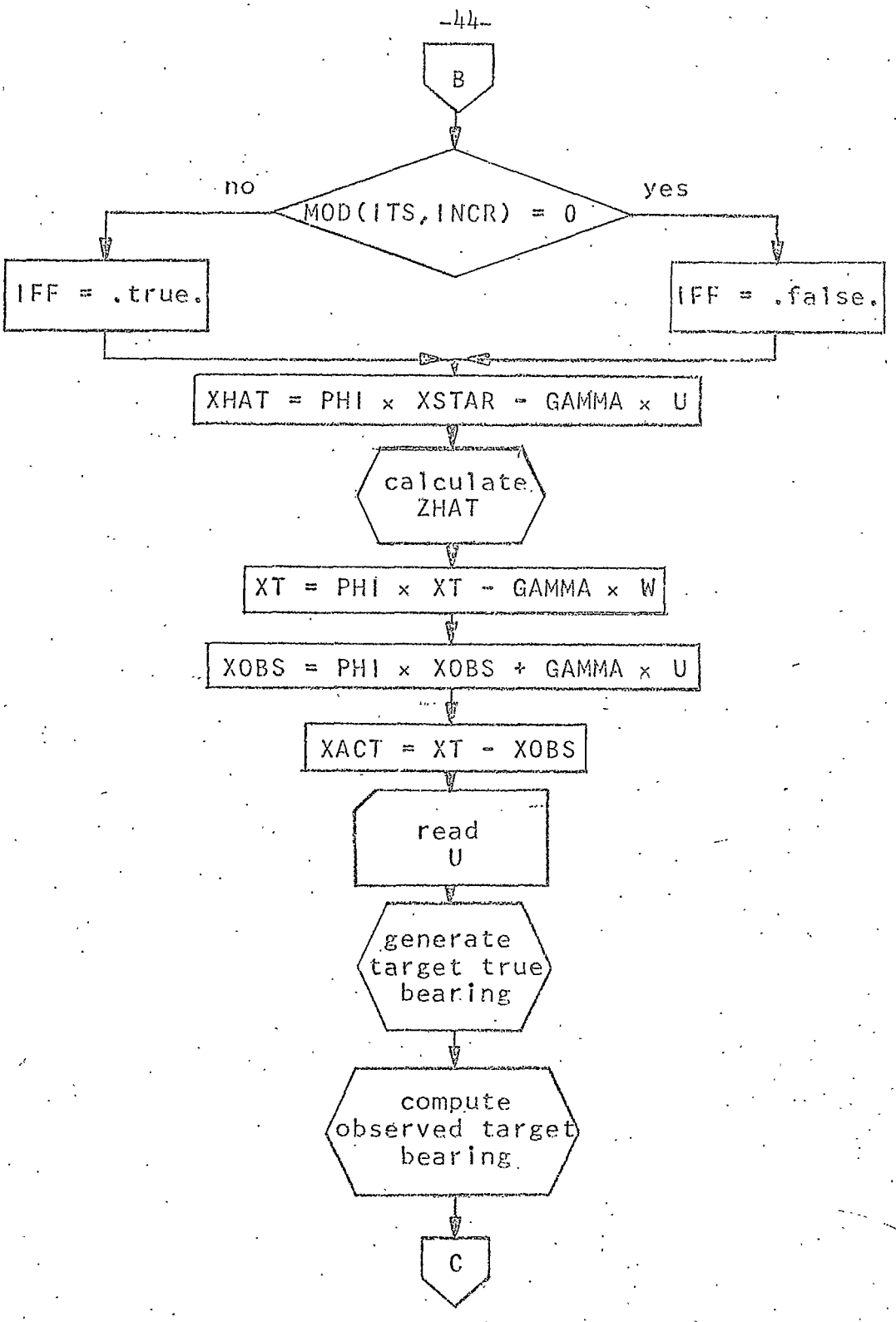


Figure 2.2. (Continued)

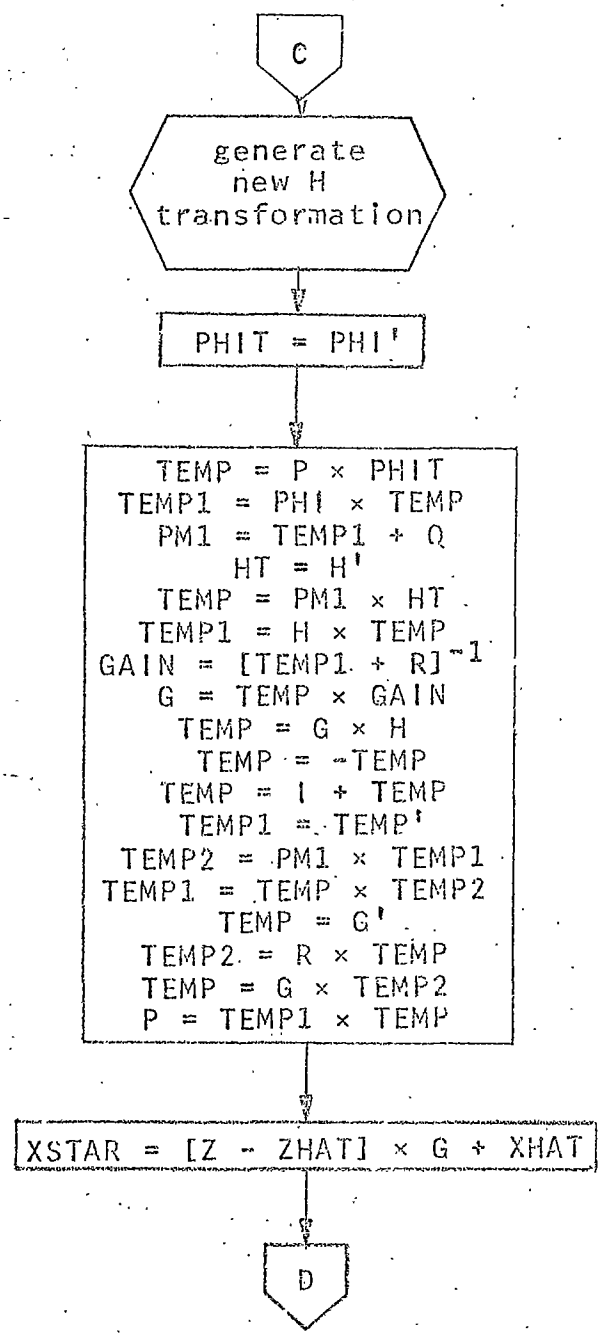


Figure 2.2. (Continued)

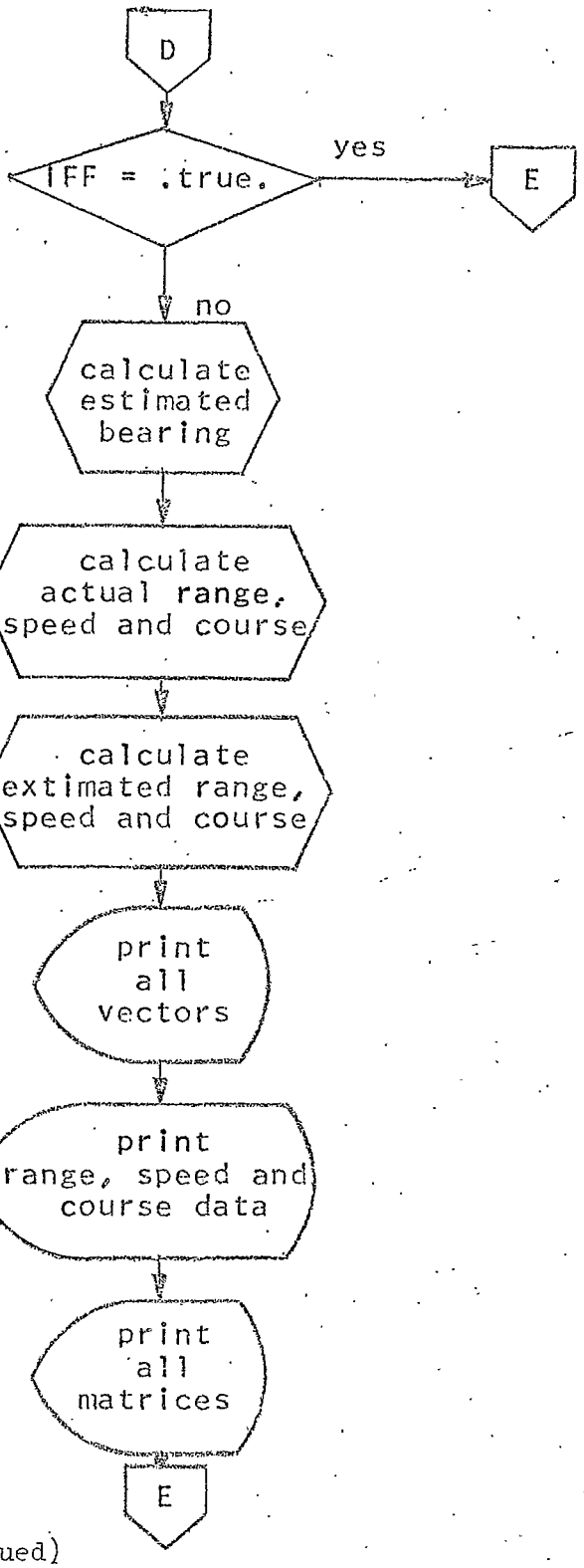


Figure 2.2. (Continued)

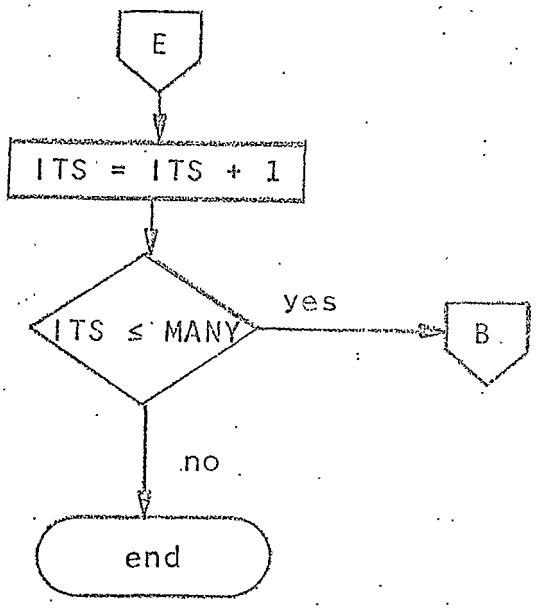


Figure 2.2. (Concluded)

Table 2.1. Identifiers for Kalman Filter Example

Identifier used in program	Purpose	Identifier used in equations
STDEV	Standard deviation of bearing error	-
STDEW	Standard deviation of target exciting noise	-
Q	Covariance matrix of target exciting noise	Q
R	Covariance matrix of bearing error	R
PHI	State transition matrix	ϕ
P	Error covariance matrix of $x^*(k\ k)$	$\Sigma(k\|k)$
PM1	Error covariance matrix of $\hat{x}(k\|k-1)$	$\Sigma(k\|k-1)$
GAMMA	Distribution matrix	Γ
XSTAR	Estimate of x(k) based on k observations	$x^*(k\|k)$
XHAT	Estimate of x(k) based on k-1 observations	$\hat{x}(k\|k-1)$
XOBS	State of observer	-
XT	State of target	-
XACT	Relative state of target	$x(k)$
Z	Actual observation	$z(k)$
ZHAT	Estimated observation	$\hat{z}(k\|k-1)$

Table 2.1. (Continued)

W	Target exciting noise	w(k)
V	Bearing noise	v(k)
U	Observer accelerations	u(k)
H	Linearized observation system	H(k)
HT	Transpose of H	H'(k)
PHIT	Transpose of PHI	ϕ'
G	The optimal filter gain	G(k)

Figure 2.3 shows a plot of the tracks of the observer, the target and the estimate of the target for a typical run of the Kalman filter example program. An x denotes the initial states of the target and of the observer and the initial estimation. Each dot on the tracks indicates 20 iterations. This is done to provide some time relationship between the tracks. The axes are marked in kilometers. It can be seen that the estimation quickly converges to the target track.

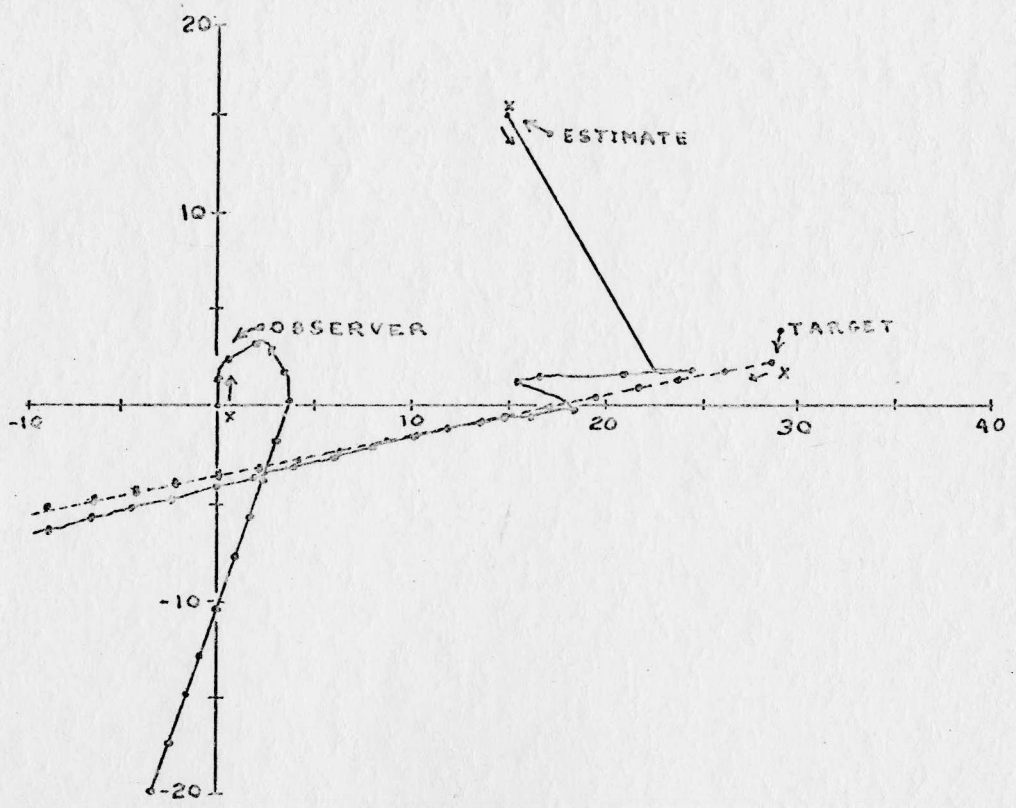


Figure 2.3. Plot of Observer, Target and Estimate Tracks

Chapter 3

DESIGN OF A CELLULAR COMPUTER

3.1 Introduction

In this chapter a design is presented for a computer which is specially structured for the types of operations used in the Kalman filter algorithm. The basic structure of the computer is given followed by detailed descriptions of the various parts. This computer will hereafter be referred to as the KF machine.

As indicated in Section 2.2 the Kalman filter algorithm involves a large number of operations on matrices and vectors. These operations include the addition, subtraction, multiplication, inversion and transposing of several matrices. Since many applications where the Kalman filter might be used require the calculations to be done as rapidly as possible, a computer used to perform these calculations should be able to perform them efficiently. Therefore, a computer with an array structure is proposed. Since not all operations deal with matrices or vectors, two types of instructions are suggested: (1) array instructions which perform operations on the matrices or vectors and (2) non-array instructions which perform operations on scalars.

3.2 KF Machine Structure

The proposed KF machine as shown in Figure 3.1 has four sections: an array of processing cells, row and column data and control registers and a global control unit.

The array of processing cells performs computations involving matrices and vectors. This type of data is stored in the array of processing cells with each ij^{th} cell containing the ij^{th} element of a matrix.

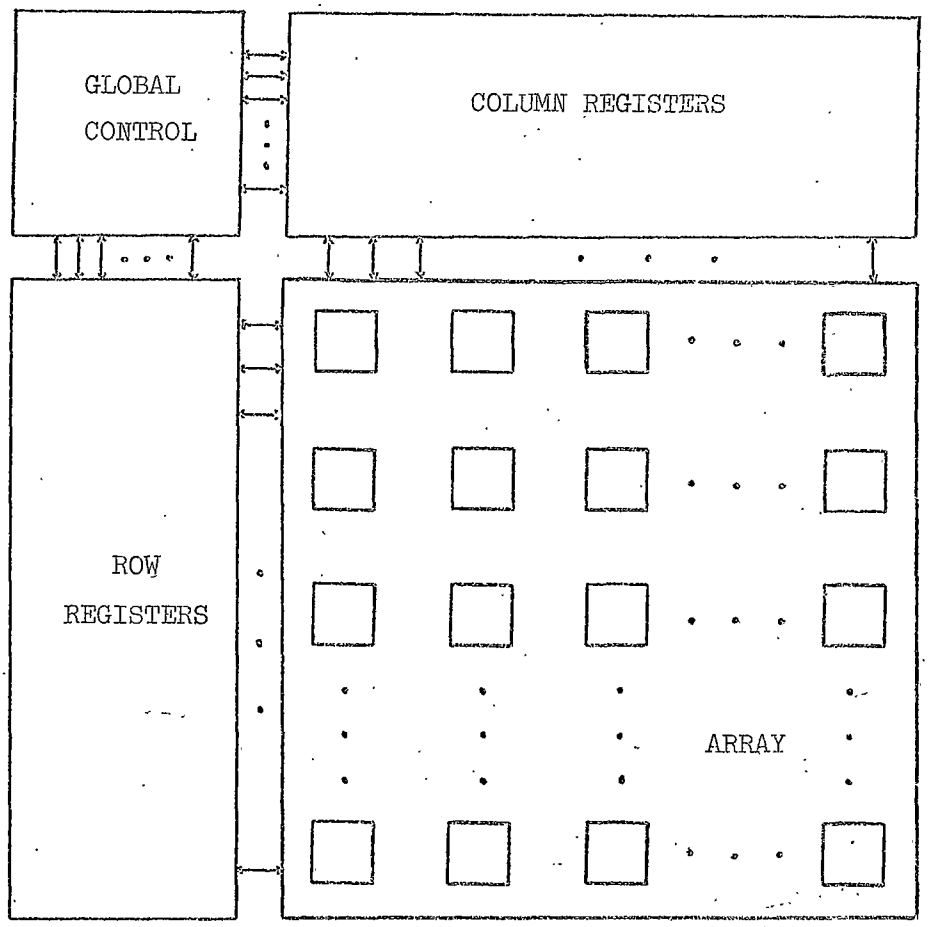


Figure 3.1. KF Machine Structure

Row and Column data and control registers provide an interface between the array of processing cells and the global control unit.

The purpose of the global control unit is to fetch instructions from its memory, decode the instructions and execute them. Non-array instructions are executed by the global control unit within its structure while array instructions are executed by the array of processing cells under the control of the global control unit.

The remainder of this chapter is devoted to the detailed description of component parts of the proposed KF machine.

3.3 Special Logic Units

In the descriptions of portions of the KF machine some specialized logic units are used extensively. This section describes some of these units and the conventions used for logic gates and for logic equations.

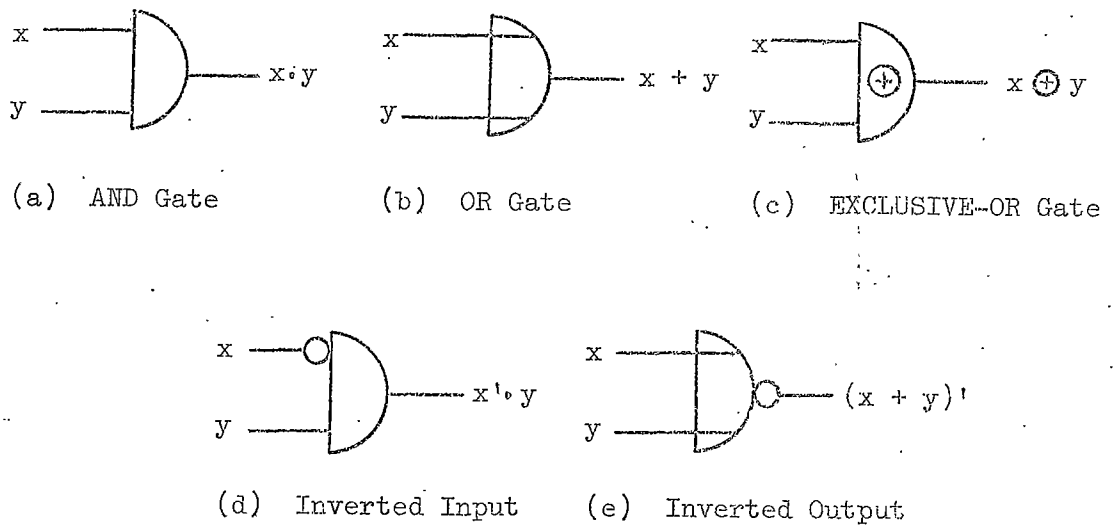
3.3.1 Logic Conventions

The specialized logic units to be described are assumed to be constructed of basic logic gates such as are shown in Figure 3.2. The inverted convention for inputs or outputs is indicated by Figures 3.2(d) and 3.2(e) respectively. For logic equations inverted variables are represented by an apostrophe (') following the variable. The logical OR operation is represented by a plus sign (+), the logical EXCLUSIVE-OR by a circled plus sign ($\oplus$) and the logical AND by a dot ($\cdot$).

3.3.2 Line-Select Gate

The line-select gate shown in Figure 3.3(a) is a three-input gate which realizes the function

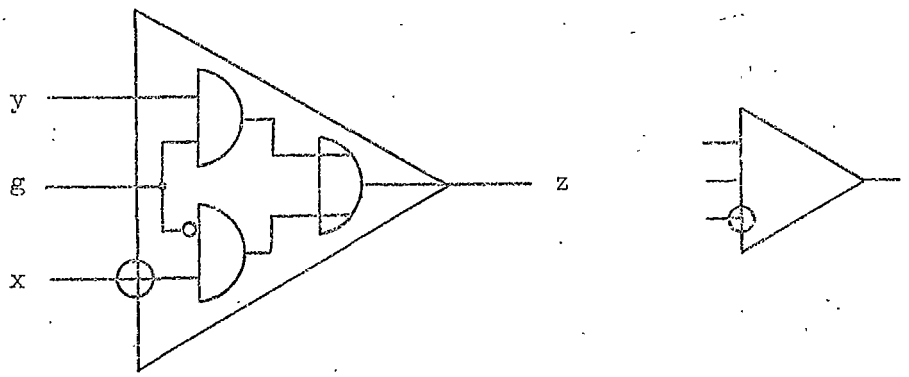
$$z = g' \cdot x + g \cdot y$$



(a) AND Gate (b) OR Gate (c) EXCLUSIVE-OR Gate

(d) Inverted Input (e) Inverted Output

Figure 3.2. Logic Gate Symbols.



(a) (b)

Figure 3.3 The Line-Select Gate

Its purpose is to select, as output z, the x input line if the gate line g is 0 or the y input line if the gate line g is 1. When used in other figures the gate will be represented as in Figure 3.3(b). The center input is always assumed to be g and the circled input is always assumed to be x.

3.3.3 Add-One Cell

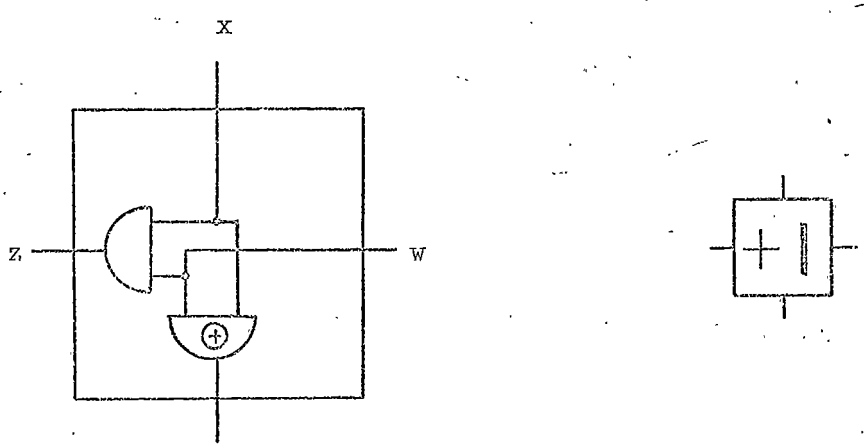
The add-one cell shown in Figure 3.4(a) is a two-input, two-output cell which produces the functions

$$z = x \cdot w$$

and

$$y = x \oplus w .$$

This cell was mentioned by Hennie [6] and later by Minnick [14].



(a) (b)

Figure 3.4. The Add-One Cell

When used in other figures the cell will be represented as in Figure 3.4(b).

The cell can be used to form a parallel add-one circuit for adding one to an N-bit binary number. N cells are connected in a cascade with

parallel inputs from the N-bit binary number as shown in Figure 3.5. This circuit is the one-dimensional decoder array discussed by Minnick (14). If the w input to the low-order add-one cell is supplied with a 0, the outputs y_i will be equal to their corresponding x_i's. However, if the input is a 1 the outputs y_i will represent a binary number whose value is one more than the input number.

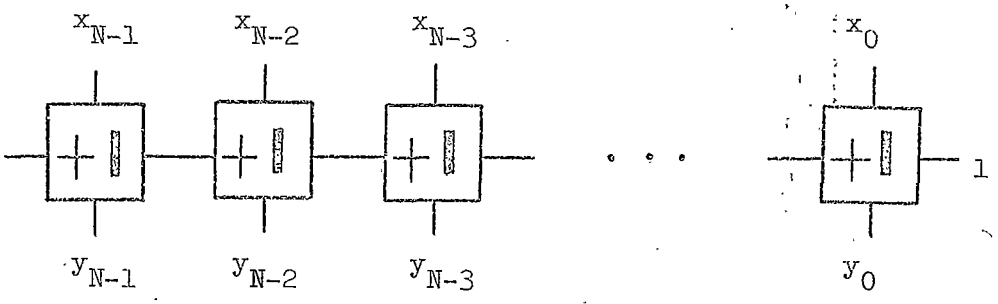


Figure 3.5. The Add-One Cascade

It should be noted that the cascade can be used to add any power of two if the 1 input is injected into the cascade at a higher-order bit position as indicated in the add-four cascade shown in Figure 3.6.

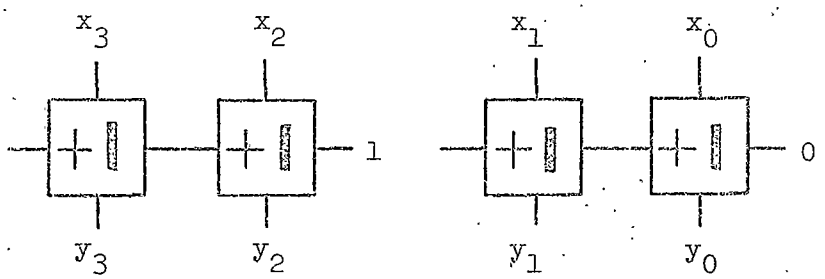


Figure 3.6. The Add-Four Cascade

3.3.4 One's-Two's Complementor Cell

The one's-two's complementor cell shown in Figure 3.6(a) is a two-input, two-output cell which produces the functions

$$c = a + b$$

and

$$d = a \oplus b.$$

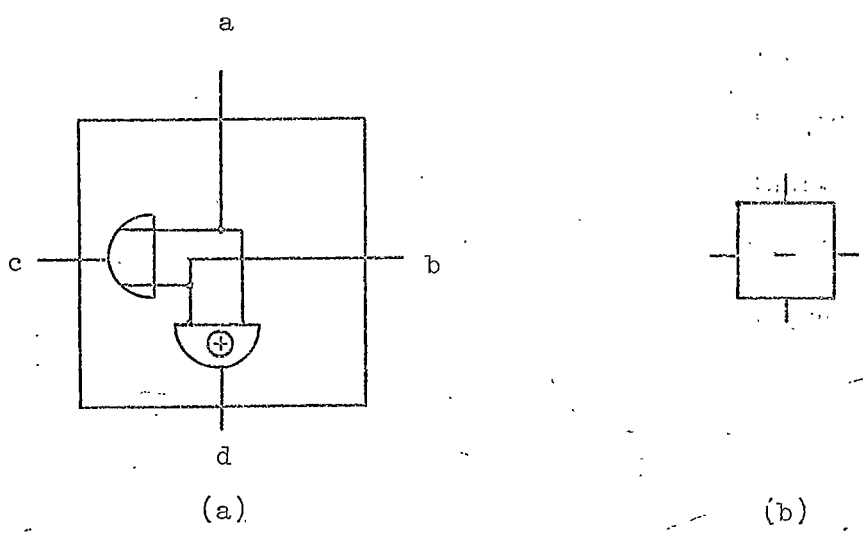


Figure 3.7. The One's-Two's Complementor Cell

When used in other figures the cell will be represented as in Figure 3.6(b).

The cell can be used to form a parallel one's-two's complementor circuit for N-bit binary numbers. N cells are connected in a cascade with parallel inputs a_i from the N-bit binary number as shown in Figure 3.8. If the b input of the low-order cell is supplied with a 0 the d_i outputs will represent the two's complement of the

binary number. However, if the input is supplied with a 1 the d_i outputs will represent the one's complement of the binary number.

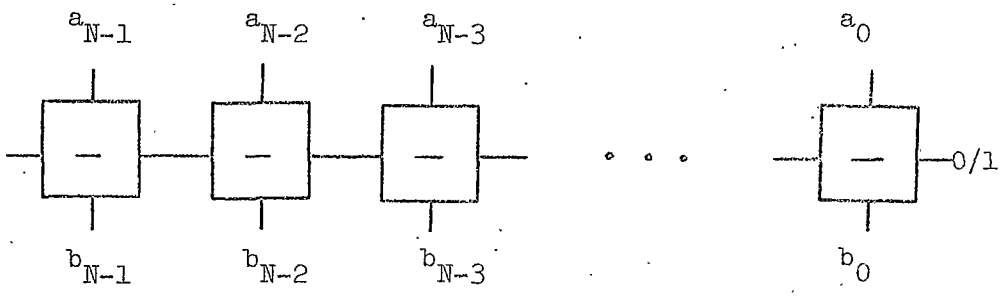


Figure 3.8. The One's-Two's Complementor Cascade

3.3.5 Comparator Cell

Much of the work on comparator circuits has been done by people such as Lee (10) and McKeever (12) who were concerned with associative memories.

The comparator cell shown in Figure 3.9(a) is a four-input, two-output cell which produces the functions

$$e = c + a \cdot b' \cdot d'$$

and

$$f = d + b \cdot a' \cdot c'$$

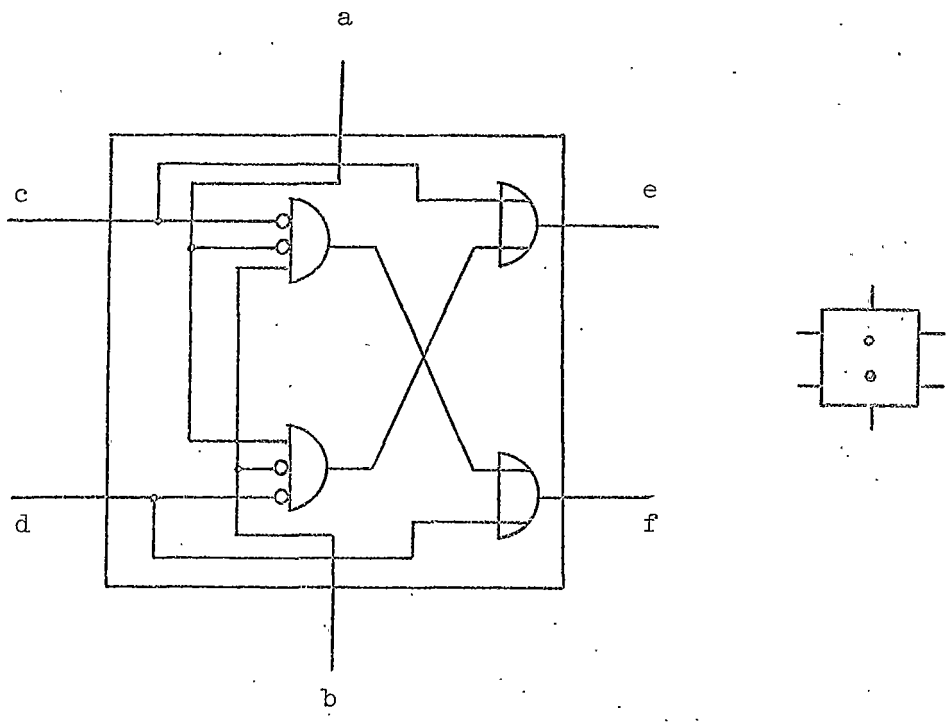


Figure 3.9. The Comparator Cell

When used in other figures the cell will be represented as in Figure 3.9(b).

The cell can be used to form a parallel comparator for comparing two N-bit binary numbers A and B. N cells are connected in a cascade with parallel inputs a_i from the binary number A, and b_i from the binary number B, as shown in Figure 3.10. The inputs c and d to the high-order comparator cell are supplied with 0's.

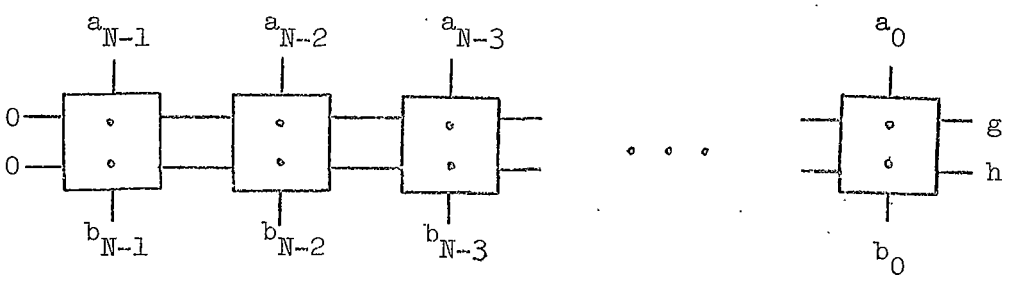


Figure 3.10. The Comparator Cascade

If the binary number A is larger than the binary number B the output line g will be a 0 and the output line h will be a 1. Similarly, if B is larger than A, the output line g will be a 1 and the output line h will be a 0. If the numbers are equal, both output lines will be 0. Thus, it is possible to detect the conditions A>B, A = B or A<B. The condition A = B is given by the function

$$k = g' \cdot h'.$$

3.3.6 Adder Cell

The adder cell shown in Figure 3.11(a) is a three-input, two-output cell which produces the functions

$$z = x \oplus y \oplus w$$

and

$$n = x \cdot y + y \cdot w + x \cdot w$$

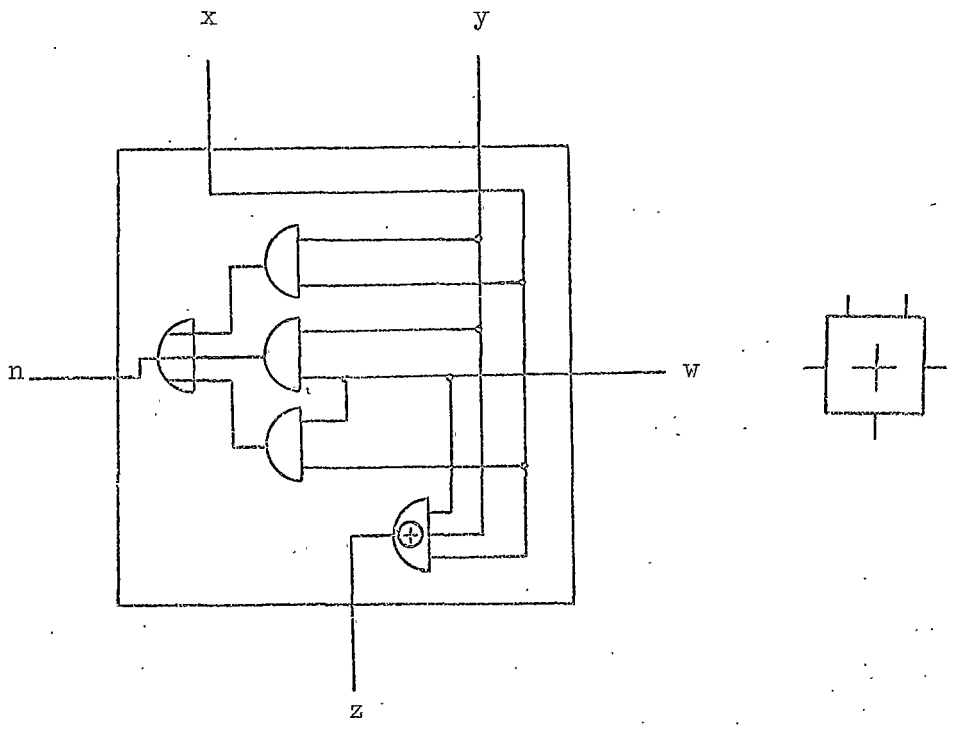


Figure 3.11. The Adder Cell

When used in other figures the cell will be represented as in Figure 3.11(b).

The cell can be used to form a parallel ripple-carry adder for obtaining the sum Z of two N-bit binary numbers X and Y. N cells are connected in a cascade with parallel inputs x_i from the binary number X and y_i from the binary number Y, as shown in Figure 3.12. The outputs z_i represent a binary number Z whose value is the sum of the input numbers X and Y. The w input to the low-order cell of the cascade is supplied with a 0.

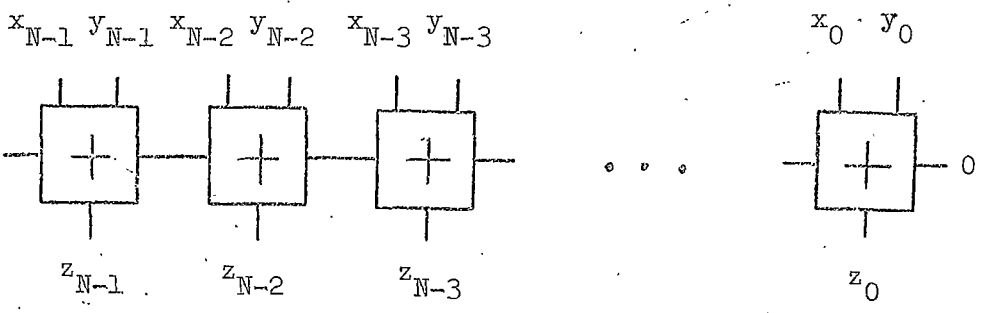


Figure 3.12. The Adder Cascade

3.3.7 Adder-Subtractor Cell

The adder-subtractor cell shown in Figure 3.13(a) is a four-input, two-output cell which produces the functions

$$z = x \oplus y \oplus w$$

and

$$n = y \cdot w + (x \oplus g) \cdot (w + y).$$

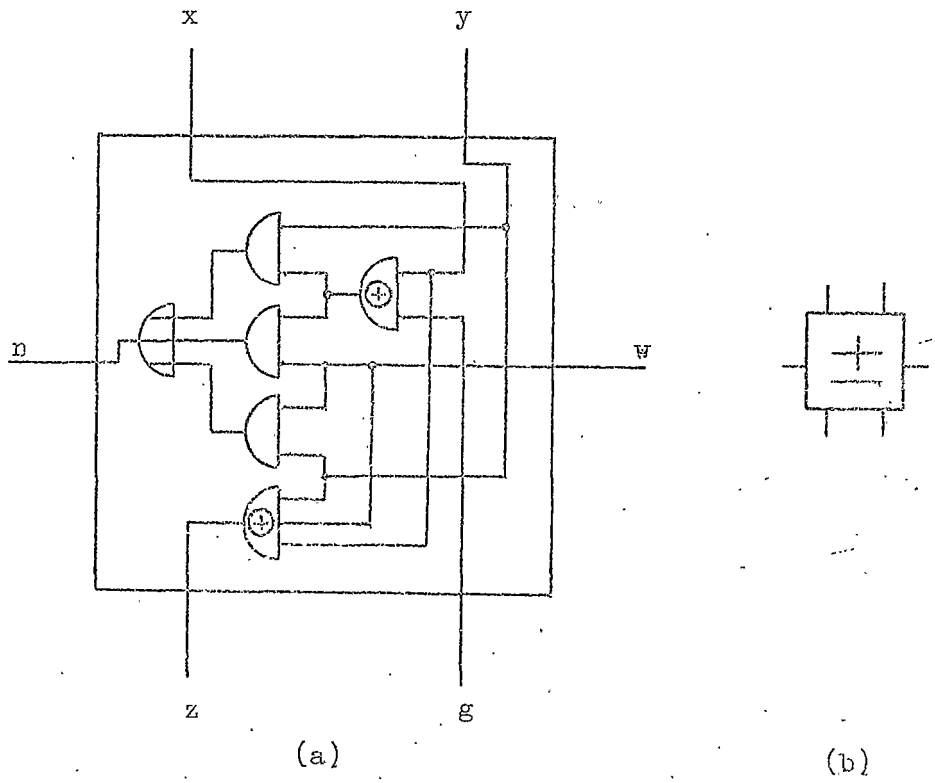


Figure 3.13. The Adder-Subtractor Cell

When used in other figures the cell will be represented as in Figure 3.13(b).

The cell can be used to form a parallel ripple-carry adder-subtractor for obtaining the sum or difference of two N-bit binary numbers X and Y. N cells are connected in a cascade with parallel inputs x_i from the binary number X and y_i from the binary number Y as shown in Figure 3.14.

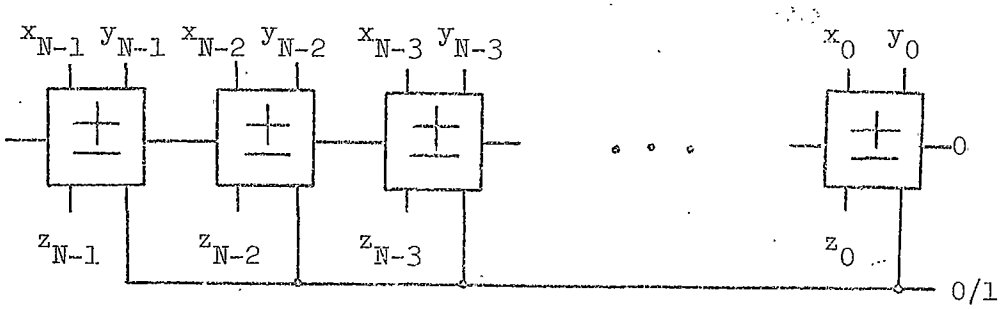


Figure 3.14. The Adder-Subtractor Cascade

The w input to the low-order cell is supplied with a 0. If the g input of each cell is supplied with a 0, the z_i outputs will represent a binary number Z whose value is the sum of the input numbers X and Y. If the g input of each cell is supplied with a 1; the z_i outputs will represent the difference of the input numbers X and Y. That is, X - Y.

3.3.8 General Register Cell

The shift and other registers can be thought of as being made of a cascade of identical cells. Such a cell is shown in Figure 3.15.

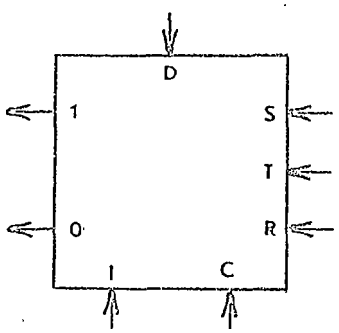


Figure 3.15. The General Register Cell

The cell contains a one-bit register whose contents and its complement appear on lines 1 and 0 respectively. An input on the toggle line T causes the register to switch to 1 if it was 0 and to 0 if it was 1. The switch occurs on the transition of the input from a 0 to a 1.

An input on the clock line C causes the data to be transferred into the register from the set line S and the reset line R. The change occurs on the transition of the input C from a 1 to a 0. Thus, a string of cells connected as shown in Figure 3.16 acts as a shift register when the C input is applied.

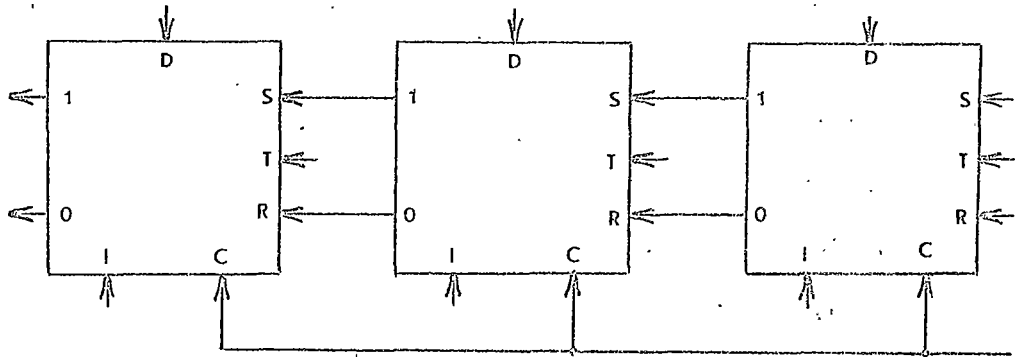
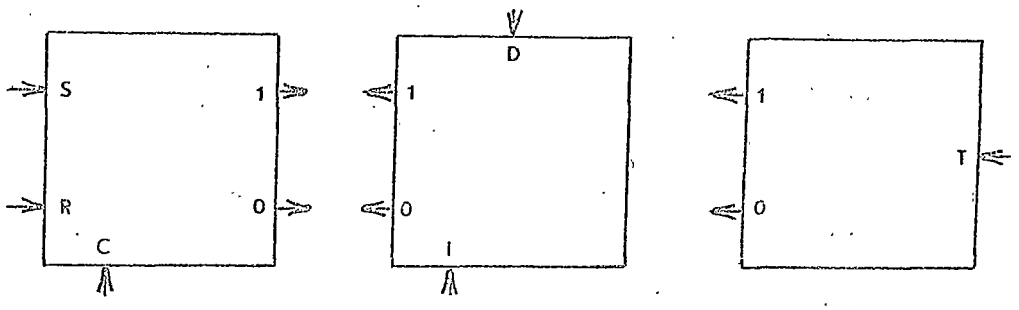


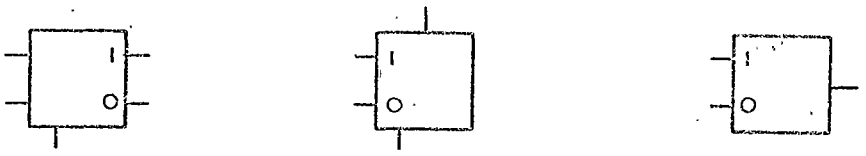
Figure 3.16. A Shift Register Cascade

An input on the input line I causes the register to be set to the value of the data line D. This input is used to transfer data in parallel into a string of cells.

Not all registers will require cells with the capability of the general register cell. However, this model serves to represent all of the types of cells required. When the general cell is used in a register, only those inputs or outputs required for that particular use will be shown. A convention will be adopted in order to avoid the necessity of labeling all inputs and outputs for every use: the outputs will be marked as 1 and 0 and the inputs will be identified by their position with respect to the outputs. Some examples of cells in which not all inputs and outputs are used are given in Figure 3.17(a); their corresponding equivalent forms are given in Figure 3.17(b).



(a)



(b)

Figure 3.17. Special Forms of General Register Cell

It should be noted that the register cell outputs may be directed either to the left or to the right. In either case the I input line is the line closest to the output side and the C line is the line closest to the side with the S, R and T inputs.

3.4 Array Interconnection Structure

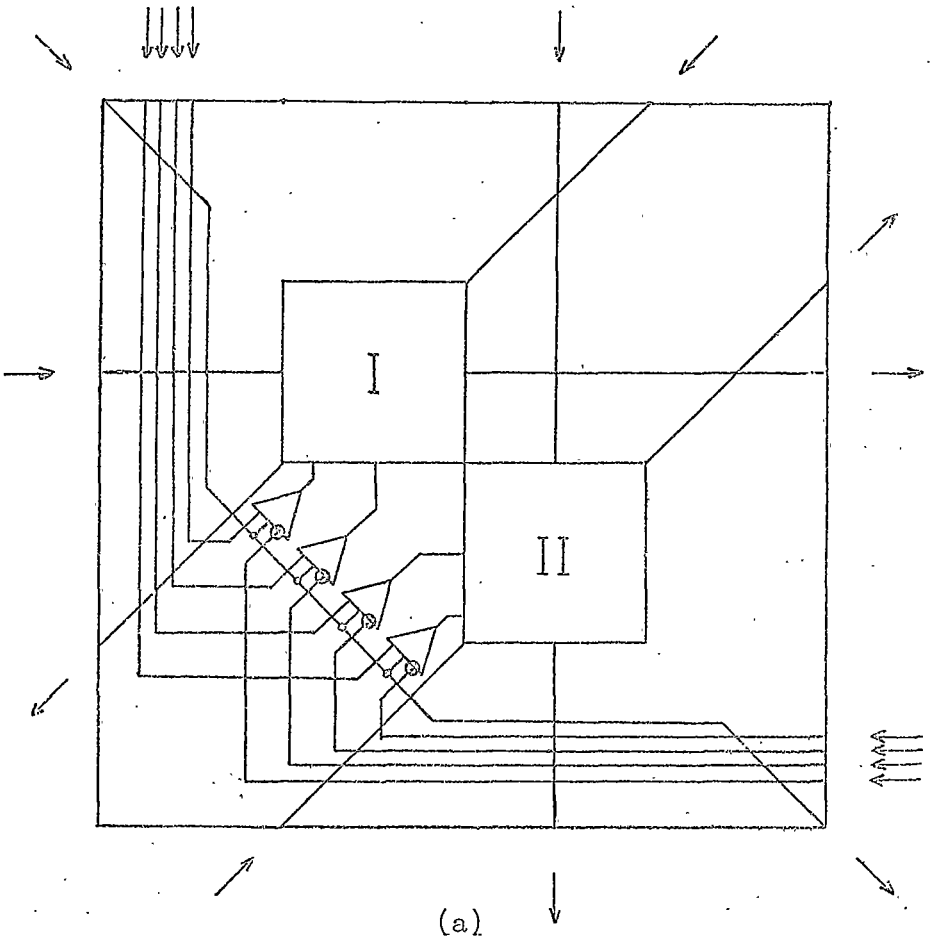
In order to describe the array interconnection structure for the proposed KF machine, it will be helpful first to consider just the routing portion of the cells. An equivalent treatment is to develop

the interconnection structure for cells with just a routing capability.

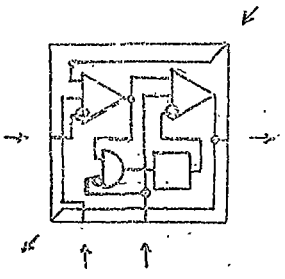
Such a routing cell is shown in Figure 3.18(a). The cell consists of two inner register cells I shown in Figure 3.18(b) and II shown in Figure 3.18(c) and some control logic. The inner cells I and II each have two data inputs, two data outputs and two control inputs. The data output lines are tied together internally so there is really only one output for each inner cell. Inner cells I have data inputs from corresponding inner cells of the routing cells west and northeast of their routing cell. Inner cells II have data inputs from the corresponding inner cells of the routing cells north and southwest of their routing cell.

One of the control lines into each of the inner cells selects the data line to be used as input to that cell. The other line selects whether the incoming data is to be routed to the register within the cell or around it. If it is routed to the register, the output of the cell corresponds to the low-order bit of the register and the input goes to the high-order bit of the register. If the data is routed around the register, the output of the cell corresponds to the data on the selected input line.

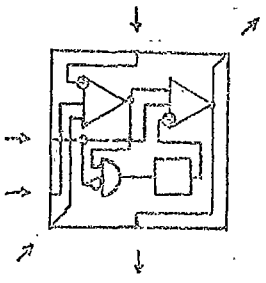
The routing cell has two sets of control line inputs for the inner cells: the set entering the cell from the top comes from the row control bus and the set entering the cell from the right comes from the column control bus. The control line passing through the cell diagonally from the upper-left to the lower-right selects which set of



(a)



(b) I Inner Cell



(c) II Inner Cell

Figure 3.18. The Routing Cell

control inputs the cell is to use. The column control lines are used if this line is 0, the row control lines are used if it is 1. This control goes to all cells in the array so either all cells are using row control lines or all cells are using column control lines. The arrangement of control lines is shown for a 4x4 array in Figure 3.19.

Each cell is capable of 16 different routing configurations depending on the values of the selected control lines. These configurations are shown in Figure 3.20. Cells in which the register is bypassed are shown with the input passing through to the output. Only the output line corresponding to the selected input line is shown although the same output would appear on the other output line as well.

Some of the basic routing operations can now be demonstrated using 4x4 arrays of the cells just described. No particular interconnection pattern is assumed at this time. When a sufficient set of basic routing patterns has been developed an interconnection pattern incorporating all of these can be constructed.

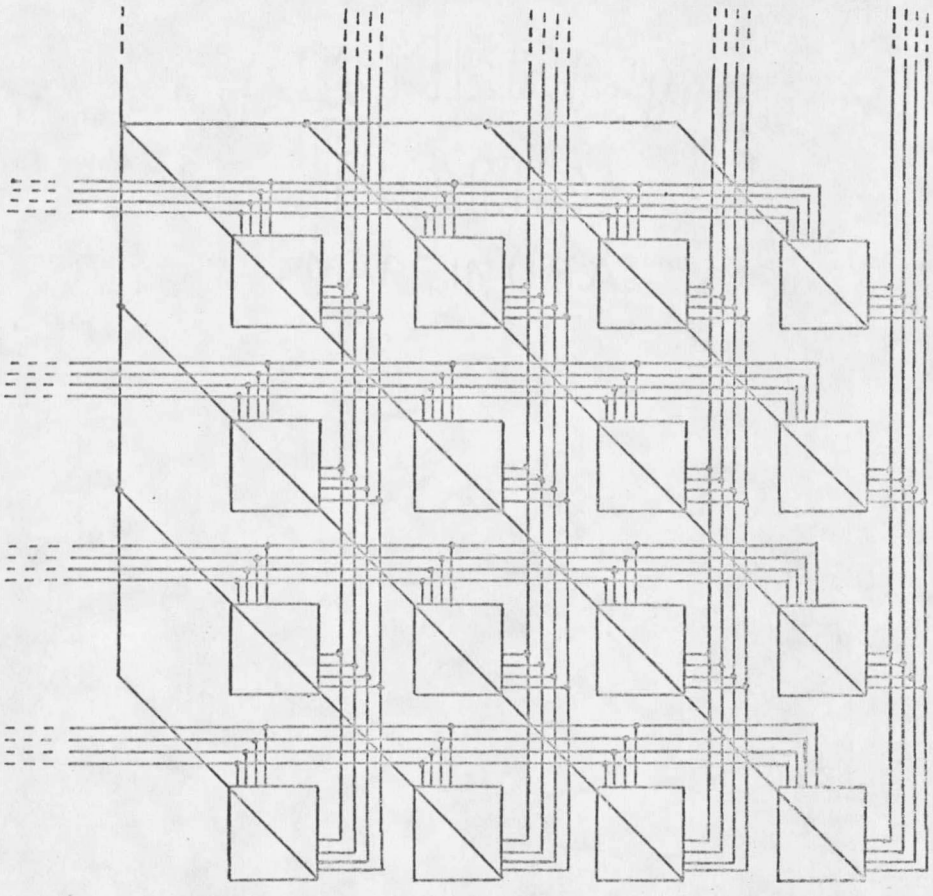


Figure 3.19. Routing Array Control-Line Interconnection

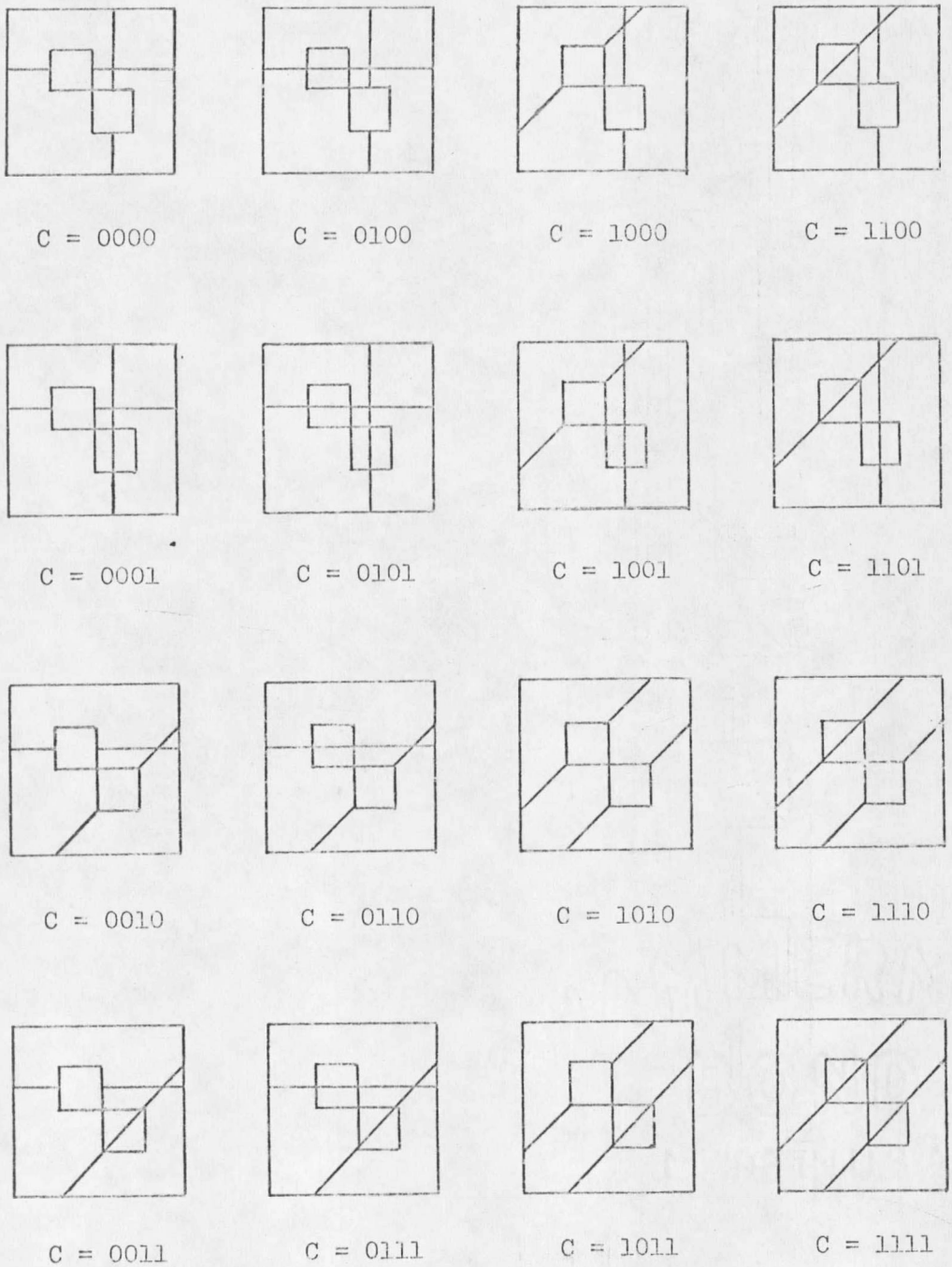


Figure 3.20. Possible Routing Cell Configurations

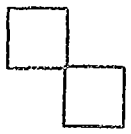


Figure 3.21. A Routing Cell Representation

The routing cells are drawn as shown in Figure 3.21 which represents the two inner cells of Figure 3.18(a). Control lines are not shown. For all of the interconnection patterns the registers are connected together to form long shift-registers. There is assumed to be one element of an array of data stored in each of the inner cells. The data is shifted from cell to cell along the interconnection paths.

Figure 3.22 shows an interconnection pattern for rotating the data array, stored in the registers of the I inner cells, to the right. The outputs of the right edge cells connect to the inputs of the left edge cells forming a cylindrical interconnection.

The interconnection pattern of Figure 3.23 is similar to that of Figure 3.22 except that cells in the second and fourth columns are bypassed. Thus only the first and third columns take an active part in the routing process. This scheme could be used to perform a column interchange operation of a sub-array rotation.

Figure 3.24 shows an interconnection pattern for rotating the data array, stored in the II inner cells, down. The outputs of the bottom

edge cells connect to the inputs of the top edge cells forming a cylindrical interconnection as before. Also as before some of the rows of cells may be by-passed; however, this is not shown in the figure.

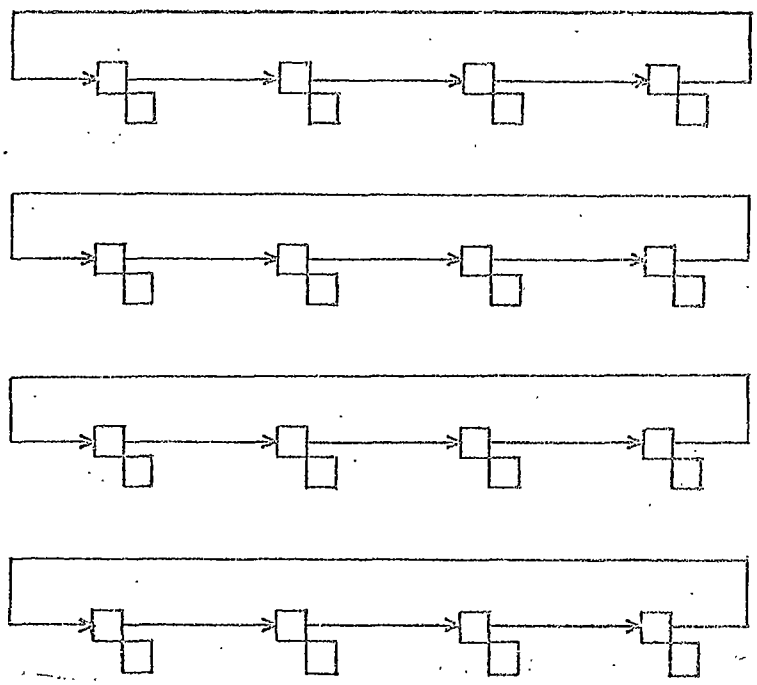
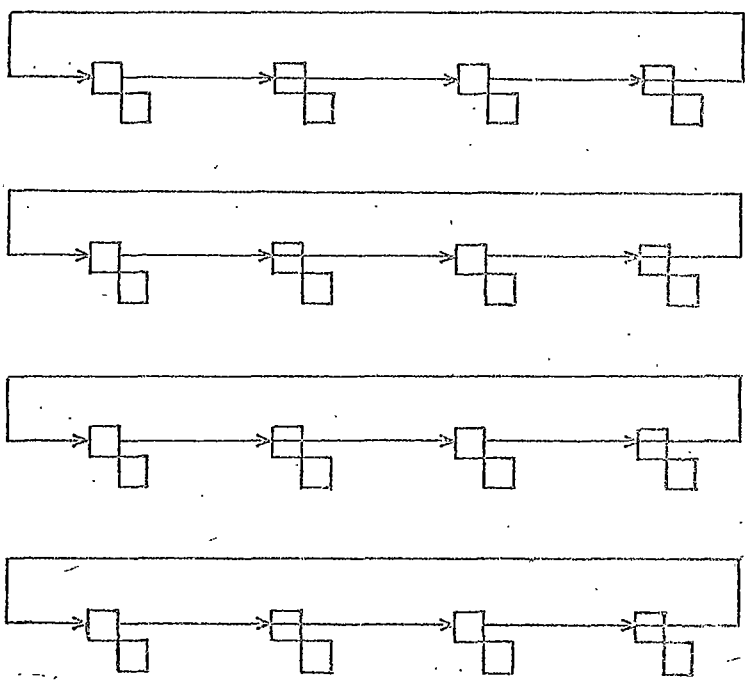


Figure 3.22. Rotate-Right Interconnection



Fogire 3.23. Interchange-Column Interconnection

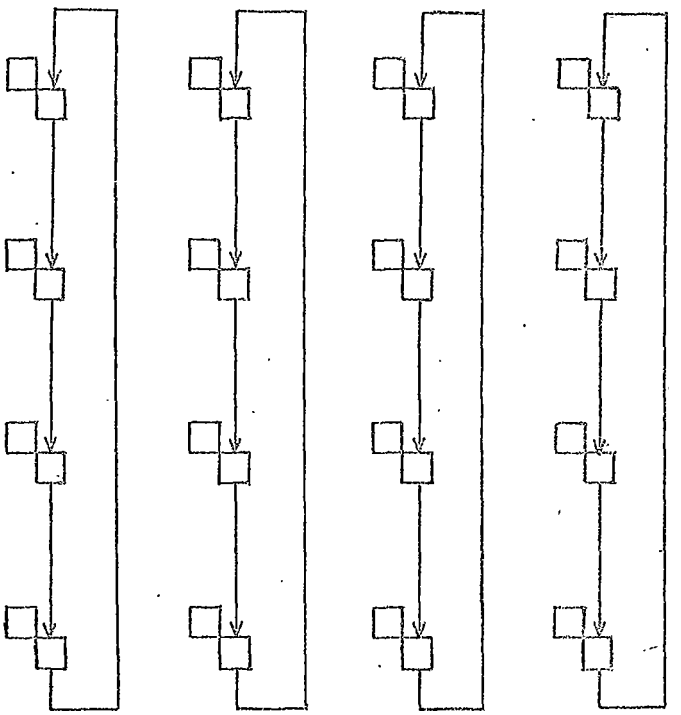


Figure 3.24. Rotate-Down Interconnection.

Figure 3.25 shows in interconnection pattern for replacing the data array, stored in the registers of the II inner cells (Figure 3.18) by a skewed-up version of the data array stored in the registers of the I inner cells.

Similarly Figure 3.26 shows an interconnection pattern for replacing the data array, stored in the registers of the I inner cells by a skewed-left version of the data array stored in the registers of the II inner cells.

Finally, Figure 3.27 shows an interconnection pattern for replacing the data array stored in the registers of the I inner cells by the transpose of the data array stored in the registers of the II inner cells.

It should be noted that in some cases more than one type of routing may occur at once. For example it would be possible to rotate the data array, stored in the registers of the I inner cells, to the right and at the same time rotate the data array, stored in the registers of the II inner cells, down.

A 4x4 routing array incorporating all of the interconnection patterns just discussed is shown in Figure 3.28. The array uses routing cells as shown in Figure 3.18.

Line-select gates along the left edge of the array are controlled by control line f; these gates are used to direct data to the west input of the I inner cells of the left edge routing cells. The data comes from either the external row inputs b_i or from the right-edge line-select gates.

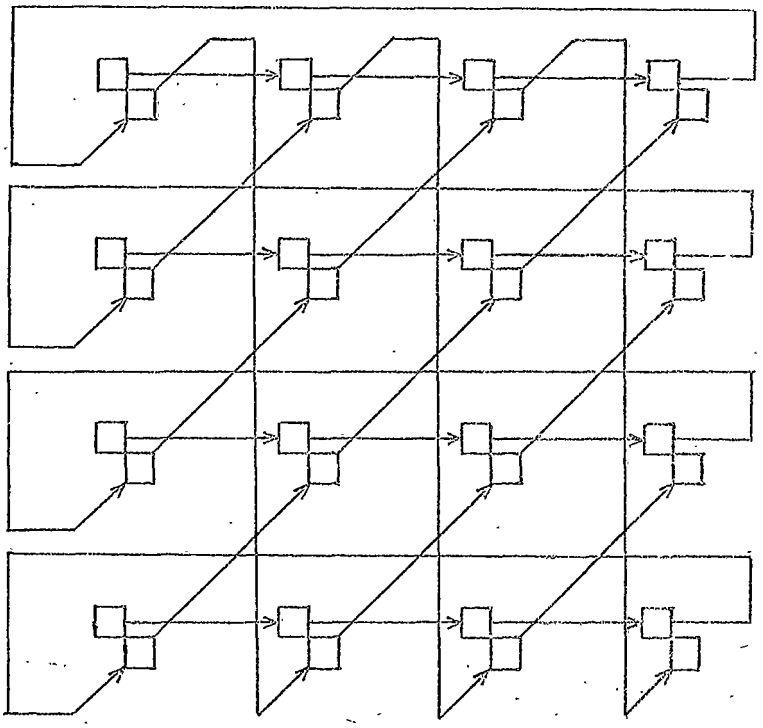


Figure 3.25. Skew-Up Interconnection

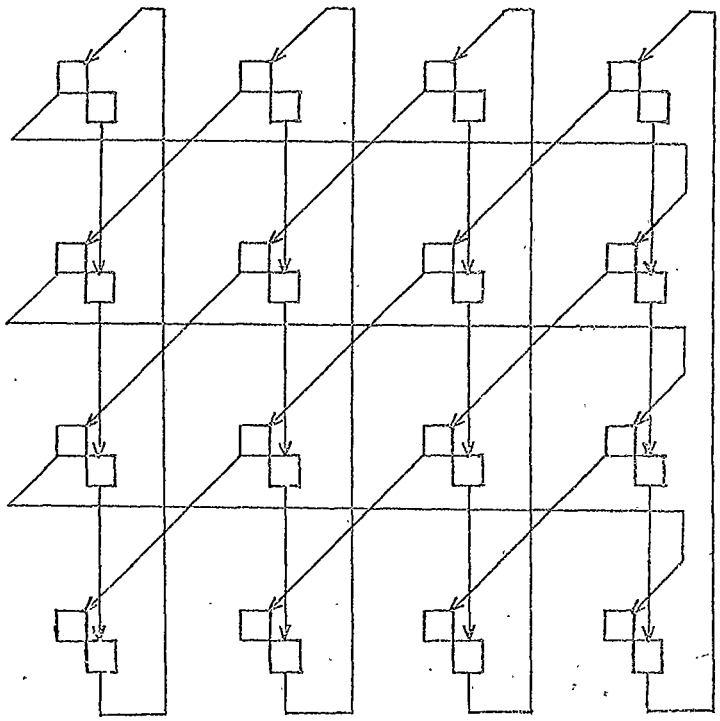


Figure 3.26. Skew-Left Interconnection

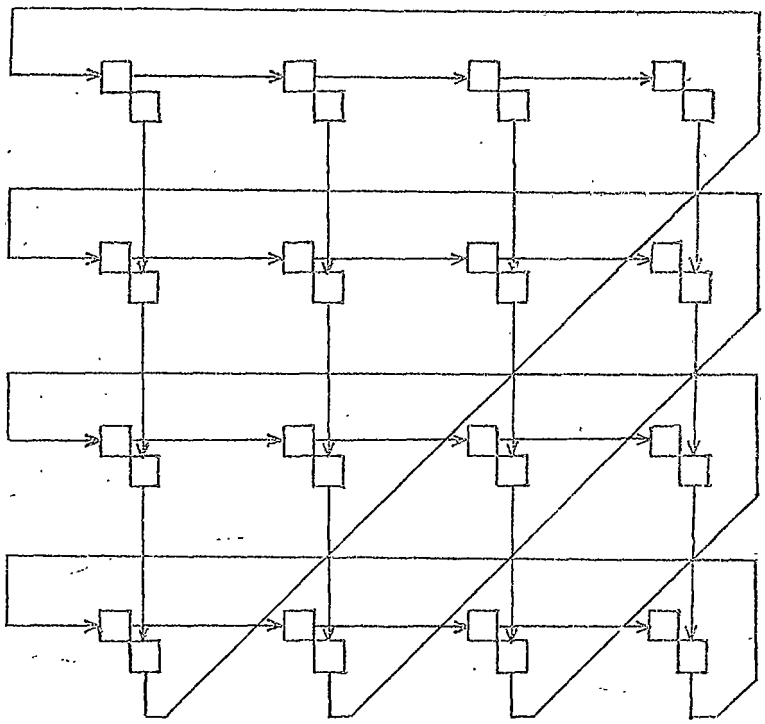


Figure 3.27. Transpose Interconnection

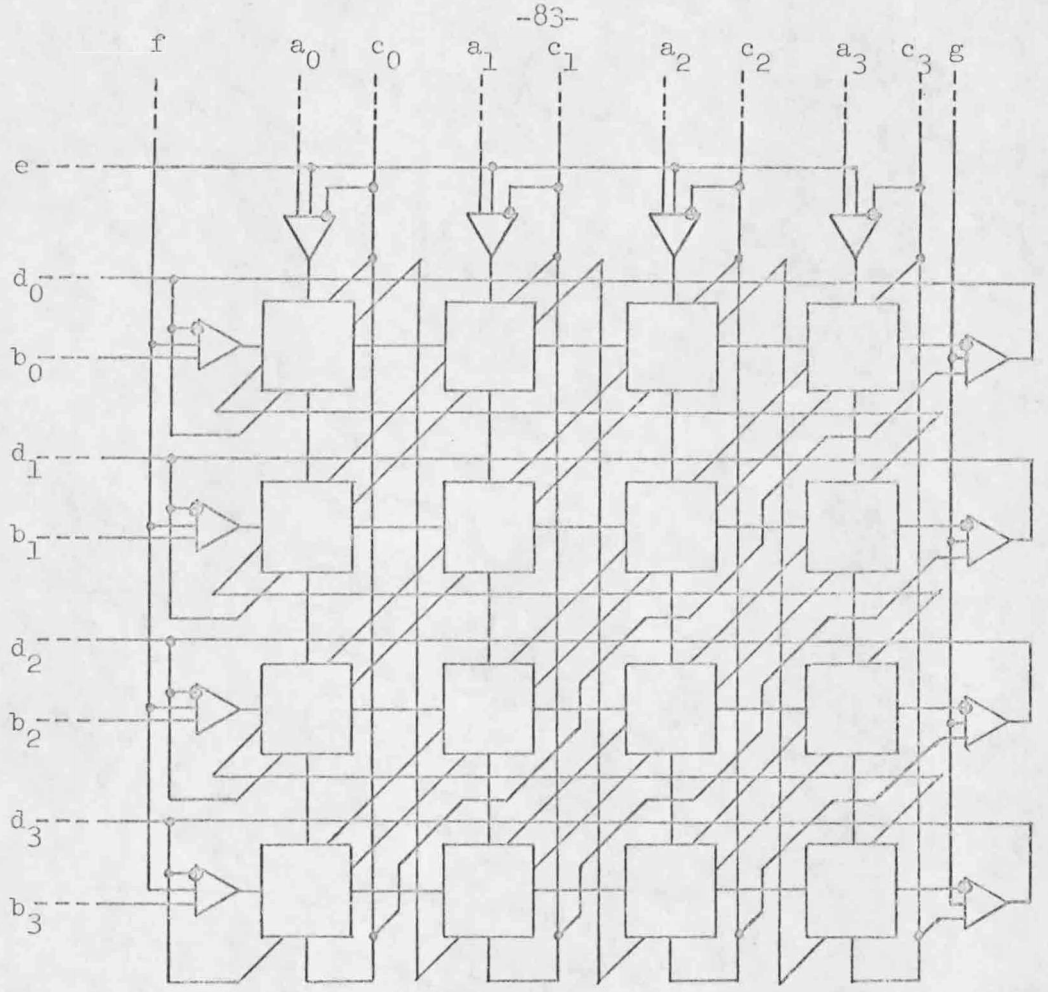


Figure 3.28. Routing Array Interconnection

Line-select gates along the right edge of the array of Figure 3.28 are controlled by control line g, and these gates are used to direct data to lines which go to the external row outputs d_i, to the left-edge line-select gates and to the south-east input of the II inner cells of the left-edge routing cells. The data comes either from the outputs of the I inner cells of the right-edge routing cells or from the outputs of the II inner cells of the bottom-edge routing cells.

Line-select gates along the top edge of the array of Figure 3.28 are controlled by control line e; these gates are used to direct data to the north inputs of the II inner cells of the top-edge routing cells. The data comes either from the external column inputs a_i or from the outputs of the II inner cells of the bottom-edge routing cells.

The outputs from the II inner cells of the bottom-edge routing cells also connect to the right-edge line-select gates, to the northeast inputs to the I inner cells of the top-edge routing cells and to the external column outputs c_i.

Within the array the outputs of the I inner cells connect to the northeast inputs of I inner cells located southwest of the given cell and to the west input of the I inner cells located east of the given cell. The outputs of the II inner cells connect to the southwest inputs of II inner cells located northeast of the given cell and to the north input of II inner cells located south of the given cell.

On the left edge of the array in Figure 3.28 the southwest outputs of the I inner cells are connected to the northeast inputs

of the I inner cells on the right edge of the array and down one row from the given cell. An exception is the lower-left cell of the array; the output of this cell is not used. The northeast outputs of the II inner cells on the top edge of the array are connected to the southwest inputs of II inner cells on the bottom edge of the array and right one column from the given cell. An exception is the upper-right cell of the array, which has an unused output.

The southwest outputs of the I inner cells on the bottom edge of the array are not used and the south outputs of the II inner cells are directed to the top and right line-select gates and to the top northeast inputs as mentioned before.

On the right edge of the array the northeast outputs of the II inner cells are not used and the east outputs of the I inner cells are directed to the right line-select gates as mentioned before.

3.5 Data Representation and Arithmetic

Before describing the structure of a processing cell it will be helpful to discuss the form in which numbers are represented and how the arithmetic is handled in the cells.

3.5.1 Data Representation

The data are represented in floating-point form. A 32-bit floating-point number consists of a sign (bit 0), a 24-bit binary proper fraction (bits 1-24) and an excess-64 base-2 exponent (bits 25-31). The binary point is assumed to precede bit 1. The floating-point format is given by Figure 3.29

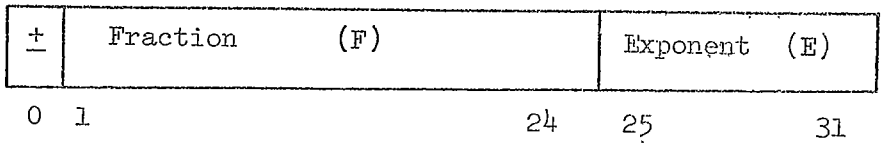


Figure 3.29. Floating-Point Format

where the value N of the number is given by

$$N = F \times 2^{E-64} \quad \text{if the sign is 0}$$

or

$$N = -(1 - F) \times 2^{E-64} \quad \text{if the sign is 1.}$$

For negative numbers the sign bit is 1 and the fraction is in two's complemented form.

The exponent is excess-64. That is, the exponent is represented by the binary equivalent of the exponent value plus 64. So for example, if the exponent field considered as a 7-bit binary integer is zero, then the fraction is to be multiplied by $2^{0-64} = 2^{-64}$. If the exponent field is 127 then the fraction is to be multiplied by

$2^{127-64} = 2^{63}$. The exponent field is not complemented for negative numbers as is the fraction.

A number is considered to be normalized if: the sign bit is 0 and either bit 1 or bit 2 is a 1, or the sign bit is 1 and either bit 1 or bit 2 is a 0. This method of normalizing numbers is used rather than strict base-2 normalization so that numbers may be normalized regardless of their sign. It does, however allow for more than one representation for some numbers. For example, 1.0 may be represented by either

0.10000000000000000000000001000001

or 0.01000000000000000000000001000010.

This floating point representation makes direct comparison of two numbers somewhat more difficult than some other representations. However, the circuit requirements for implementing this representation seem to be less that for others.

3.5.2 Arithmetic

In this section methods of doing arithmetic with the assumed form of floating-point numbers are developed. For clarity the 8-bit representation will be used.

As an example of the addition process the sum A + B will be found, A = 2.5 and B = -1.25. In the 8-bit form with an extra sign bit A and B are represented by

A = 00 .1010 110

and B = 11 .0110 101.

The first step in the addition is to align the fractions. To do this compare the A exponent (A_e) to the B exponent (B_e). Since B_e is smaller shift the B fraction (B_f) right 1 bit position, extending the sign and increment B_e by 1 giving

$$A_f = 00\ .1010 \qquad A_e = 110$$

$$B_f = 11\ .0110 \qquad B_e = 101$$

Since A_e is now equal to B_e the fractions are aligned for addition. The sum of the fractions is given by

$$C_f = 0\ .00101$$

where the assumed binary point is now one bit to the left of its original position. This is to allow for possible overflows which may occur in the addition. A 1 is added to the exponent to account for the change in position of the assumed binary point. So the exponent is now

$$C_e = 111.$$

The result can now be normalized. To normalize, check for the normalize condition. It is not satisfied yet so shift the fraction C_f left and decrement the exponent C_e to give

$$C_f = 0\ .01010 \qquad C_e = 110.$$

Therefore C is 1.25.

As an example of multiplication, the product A x B will be found, where A = -1.5 and B = 1.5. Again the 8-bit representation will be used. The numbers are given by

$$A_f = 1.0100 \qquad A_e = 101$$

and $$B_f = 00000.1100 \qquad B_e = 101.$$

A double length register is used for B_f so its sign bit is extended left 4 bits.

The fraction portion C_f of the product C register is cleared to 0. The exponent portion C_e is set to the sum of A_e and B_e with the high-order bit complemented to account for the excess-4 exponent representation giving

$$C_e = 110.$$

The fraction multiplication now consists of adding B_f to C_f if the low-order bit of A_f is 1, shifting A_f right 1 bit, extending the sign bit, and shifting B_f left 1 bit, filling with 0's on the right. If the low-order bit of A_f is 0 the process is the same except that the addition is not performed. This process is repeated once for each bit of the B_f register, (nine times for this case) to produce a correct product regardless of the sign of A or B. These steps are shown as Table 3.1.

Table 3.1. Multiplication Steps

A_f = 10100	C_f = 000000000
	B_f = 000001100
A_f = 11010	C_f = 000000000
	B_f = 000011000
A_f = 11101	C_f = 000000000
	B_f = 000110000
A_f = 11110	C_f = 000110000
	B_f = 001100000
A_f = 11111	C_f = 000110000
	B_f = 011000000
A_f = 11111	C_f = 011110000
	B_f = 110000000
A_f = 11111	C_f = 001110000
	B_f = 100000000
A_f = 11111	C_f = 101110000
	B_f = 000000000
A_f = 11111	C_f = 101110000
	B_f = 000000000

The normalize process is the same as that for addition giving

$$C_f = 1\ .0111 \qquad C_e = 110$$

so C = -2.25.

Other multiplication methods perform the multiplication in fewer add cycles but require correction of the results depending on the signs of the operands.

3.6 Processor-Cell Structure

The processor cell shown in Figure 3.30 consists of the cell control unit labeled C, the memory address decoder D, the memory bit-storage section B, the cell selector S, the cell routing unit R, the cell adder-subtractor A and the cell multiplier M. The only data or control lines shown entering or leaving the cell are those associated with the cell routing logic.

There are four address lines that connect the cell control to the address decoder. Sixteen word-select lines connect the memory address decoder to the memory bit-storage section. The store line in Figure 3.30 connects the cell control to the bit-storage section. Thirty-two read lines interconnect the bit-storage section and the cell selector, the cell routing unit, the cell adder-subtractor and the cell multiplier. Thirty-two write lines are used in Figure 3.30 to connect the cell selector to the bit-storage section of the memory. The cell adder-subtractor and the cell multiplier are each connected to the cell selector with 32 lines, while the cell routing unit is connected to the cell selector with two sets of 32 lines each.

Two selector control lines connect the cell control to the cell selector. The routing unit, adder-subtractor and multiplier each have several control and data lines tying them to the cell control unit.

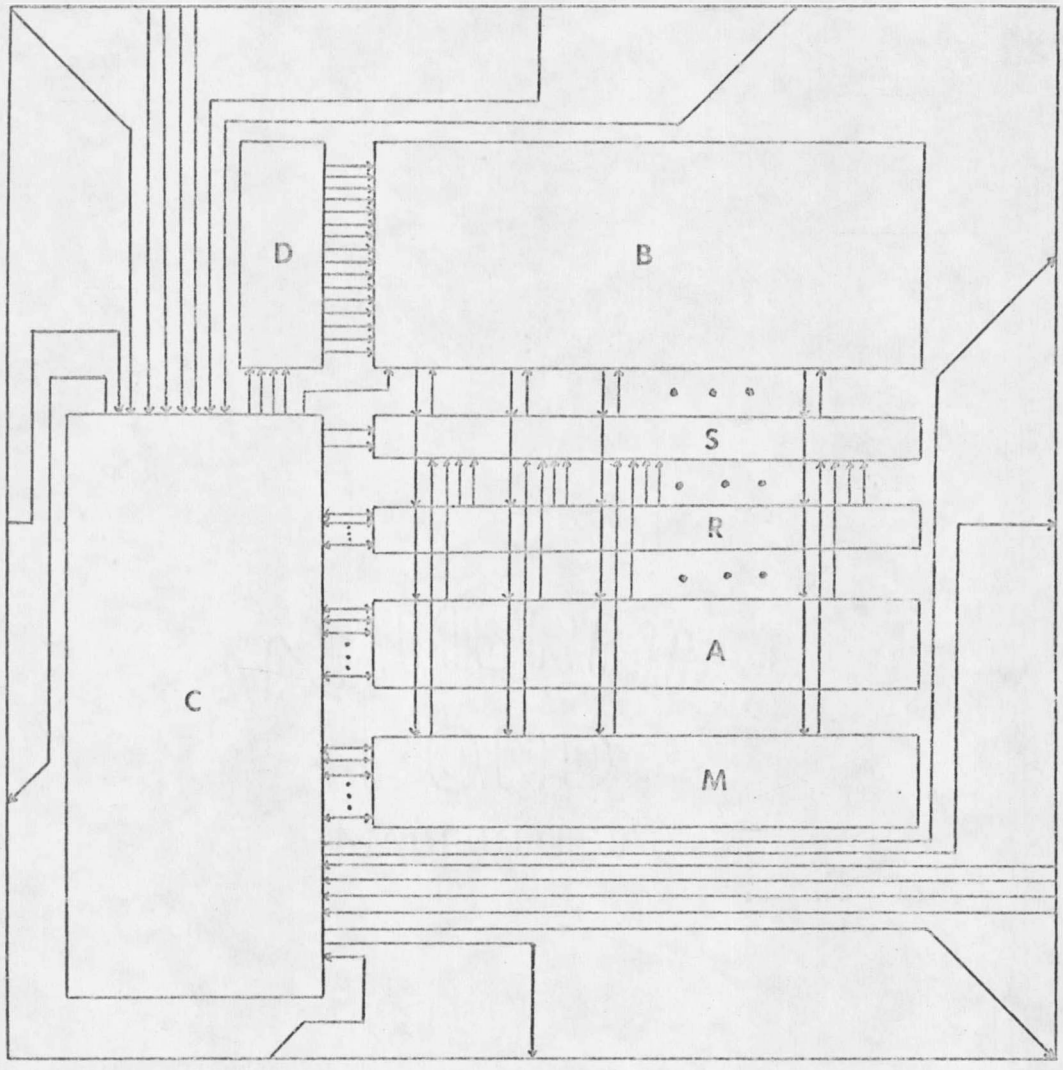


Figure 3.30. Structure of Processor Cell

3.7 Cell Memory Unit

The cell memory unit, shown in Figure 3.31, consists of two sections, indicated by D and B in Figure 3.30. The section on the left of Figure 3.31 is an address decoder while the section on the right of the figure is the actual bit-storage portion of the memory.

The address decoder section of the memory has four inputs and 16 outputs corresponding to the $2^4 = 16$ possible input combinations. Only one output line can be true at a time. For example if the address lines a_0-a_3 are set to 1011, output line 11 (actually the 12th line) will be set to 1; all others will be set to 0.

The bit-storage section of the memory consists of an array of identical cells of the type shown in Figure 3.32 arranged in 16 rows of 32 cells each and interconnected as shown in Figure 3.31. The 16 input lines to the array from the address decoder serve as word-select lines. Data is read from the selected word in parallel on the 32 r_i read lines. Since one word will always be selected, that word will be available on the read lines at all times. Thus, no action other than setting up the appropriate address is required to access and read a word. Data is written into the selected word in parallel from the 32 w_i write lines. The content of the selected word is not changed until a signal is supplied on the store lines. Thus, to access and write a word the appropriate address must be set up, data to be written supplied to the write lines and a signal supplied to the store line.

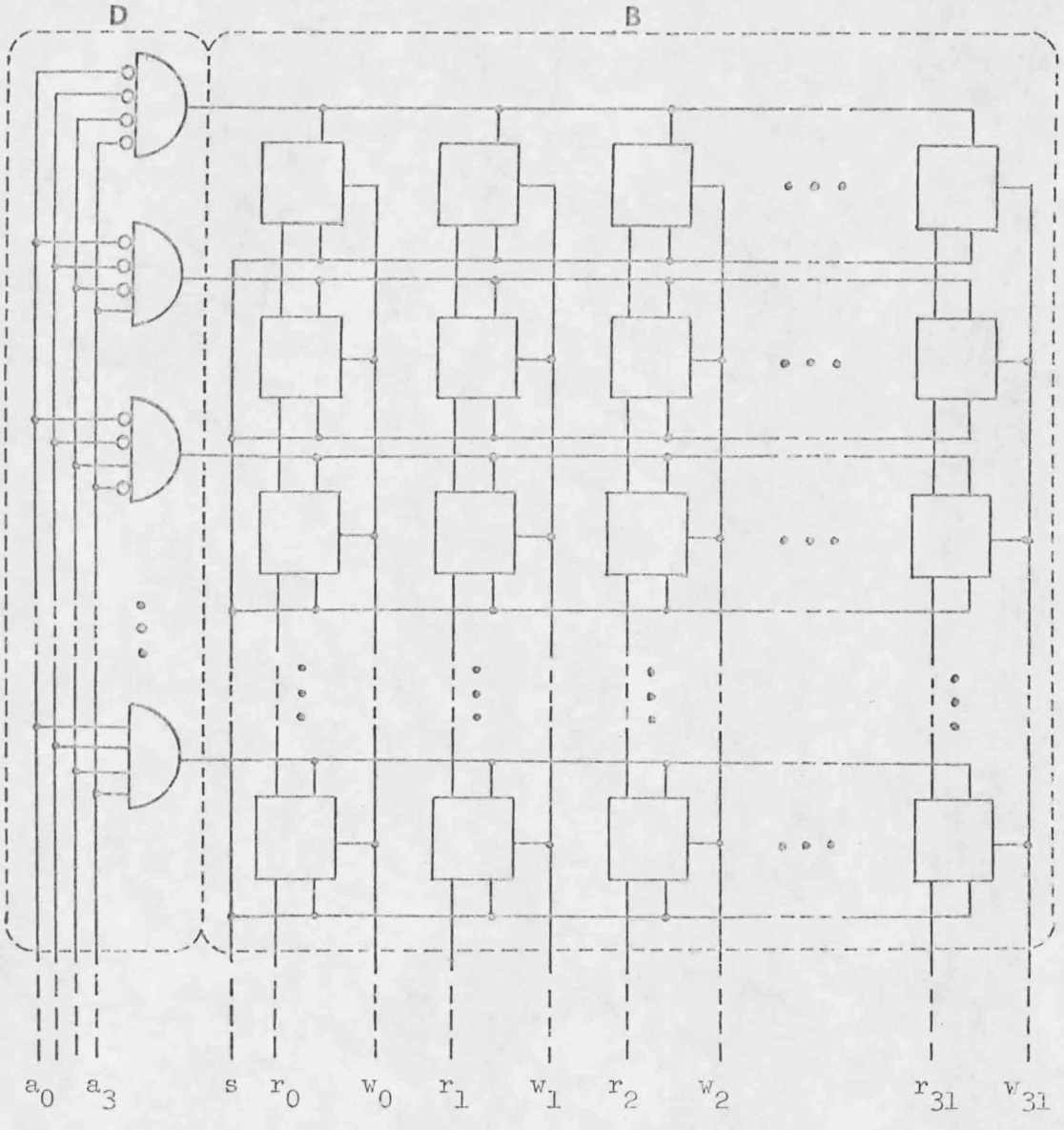


Figure 3.31, Cell Memory Unit

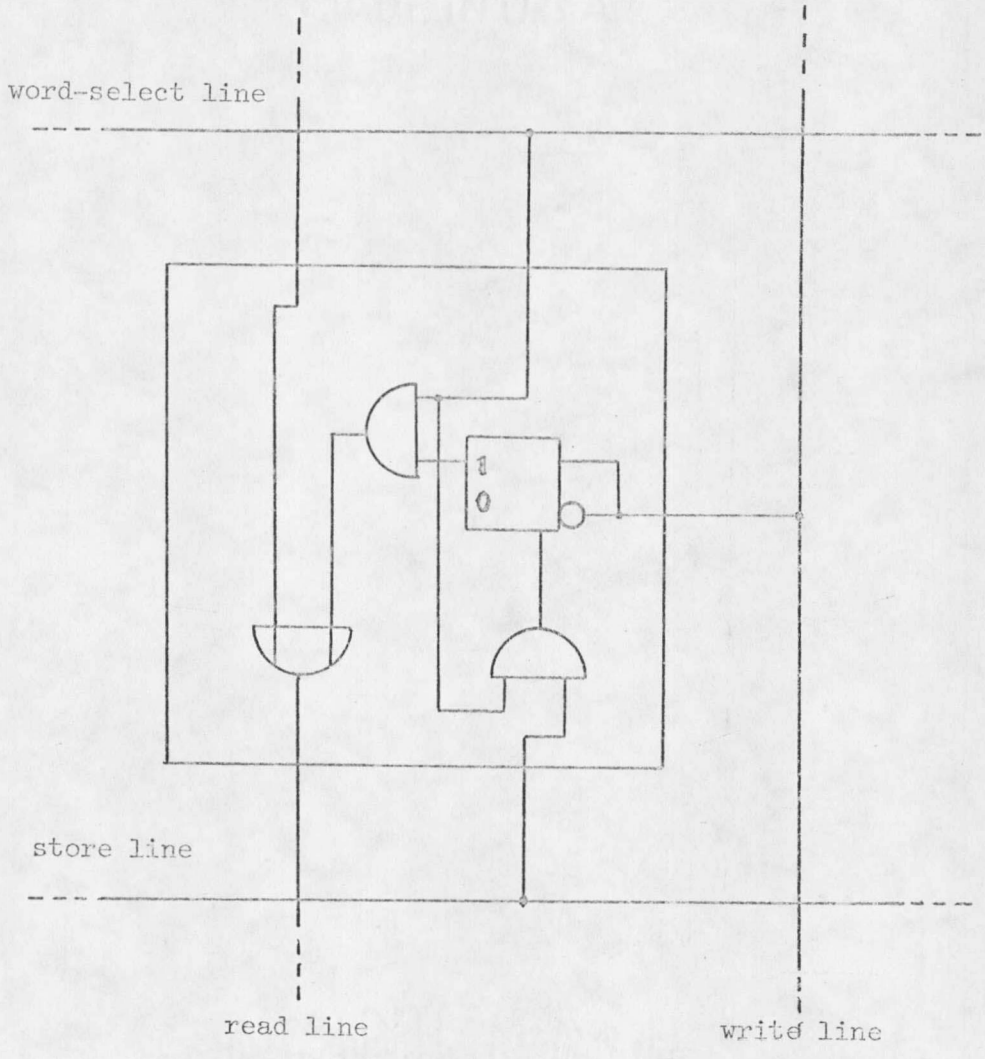
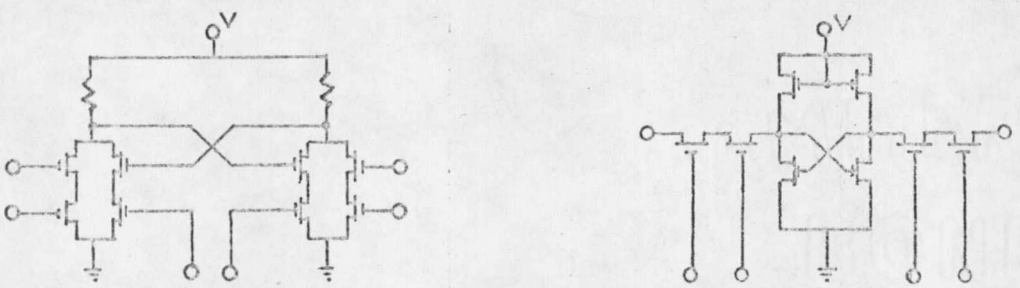


Figure 3.32. Bit-Storage Cell

As an example suppose that word 11 is selected. The contents of word 11 then appear on the read lines. If a signal is supplied to the store line s the contents of word 11 are replaced by the data supplied to the write lines. No other words are affected and the new contents of word 11 now appear on the read lines.

It would be possible to operate the memory serially by using shift-registers for data storage. However the circuitry which may be saved by doing this does not seem worth the extra time which would be required to transfer a word of data.

Figure 3.33 shows some typical MOS circuits [18,20] for implementing (a) a shift-register stage and (b) a bit-storage cell.



(a) Shift-Register Stage (b) Bit-Storage Cell

Figure 3.33. Storage Cells

It can be seen that there is very little difference in the number of active elements per cell for the two types of storage. However, the parallel data paths to and from the memory require more circuitry to direct than would a serial data path.

3.8 Cell Selector Unit

The cell selector unit S in Figure 3.34 consists of 96 line-select gates arranged as shown in Figure 3.34. The unit has 32 inputs from each of: the cell routing unit A register (ra_i), the cell routing unit B register (rb_i), the cell adder-subtractor C register (ac_i) and the cell multiplier C register (mc_i). Two control lines, sel_0 and sel_1, are used to select which of the four input sets is to be transferred to the output (w_i) according to Table 3.2.

TABLE 3.2. Truth Table For Cell Selector Control

sel_0	sel_1	w_i
0	0	ra_i
0	1	rb_i
1	0	ac_i
1	1	mc_i

The output lines w_i of the selector are used to drive the write lines of the cell memory unit. Thus the purpose of the selector unit is to select the register to be stored in memory for a store operation.

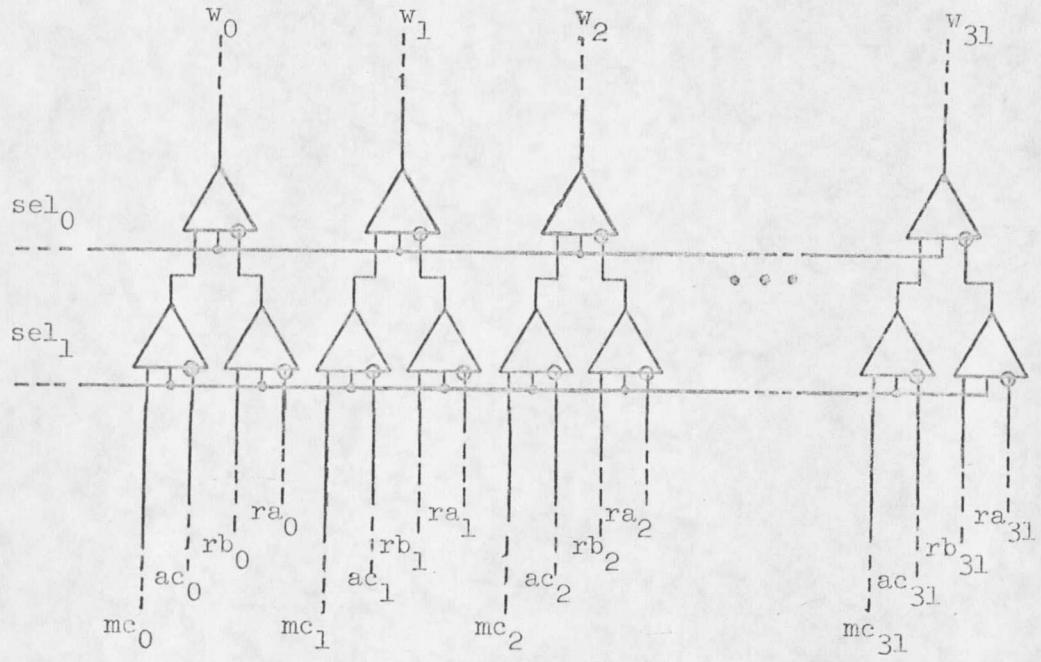


Figure 3.34. Cell Selector Unit

3.9 Cell Routing Unit

The cell routing unit shown as R in Figure 3.30 consists of two 32-bit registers connected as shown in Figure 3.35. The registers, called A and B, are capable of handling data in either a serial mode as shift-registers, or in a parallel mode as conventional registers. These registers correspond to the I and II inner cell registers referred to in the discussion of the array routing (Section 3.4, and Figure 3.18).

Parallel transfer of data into the registers of Figure 3.35 is controlled by the control lines "loada" and "loadb". Parallel data is made available to the registers on lines c_i and d_i which are both connected directly to the cell memory unit read lines r_i. The contents of the registers are available on lines a_i and b_i which are connected to corresponding ra_i and rb_i lines of the cell selector unit shown in Figure 3.34.

Serial transfer of data into the registers is controlled by the control lines "shifta" and "shiftb".

Associated with each register are two line-select gates. One of these is used to select the source of serial data to be supplied to the register. For register A it is either from the upper-right if control k_0 is 1 or from the left if control k_0 is 0. For register B it is either from the lower-left if control k_3 is 1 or from the top if control k_3 is 0. The other line-select gate associated with each register is used to determine if the data supplied to the output lines, a for register A and f for register B, is to come from the last bit of the register or from the selected input to the register. In the latter

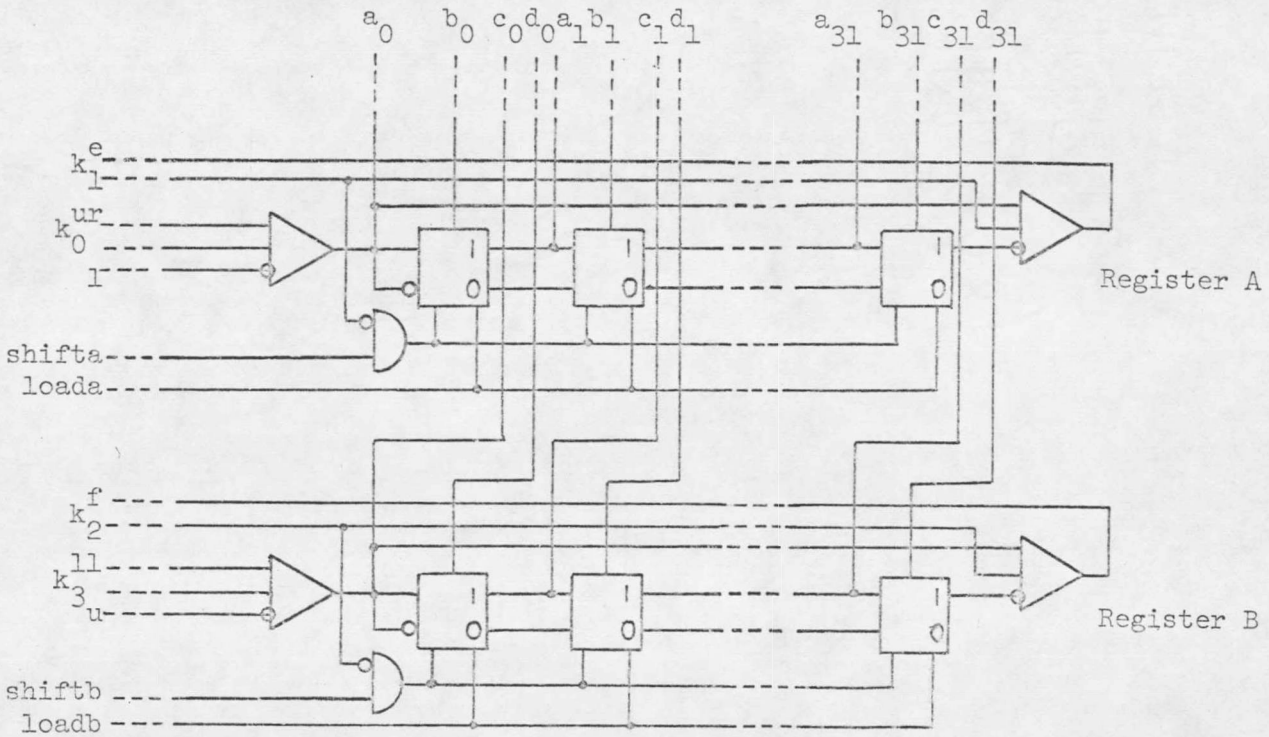


Figure 3.35. Cell Routing Unit

case the cell register is effectively by-passed. The cell routing unit is discussed in greater detail in Section 3.4, where the purposes of the features are described.

3.10 Cell Adder-Subtractor

The cell floating-point adder-subtractor shown in Figure 3.36 consists of an alignment section, an add-subtract section and a normalize section. This adder-subtractor corresponds to A of Figure 3.30.

While the adder-subtractor shown is for a 8-bit floating point numbers, the circuits extend in an obvious way to the 32-bit representation discussed. The 8-bit numbers are used to simplify the figures.

The 8-bit format used consists of a sign bit (bit 0), a 4-bit binary proper fraction (bits 1-4) and a 3-bit excess-4 base 2 exponent.

The purpose of the floating-point adder-subtractor is to produce the normalized sum (A + B) or difference (A-B) of two floating point numbers A and B.

The alignment section contains a 26-bit register (shown as 6-bit) for the sign and fraction portion of each of the two numbers (A and B) and a 7-bit register (shown as 3-bit) for the exponent portion of each of the two numbers. Two identical sign bits are used for each of the fraction registers. The alignment section is used to align the fractions of the two numbers before adding or subtracting. To accomplish this the alignment section uses a string of 7 comparator cells, previously discussed in Section 3.3.5, to act as a parallel comparator for comparing the exponent portions of A and B. Both inputs on the left end of the string are tied to 0. A clock signal (align) provided by the global control causes the smaller of the two exponents to be increased by one and its corresponding fraction to be shifted to the right one bit position. The sign bits do not receive the clock signal so the

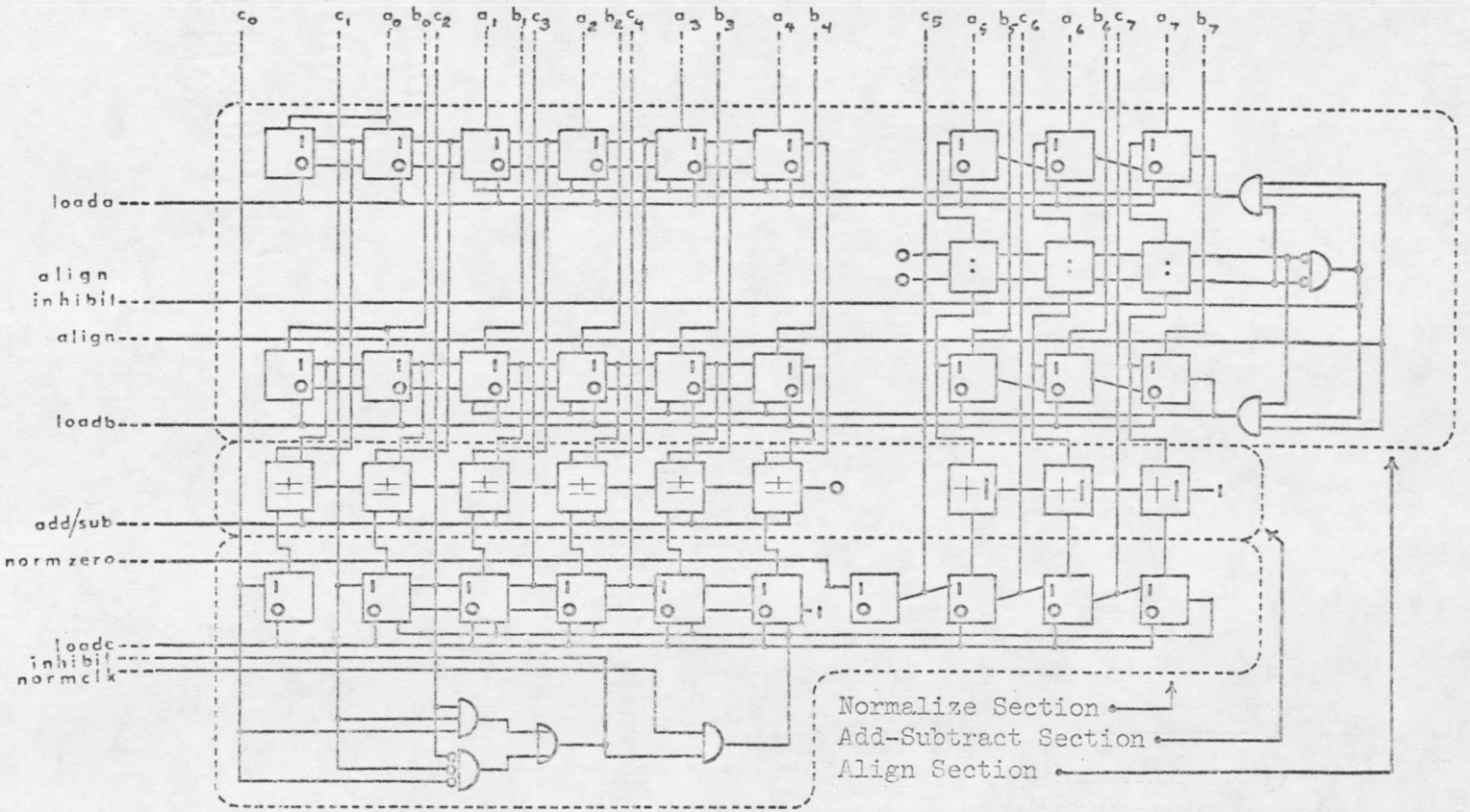


Figure 3.36. Cell Floating-Point Adder-Subtractor

sign of the fraction is extended in the shift process. The smaller exponent is selected by using the comparator output to inhibit the clock signal going to the register containing the larger exponent. The exponent registers are connected as ripple-through binary counters to accomplish the exponent incrementing. The fraction registers are connected as shift-registers to accomplish the shifting.

When the two exponents are equal the alignment clock will be inhibited from both registers by a signal from the comparator. The same signal that inhibits the clock can be used to indicate to the global control that the alignment is complete for that cell.

Data is loaded into the A or B registers in parallel from the 32 a_i or b_i lines when a signal is supplied to the "loada" or "loadb" lines, respectively. The sign bits are extended left to fill the high-order bit positions of the registers. Thus, there are two sign bits (bits 0 and 1) for each register.

The add-subtract section consists of a string of 26 of the add-subtract cells previously described and a string of seven of the add-one cells, also previously described. The adder-subtractor cells are connected as a parallel ripple-carry adder-subtractor. A 0 is supplied to the carry-borrow input of the rightmost cell. The adder-subtractor receives 26 parallel inputs from each of the A and B registers and has 26 outputs coming from the sum-difference outputs of the add-subtract cells. The additional bit on the left allows for possible overflow when the sum of the two fractions is greater than one.

The add-one cells are used to add a one to the exponent. The exponent from the B register is used since the A and B exponents must be the same after alignment. The exponent is increased by one because the output of the adder-subtractor is taken, shifted one bit to the left of the input to allow room for an overflow in the add-subtract process.

The normalize section of Figure 3.36 is used to normalize the result in the output register. This section consists of a 26-bit register connected as a shift-register for the fraction portion of the sum or difference, and an 8-bit register connected as a decrementing binary counter for the exponent portion of the sum or difference. The registers are loaded in parallel from the outputs of the add-subtract section when a signal is supplied on the "loadc" line.

To accomplish the normalization a clock signal provided by the global control causes the exponent to be decreased by one and the fraction to be shifted left one bit. The sign bit is not changed in the shift process. A check-for-normalize circuit detects when the contents of the register satisfy the conditions for a normalized number and inhibit the normalize clock. The inhibit signal can again be used to indicate to the global control that the normalization is complete for this cell.

The additional cell on the left of the exponent portion of the register is used to indicate the underflow condition which can occur when an attempt is made to normalize a number which has an all zero fraction. This additional cell is set to 0 when the exponent register

is loaded. The normalize-zero line of Figure 3.36 that is connected to this cell can be used to indicate the underflow condition to the global control.

Outputs from the 25 high-order bits of the fraction register and the seven low-order bits of the exponent register constitute the 32 c_i output lines in Figure 3.37 which connect to the ac_i input lines of the cell selector unit of Figure 3.34. The a_i and b_i input lines to the adder-subtractor in Figure 3.37 connect to the r_i read lines of the cell memory unit of Figure 3.31.

3.11 Cell Multiplier Unit

The cell floating-point multiplier shown in Figure 3.37 consists of a multiplier register A, a multiplicand register B, an adder, a product register C and normalize logic. The multiplier corresponds to M of Figure 3.30. As was done for the adder-subtractor, the multiplier is shown for 8-bit floating-point numbers. Also as before, the circuits extend in an obvious way to the 32-bit representation.

The multiplier register in Figure 3.37 consists of a 25-bit sign-fraction register connected as a shift register which shifts right and a 7-bit exponent register. When a signal is supplied on the "mloada" line the multiplier register is loaded in parallel from 32 a_i lines, which are connected to the 32 r_i read lines from the cell memory unit. The fraction portion shifts right when a signal is supplied on the "mshifta" line. The sign bit is not included in the shift so the sign is extended in a shift.

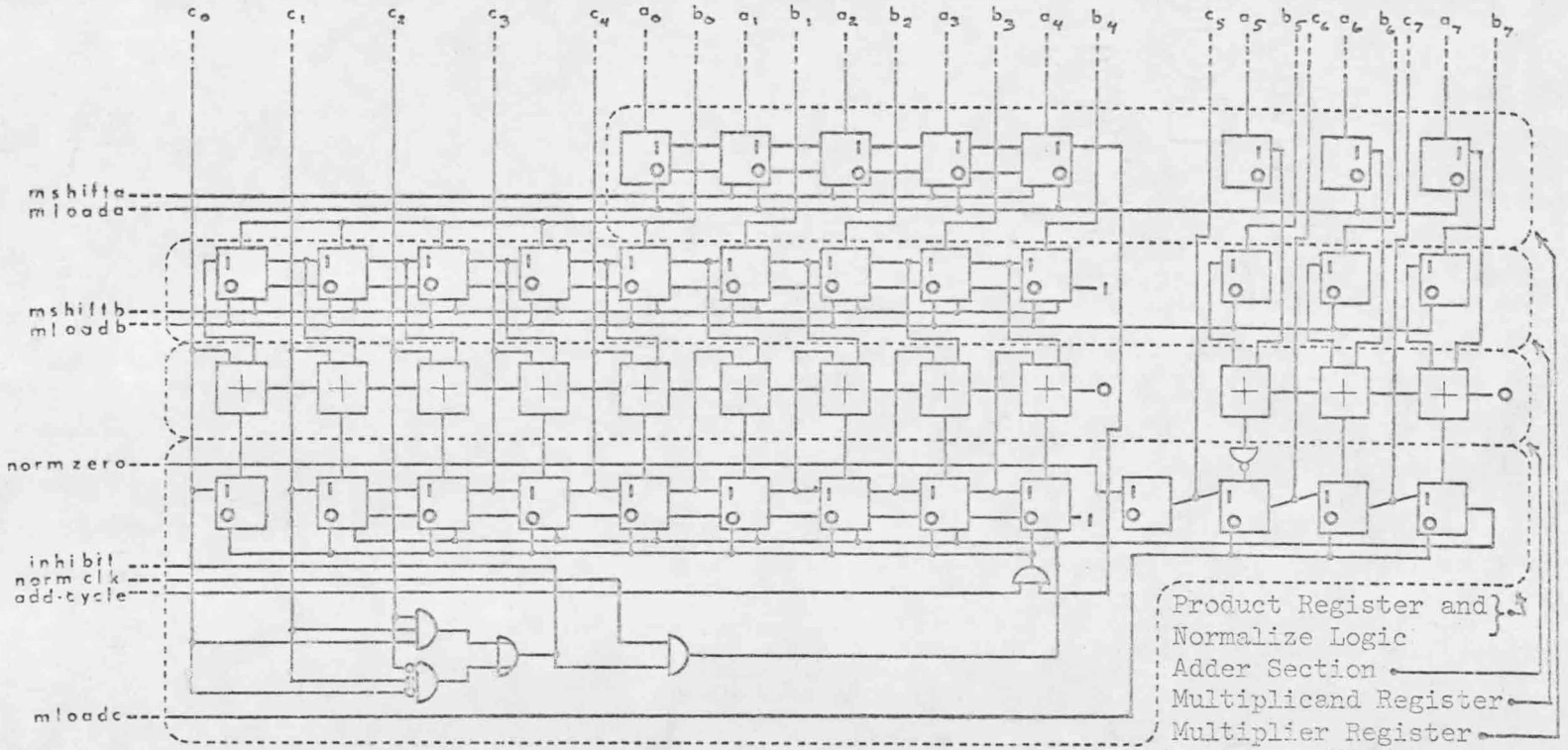


Figure 3.37. Cell Floating-Point Multiplier

The multiplicand register consists of a 49-bit sign-fraction register connected to shift left and an 8-bit exponent register.

Data is entered into the multiplicand register in parallel from 32 b_i lines, connected to the 32 r_i lines from the cell memory unit, when a signal is supplied on the "mloadb" line. When data is entered into the fraction portion of the register it is right-justified and the sign-bit is loaded into all of the 25 high-order bit positions of the register. The fraction portion is shifted to the left and a 0 is shifted into the low-order bit when a signal is provided on the "mshiftb" line.

The adder portion of this multiplier consists of a string of 49 adder cells, previously described in Section 3.3.6, for the fraction and a string of seven adder cells for the exponent. The cells are connected as parallel ripple-carry adders with a 0 supplied to the carry inputs of the low-order bit positions of each.

Inputs to the fraction adder come from the 49 bits of the multiplicand fraction register and from the 49 bits of the product fraction register. Outputs go the 49 bits of the product fraction register.

Inputs to the exponent adder come from the seven bits of the multiplicand exponent register and from the seven bits of the multiplier exponent register. Since the exponents are in an excess-64 representation, adding the A exponent A_e to the B exponent B_e gives

$$(A_e + 64) + (B_e + 64) = A_e + B_e + 64 + 64 = (C_e + 64) + 64.$$

Therefore, it is necessary to subtract 64 from the result. This is accomplished by complementing the high-order bit of the exponent sum.

The product register in Figure 3.37 consists of a 49-bit sign-fraction register connected to shift left and an 8-bit exponent register connected as a decrementing binary counter.

Data is loaded into the fraction portion of the product register in parallel from the 49 output lines of the fraction adder when a signal is supplied on the add-cycle line and when the low-order bit of the multiplier register is a 1. If the low-order bit of the multiplier register is a 0 the contents of the product fraction register is not changed when the add-cycle signal is supplied.

Data is loaded into the exponent portion of the product register in parallel from the seven output lines of the exponent adder when a signal is supplied on the "mloadc" line. The high-order bit is set to 0.

The normalize portion of the multiplier includes the product register and the check-for-normalize circuitry. The normalization is done in the same manner as for the adder-subtractor, with the normalize clock being inhibited when the conditions for normalization are met and with the inhibit and normalize-zero lines as outputs to the global control.

3.12 Row and Column Registers

Along the top and left edges of the array shown in Figure 3.1, are extensions of the global control which are used to distribute commands to and data to and from the array of processing cells. The unit

along the top is logically identical to the one along the left edge except that it is connected to the columns of command and data lines rather than the rows. Thus it is sufficient to discuss the left-edge unit.

The unit shown in Figure 3.38 contains an address decoder, sixteen 32-bit registers, sixteen 4-bit registers and sixteen 1-bit registers. There is one of each of these registers associated with each row of the array.

The address decoder receives inputs from the global control on lines a_0-a_3 and selects one row in a manner similar to that in the cell memory unit. If the line d is a 1 all rows are selected at once.

The selected 32-bit registers may receive data in parallel from the global control on the 32 w_i lines. As in the cell memory unit this data is not entered into the register until a signal is provided on the store line k. The content of the selected register is available on the 32 r_i lines. If the line d is a 1, the result on these lines is the logical OR of the contents of all 16 registers. The 32-bit registers are also connected as shift registers which shift to the right when a signal is supplied on the shift line s. The input to the high-order bit of each register comes from a line-select gate controlled by line b. This gate determines if the input to the shift-register comes from the low-order bit of the same register if b is 1 or from the row line f_i coming from the array if b is 0. The low-order bit of the register goes to the line-select gate and to the line t_i going to the array.

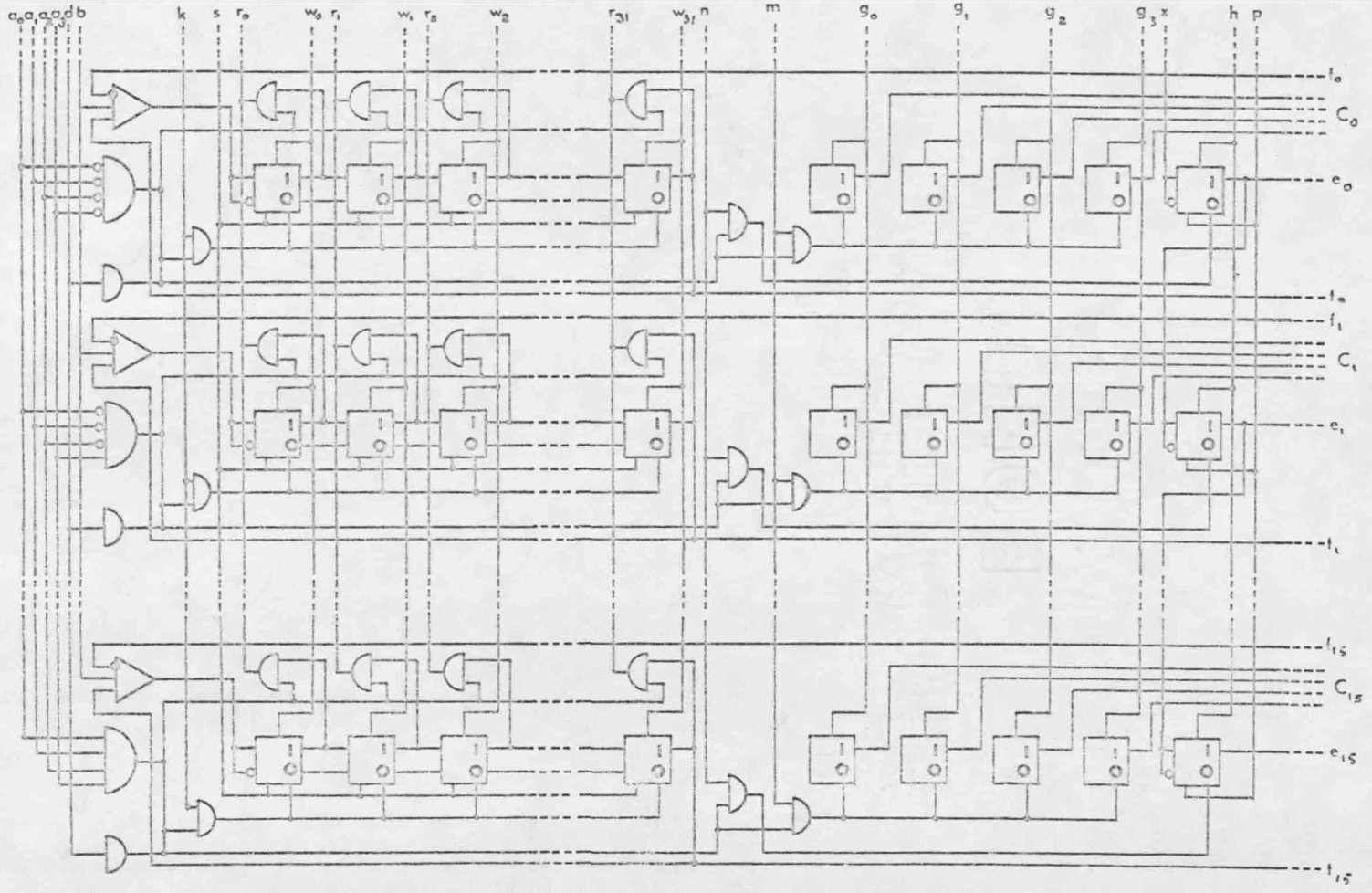


Figure 3.38. Row Registers

The selected 4-bit register may receive data in parallel from the global control on the four g_i lines. The data is not entered into the register until a signal is provided on the line m. The contents of the registers are available on the 64 c_i lines. These lines are connected to the row control lines discussed in Section 3.4 on routing logic.

The selected 1-bit register may receive data from the global control on the h line. The data is not entered into the register until a signal is supplied on the line n. If the line d is a 1 all registers are set the same. The contents of the registers are available on the 16 e_i lines. These lines are used to select those rows of processor cells which will be in an active state. The 16 registers are also connected to form a shift-register so that data may be shifted in from the global control. The shift occurs when a signal is supplied on the line p. The register associated with the first row is treated as the high-order bit. Data is shifted from this register to the register of the second row which shifts to the third and so on. Data input to the first row register from the global control is on line x.

3.13 Software for the KF Machine

Instructions sequences for the KF machine are stored in the global control unit which in itself is capable of performing all of the basic operations of a general purpose computer. Since the KF machine is a general purpose computer, it can perform any computation. However, because of the array of processing cells, computations involving matrices can be done much more efficiently than with a conventional general purpose computer.

Programs for the KF machine consist of array instructions and non-array instructions. Non-array instructions include those which are executed within the global control unit and instructions which perform input or output operations. Array instructions are executed by the array of processing cells under the supervision of the global control unit.

Although no formal assembly language has been written for the KF machine, it is expected that most programming would be done using such a language. A typical instruction for adding matrices in the assembly language might be ADDM 1 2 3 which would transfer data from memory locations 1 and 2 of each cell to the adder-subtractor in the cell, perform the addition and store the resulting sum in memory location 3 of the cell. Other instructions are easily imagined for performing a full set of operations.

A set of micro-instructions was formed for use in the simulation program in Chapter 4 and is presented there. This set of instructions

is not intended to serve as an assembly language for the KF machine but rather as a means of controlling the simulation at the gate level. A typical instruction in an assembly language for the machine would have the same effect as several of the micro-instructions. For example the matrix addition instruction mentioned above might generate the micro-instruction sequence

```
LDAA  1
LDAB  2
SETI  0  80
ALIN
BDIB  0  02
LDAC
SETI  0  18
ANOR
BDIB  0  02
STAC  3,
```

where the micro-instructions are those used by the simulation program in Chapter 4.

3.14 Summary

In this chapter a design was presented for a specially-organized, general purpose computer which is particularly efficient when performing matrix operations. Section 3.2 gives the overall layout of the machine, Section 3.4 describes the interconnection structure and Section 3.6 shows the structure of the cells. Detailed designs for part of the computer are given in Sections 3.7 through 3.12.

Since the computer is designed for efficiency when performing matrix operations, those parts of the computer which are directly responsible for operations on matrices have been considered in greater detail than others.

The machine consists of a macrocellular array of cells and a central control unit. Although a 16x16 array of cells was chosen, there is no reason why much larger arrays of the same cells could not be considered. Similarly, the amount of memory chosen for each cell could just as well be larger, keeping in mind that for an N x N array, N^2 components are added to the array for each one that is added to a cell.

Since all cells are identical, the design of one cell represents a large portion of the logic of the computer. Similarly since the array of cells comprise a major part of the computer, the cost per gate of the computer should be much less than for a conventionally organized computer. Further manufacturing and maintenance gains are made by the repetitious use of one circuit (a cell) throughout much of the machine.

Judging from the current status of large scale integrated circuit (LSI) technology and the predictions for future capabilities in this area[20], it is not unreasonable to make estimations of the possible hardware realizations of the proposed KF maching. Since a single cell contains about 1000 logic units with approximately the same complexity as a shift register stage, there should be somewhat less than 8000 active devices in a cell. It appears that in the near future it may be possible to realize an entire cell of the KF machine with a single LSI package.

However, if this is not feasible there are several logical divisions of a cell which might be made in order to break it into sections which may be realized. For example, since about half of the logic of a cell is devoted to the cell memory, it might be possible to realize the memory in one package and the rest of the cell in another. Other divisions might break the cell into many sections which may be realized with currently available integrated circuits.

In the Kalman filter example discussed in Section 2.5 roughly 80% of the computations involved matrices. Therefore, a computer which performs matrix operations more effciently than a conventional computer should have significantly shorter execution times for this problem.

In order to compare the speed of the proposed KF machine with that of a conventionally structured computer, some matrix operations will be considered.

The addition or subtraction of two N x N matrices requires one add cycle for the KF machine compared with N^2 add cycles for the conventional computer.

A matrix multiplication of two N x N matrices in the KF machine requires N multiplication cycles, N-1 add cycles and 3N shift cycles. A shift cycle is the time required to route data a distance of one cell. In the conventional computer, the matrix multiplication would require N^3 multiplication cycles and N^3-N^2 addition cycles.

Using the matrix inversion algorithm described in Section 2.4, the KF machine would require N divide cycles, 4N shift cycles, N add cycles

and 2N multiply cycles to invert an N x N matrix. On the other hand, the conventional computer would take N divide cycles, N^3-N^2 add cycles and N^3-N^2+N multiply cycles.

The above comparisons are summarized in Table 3.3.

Table 3.4. Comparison of Machine Cycles for Matrix Operations.

Matrix Operation	Machine Cycles for KF machine	Machine Cycles for conventional machine
Add	1 Add cycle	N^2 Add cycles
Subtract	1 Add cycle	N^2 Add cycles
Multiply	N multiply cycles N-1 Add cycles 3N Shift cycles	N^3 Multiply cycles N^3-N^2 Add cycles
Invert	N Divide cycles 2N Multiply cycles N Add cycles 4N Shift cycles	N Divide cycles N^3-N^2+N Multiply cycles N^3-N^2 Add cycles

Most of the matrices in the Kalman filter example are 4 x 4 matrices. Using the results in Table 3.3 a matrix multiplication in the

KF Machine should be performed roughly 20 times as fast as in the conventional computer. For a 16 x 16 matrix the ratio becomes approximately 250.

Several things have not been taken into account in the calculation of these rough estimates. The arithmetic units of the cells are designed to perform calculations rapidly without resorting to large amounts of hardware. While they should be much faster than a serial arithmetic unit, they would probably not be as fast as the high-speed arithmetic unit found in most conventional computers. On the other hand, since data is stored in a non-destructive read-out memory within the cells, the time required to transfer an entire array of data from memory to the arithmetic units of all cells is likely less than that required to transfer a single word of data from the memory to the arithmetic unit of the conventional computer.

If the KF machine is assumed to be 20 times as fast as the conventional for matrix operations, a problem in which these operations consumed 80% of the computation time would run roughly four times as fast on the KF machine as on the conventional and less than 20% of the time would be used for the matrix operations. It is assumed here that the non-matrix operations are performed at the same rate in either machine. If the KF machine performs matrix operations 100 times as fast, a problem in which these operations required 90% of the computation time would run about ten times as fast and the matrix operations would consume less than 10% of this time.

Chapter 4

SIMULATION OF

THE CELLULAR COMPUTER

4.1 Simulation Program

In order to verify that the proposed array interconnection structure and cell logic will perform the operations for which they were intended, a detailed simulation program was written. The program was written in SYMBOL, an assembly language for the SDS Sigma 7 computer, as a subroutine to a FORTRAN mainline program. A FORTRAN subroutine is used for input and output operations and another SYMBOL subroutine is used to convert data from Sigma 7 floating-point form to the form used in the KF machine for input and to convert back for output.

Listings of the simulation program and associated subroutines are given in Appendix B. Modifications to the design of the KF machine may be checked by modifying corresponding parts of the simulation program. A flow chart of the simulation program is given in Figure 4.1.

To initialize the simulation, all of the memory areas for the cells and the load-location counter, called LOADLOC, are set to zero. The block labeled INITIALIZE in Figure 4.1 corresponds to the initialization process.

The LOAD INSTRUCTIONS process indicated in Figure 4.1 is shown in more detail in the flow chart of Figure 4.2. To load an instruction, LOADLOC is incremented by one, a card containing the instruction is read and a copy of the card is first displayed on the line printer. It is then stored on a random access disc, using LOADLOC as a record address on the disc. A disc is used in the simulation as the program

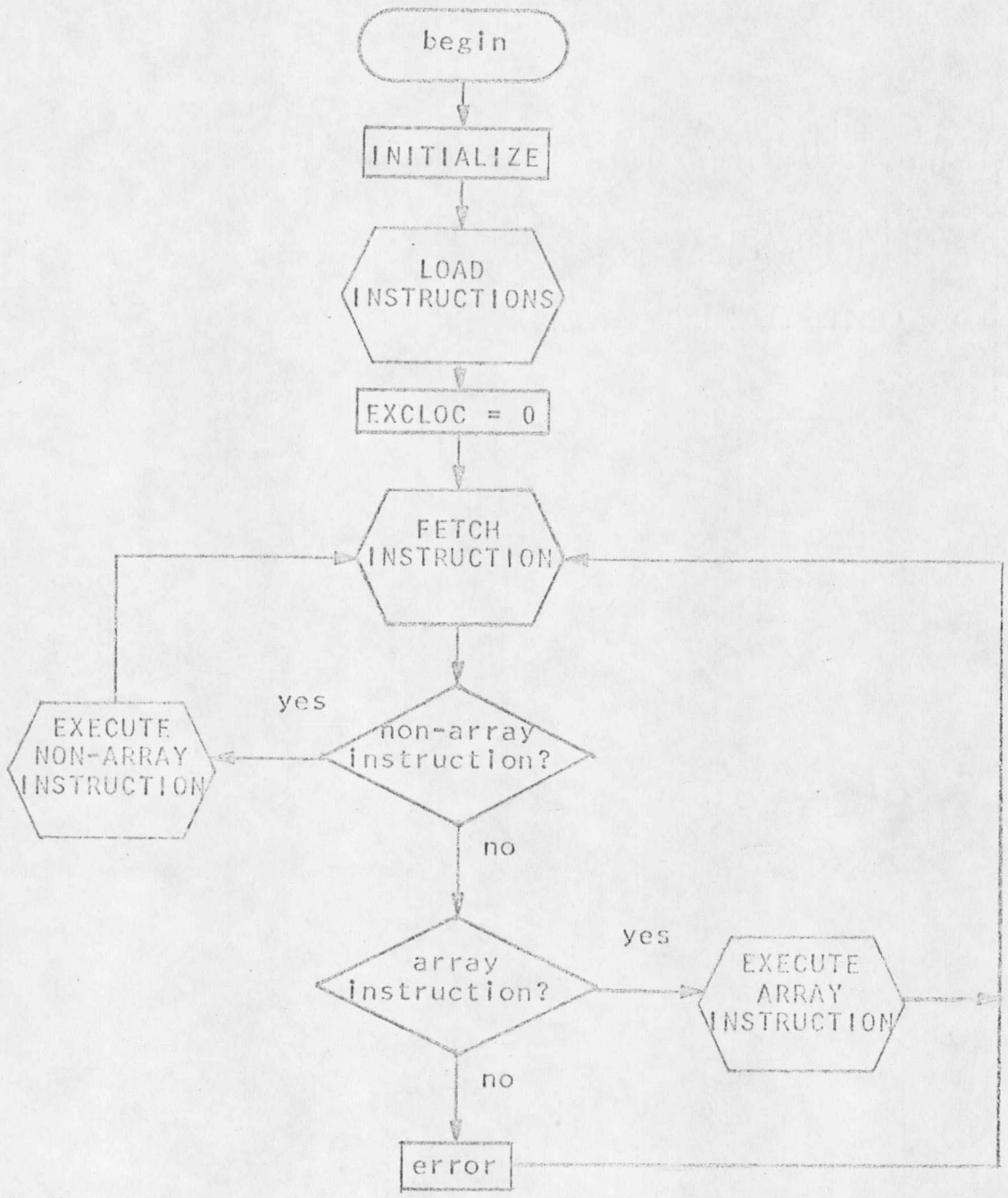


Figure 4.1. Flow Chart for Logic Simulation Program

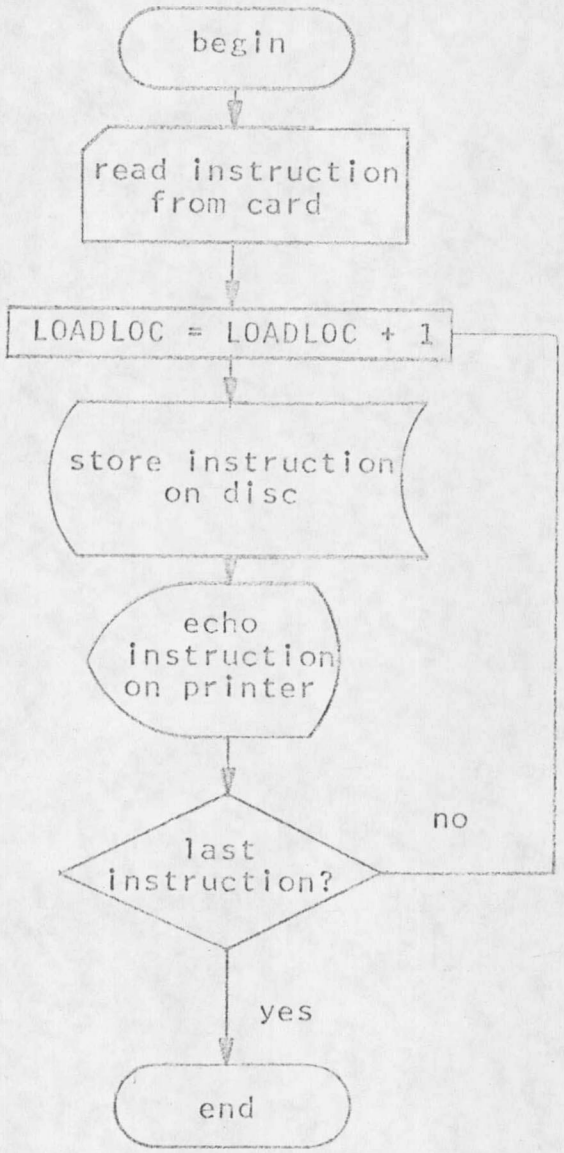


Figure 4.2. LOAD INSTRUCTIONS Flow Chart

storage memory of the global control unit in order to conserve core storage for other program data. After the instruction has been stored it is checked to see if it was the last instruction. A dollar sign ($) in the first column of an instruction card signifies that it is the last card.

When all instructions have been stored on the disc, the execution begins. The execution location counter, called EXCLOC, is first set to zero as indicated in Figure 4.1.

Figure 4.3 shows a flow chart corresponding to the FETCH INSTRUCTION block of Figure 4.1. To begin the execution of an instruction, EXCLOC is incremented by one, and the instruction is read from the disc using EXCLOC as a record address on the disc. Once read, all blanks are removed from the instruction and it is left-justified in the instruction buffer. If the instruction just read is the last instruction, the simulation is terminated.

As indicated in Figure 4.1, the instruction is compared with each of the list of non-array instructions shown in Table 4.1. If the instruction is found in the list the simulation program branches to a routine which executes the instruction. There is a separate routine for each of the non-array instructions. As an example of a non-array instruction, consider the branch instruction. To execute a branch instruction, EXCLOC is modified by the amount specified in the instruction. Thus, the next instruction to be read from the disc will come

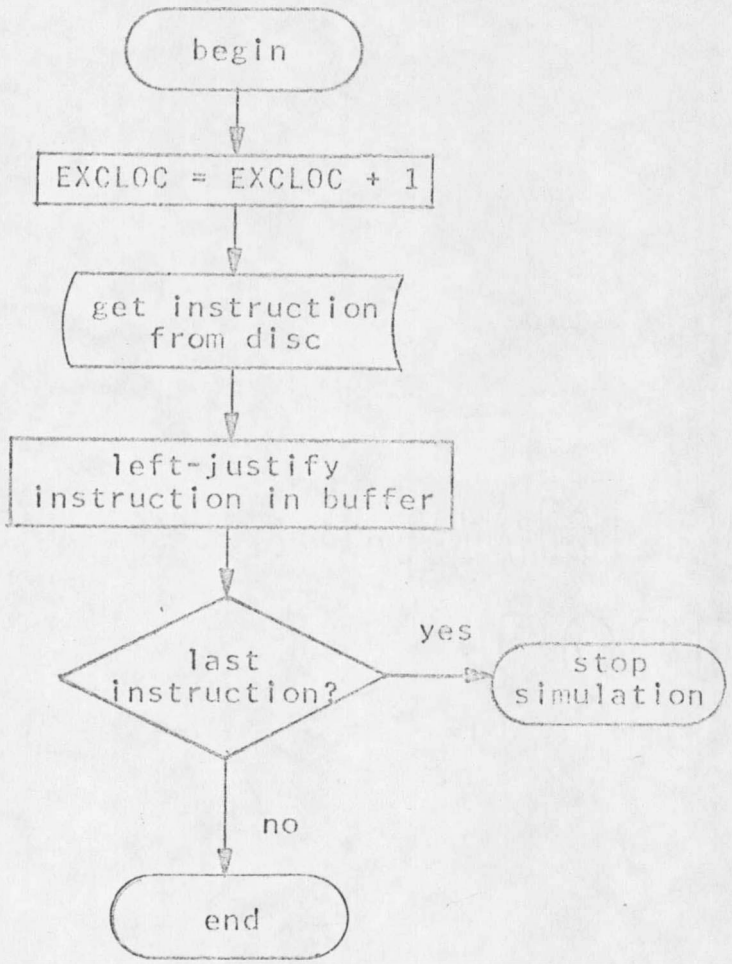


Figure 4.3. FETCH INSTRUCTIONS Flow Chart

Table 4.1. Non-Array Instructions Used in Simulation.

Instruction	Purpose
DUMP X Y	Dump memory location (X,Y),
INVR X	Invert row buffer X.
INVC X	Invert column buffer X.
RBON	Set the row register feedback line to 1.
RBOF	Set the row register feedback line to 0.
CBON	Set the column register feedback line to 1.
CBOF	Set the column register feedback line to 0.
GLON	Set the global routing control line to 1.
GLOF	Set the global routing control line to 0.
RION	Set the row input control line to 1.
CION	Set the column input control line to 1.
RIOF	Set the row input control line to 0.
CIOF	Set the column input control line to 0.
TRON	Set the transpose control line to 1.
TROF	Set the transpose control line to 0.
BDIB X YY	Decrement index X, if result is not greater than zero, branch back YY instructions.
BFWD XX	Branch forward XX instructions.
BBAK XX	Branch back XX instructions.
SETI X YY	Set index X to the value YY.

Table 4.1. (Continued).

BDZF XX	If done line is zero branch forward XX instructions.
BDZB XX	If done line is zero branch back XX instructions.
RCON X Y	Set routing row control X to the value Y.
RCAM X	Set all routing row controls to the value X
CCOM X Y	Set routing column control X to the value Y.
CCAM X	Set all routing column controls to the value X.
INRO	Input data to the row buffers.
OURO	Output data from the row buffers.
INCO	Input data to the column buffers.
OUCO	Output data from the column buffers.
$$$$	Stop.

from the location specified by the new EXCLOC. When the execution of a non-array instruction is complete, the simulation returns to the FETCH INSTRUCTION procedure as shown in Figure 4.1.

If the instruction is not found in the list of non-array instructions, the list of array instructions shown in Table 4.2 is scanned. If the instruction is found in this list the procedure labeled EXECUTE ARRAY INSTRUCTION in Figure 4.1 is entered. If not, an error indication is given and the simulation skips to the next instruction.

A flow chart for the EXECUTE ARRAY INSTRUCTION procedure is given in Figure 4.4. The first step in this procedure is to obtain a word of data containing control line settings corresponding to the instruction to be executed from Table 4.3. The arguments associated with the instruction are merged into this word and the result is stored in a word called CONTROL. The first argument is then stored in ADDRESS. At this point the global routing control line, called GLOBAL, is tested to determine if row, called ROWCOM, or column, called COLCOM, control lines are to be used for routing. The routing control lines for each cell, called CELLCOM, are then set to the appropriate values.

If the instruction being executed is a routing instruction, the interconnection structure simulation is now performed as shown in Figure 4.4. If not, the program skips ahead to the processor cell scanning portion.

Table 4.2. Array Instructions Used in Simulation.

Instruction	Purpose
LDRA X	Load routing A register from memory location X.
LDRB X	Load routing B register from memory location X.
LDAA X	Load adder A register from memory location X.
LDAB X	Load adder B register from memory location X.
LDAC	Load adder C register (add).
LDMA X	Load multiplier A register from memory location X.
LDMB X	Load multiplier B register from memory location X.
LDMC	Load multiplier C register (add exponents).
STRA X	Store routing A register in memory location X.
STRB X	Store routing B register in memory location X.
STAC X	Store adder C register in memory location X.
STMC X	Store multiplier C register in memory location X.
ALIN	Pulse the adder alignment clock.
ZERC	Clear multiplier C (fraction) register.
MADD	Pulse the multiplier add cycle clock.
ANOR	Pulse the adder normalize clock.
SUBT	Load adder C register (subtract).
MNOR	Pulse the multiplier normalize clock.
SFMA	Shift the multiplier A register right.
SFMB	Shift the multiplier B register left.

Table 4.2. (Continued).

SMAB	Shift multiplier A and B registers.
RTAO	Pulse routing A register clock.
RTBO	Pulse routing B register clock.
RTAB	Pulse routing A and B register clocks.

Table 4.3. Array Instruction Control-Line Settings

Instruction	Control Lines		
	SEL0 SEL1 STORE ARTLOAD ARCLOCK BRTLOAD BRCLOCK TEMP	ALOADA ALOADB ALIGN ALOADC ASLINE NORMCLK MLOADA MSHIFTA	MLOADB MSHIFTB MLOADC ZEROMC MADDCYC MNORMCLK
LDRA	00010000	00000000	000000
LDRB	00000100	00000000	000000
LDAA	00000000	10000000	000000
LDAB	00000000	01000000	000000
LDAC	00000000	00010000	000000
LDMA	00000000	00000010	000000
LDMB	00000000	00000000	100000
LDMC	00000000	00000000	001000
STRA	00100000	00000000	000000
STRB	01100000	00000000	000000
STAC	10100000	00000000	000000
STMC	11100000	00000000	000000
ALIN	00000000	00100000	000000
ZERC	00000000	00000000	000100
MADD	00000000	00000000	000010
SUBT	00000000	00011000	000000
ANOR	00000000	00000100	000000
MNOR	00000000	00000000	000001
SFMA	00000000	00000001	000000
SFMB	00000000	00000000	010000
SMAB	00000000	00000001	010000
RTAO	00001000	00000000	000000
RTBO	00000010	00000000	000000
RTAB	00001010	00000000	000000

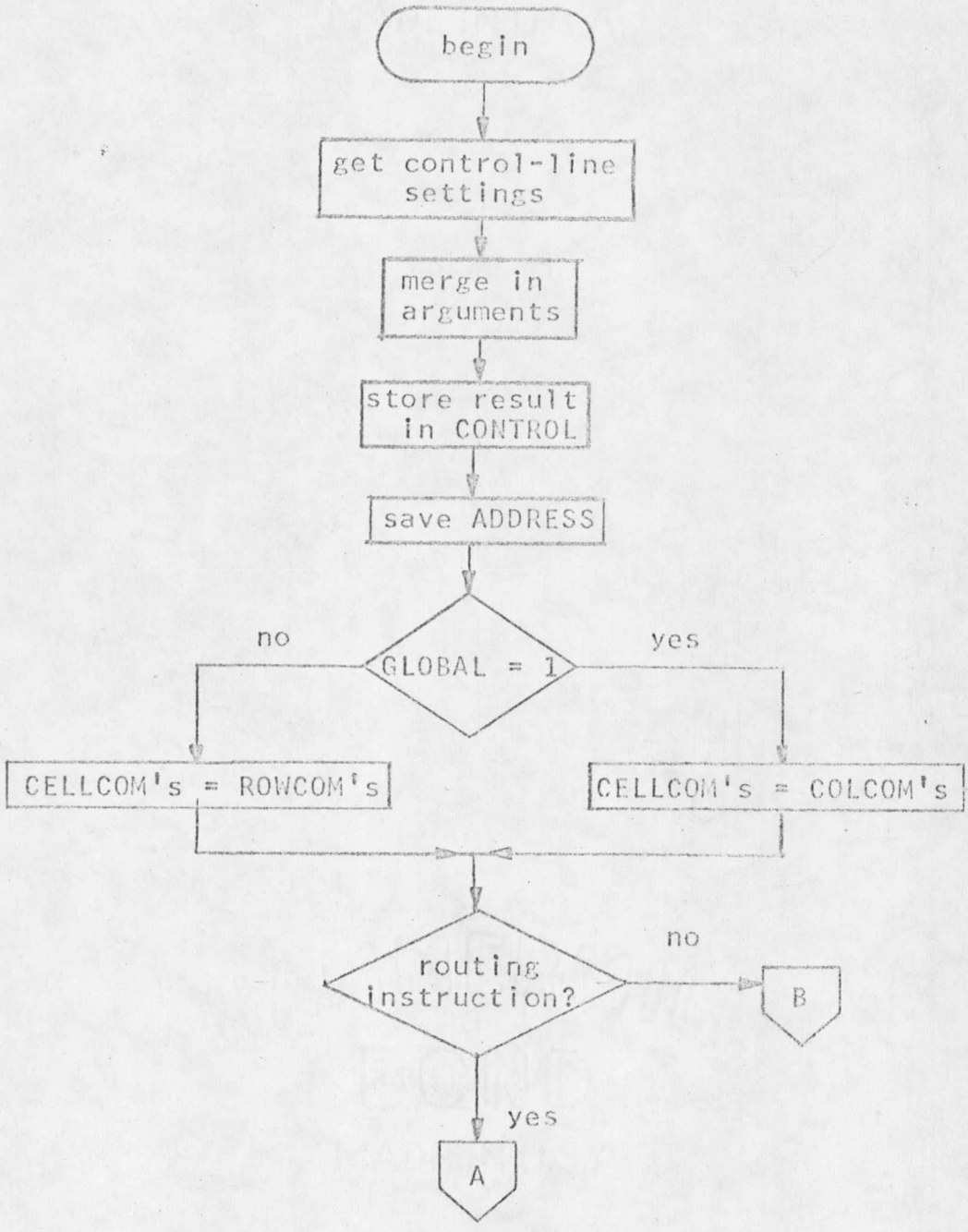


Figure 4.4. EXECUTE ARRAY INSTRUCTIONS Flow Chart

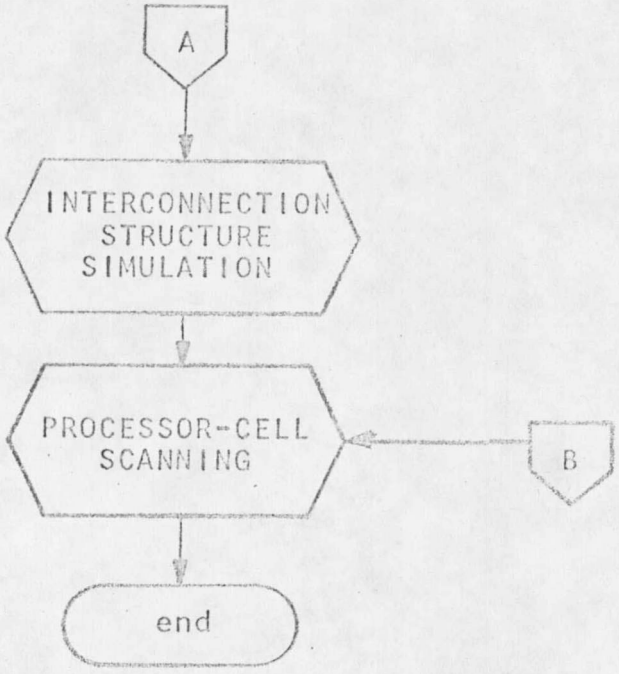


Figure 4.4. (Continued)

To simulate the interconnection structure, two auxiliary data blocks celled INBIT and OUTBIT are used. These blocks each have 544 bits of storage in the format of Figre 4.5 where A represents bits associated

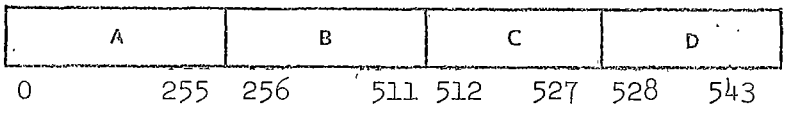


Figure 4.5. Format for INBIT and OUTBIT Blocks.

with the routing A registers of all cells in row precedence order, B represents bits associated with the routing B registers in row precedence order, C represents bits associated with the row buffer routing registers and D represents bits associated with the column buffer routing registers. The INBIT block represents those bits which appear as inputs to the corresponding routing registers. The OUTBIT block represents those bits which appear as outputs from the corresponding routing registers. Since it is possible for the routing registers, themselves, to be by-passed in some cells, the INBIT and OUTBIT blocks do not necessarily represent bits which would be shifted in or out, respectively, of the registers. For this reason it is necessary to take special action to account for the by-passed cells, as will be seen later.

A flow chart for the INTERCONNECTION STRUCTURE SIMULATION is given in Figure 4.6.

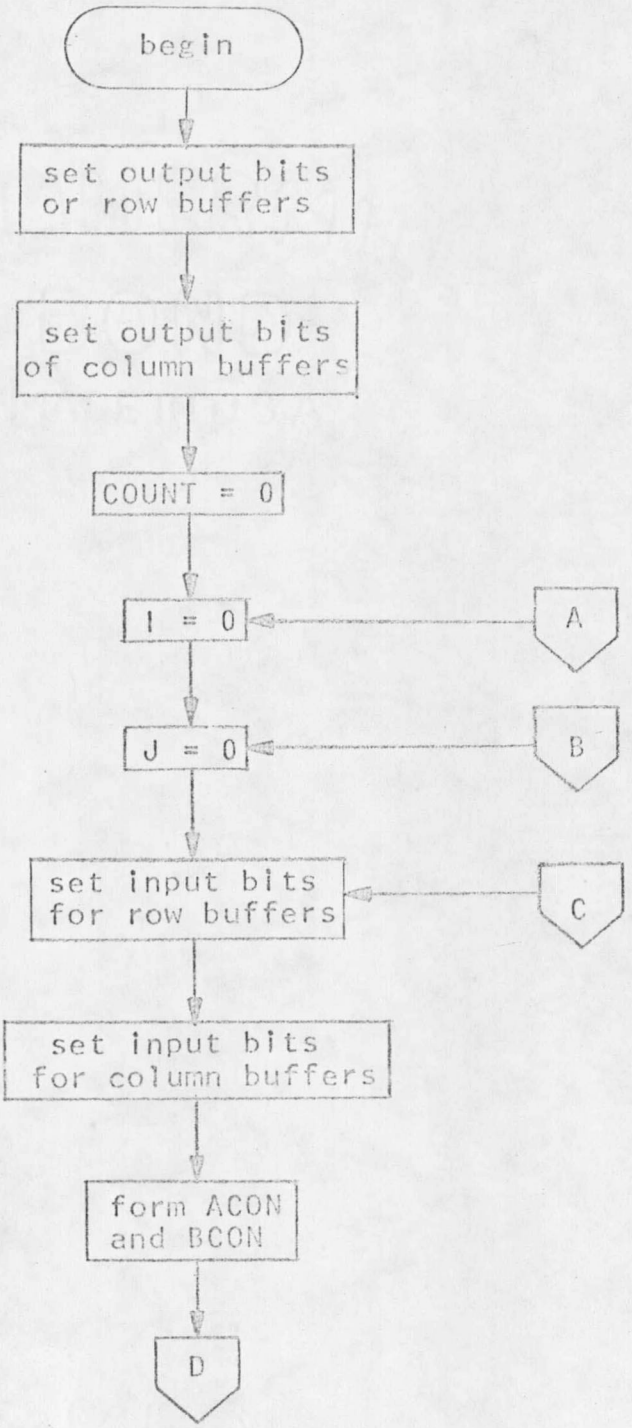


Figure 4.6. INTERCONNECTION STRUCTURE SIMULATION Flow Chart

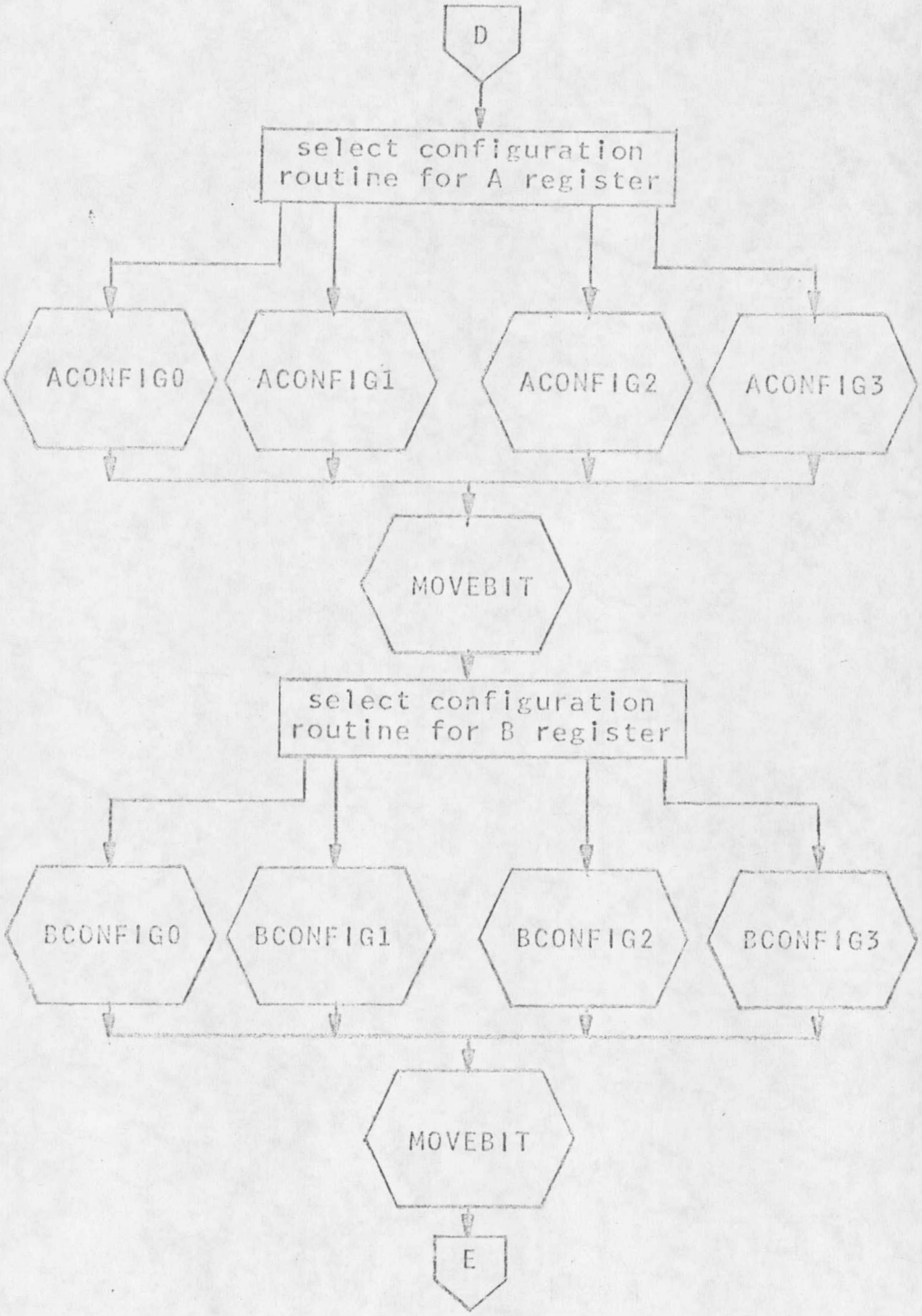


Figure 4.6. (Continued)

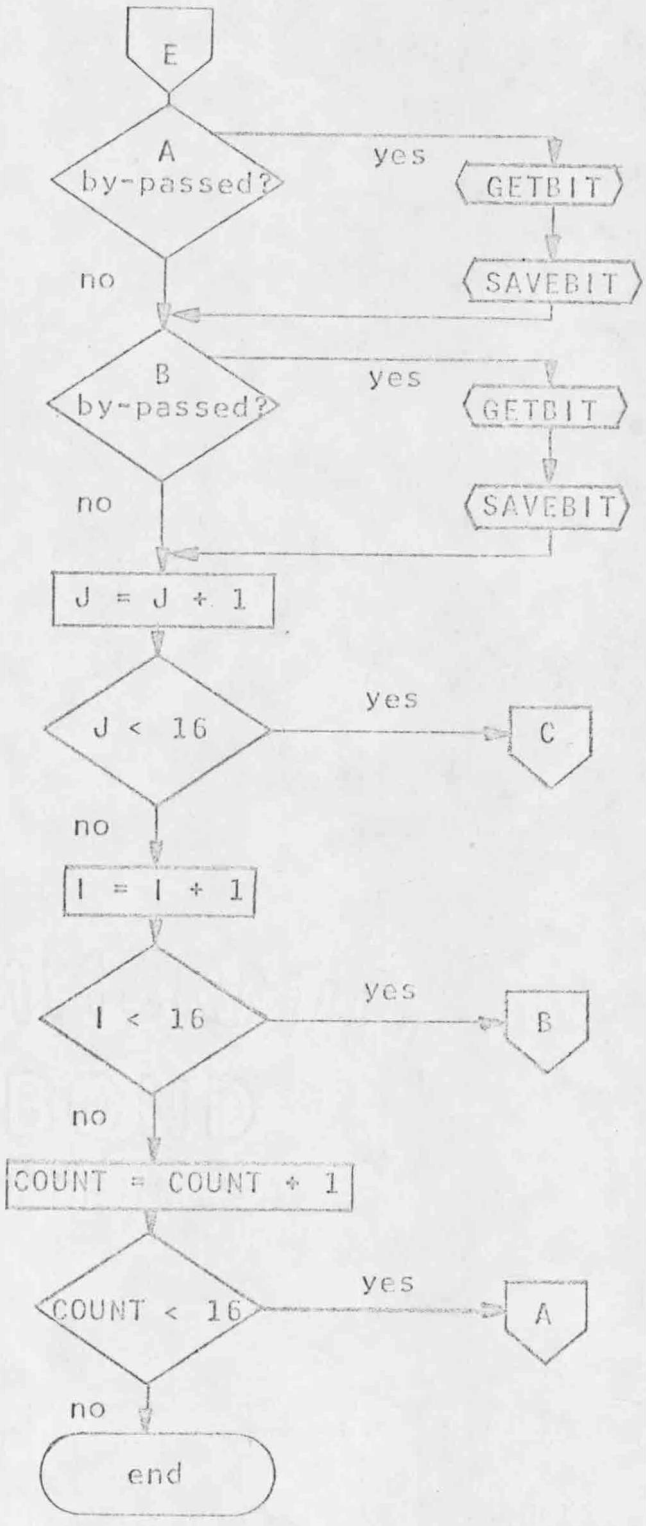


Figure 4.6. (Continued)

The first step in Figure 4.6 is to set the output bits of the row and column buffers. Since these registers cannot be by-passed, the output bits are just the low-order bits of the registers.

Next, the input bits for the row and column buffers are set. The source for the input bits depends only on the value of the transpose control line, called TRANSP, the row buffer feedback line, called ROWBACK, and the column buffer feedback line, called COLBACK. ROWBACK and COLBACK determine if the inputs to the row and column buffers are to come from the array or are to be fed back from the outputs of the row and column buffers themselves. If the latter is the case, data will just cycle within the registers of the row and column buffers as the routing is performed. The input bits are set by selectively transferring bits from the OUTBIT block to the INBIT block according to the values of TRANSP, ROWBACK, and COLBACK.

The next step is to form new two-bit variables ACON and BCON by combining TRANSP with ROWINP and COLINP respectively. ROWINP and COLINP indicate whether or not data are to be transferred from the row or column buffers to the array in a routing operation. ACON and BCON will be used later.

As shown in Figure 4.6 the input bits for all routing registers are to be set now. All of the cells are scanned to do this. The first step in the scanning loop is to find the input bit for the routing A register. ACON along with CELLCOM for the current cell are used, with

the exception of the by-pass bits of CELLCOM, to determine which of four interconnection configurations is to be used to find the input bit for the routing A register. After determining the configuration to be used, the input bit is found by using the appropriate routine of ACONFIG0, ACONFIG1, ACONFIG2 or ACONFIG3. The purpose of these routines is to set up a bit address, called BITADR, which indicates which bit in INBIT is to be the input. Since this address is a function of the location of the current cell within the array, it must be calculated within the cell scanning loop. Once the address has been determined another bit address, called TOADR, representing the position of the current cell is generated. A routine called MOVEBIT is then used to move the bit of OUTBIT specified by BITADR to the bit position in INBIT specified by TOADR.

A similar procedure is then used to find and set up the input bit for the routing B register, except that BCON is used to determine the configuration and one of the routines BCONFIG0, BCONFIG1, BCONFIG2 or BCONFIG3 is used to find the bit address of the input bit.

Once the input bits have been set up this way, the problem of by-passed cells is resolved. Using the by-pass bit from CELLCOM, the cell output bits are either left alone, if the cell is by-passed, or modified, if the cell is not by-passed. If the cell is to be by-passed a bit address, again BITADR, corresponding to the cell address is set up. A routine called GETBIT is used to get a bit from the bit position

in INBIT specified by BITADR and a routine called SAVEBIT is used to place the bit in the corresponding bit position in OUTBIT, thus by-passing the register. This process is done first for the routing A register then for the routing B register as shown in Figure 4.6.

This terminates the cell scanning loop. Now the entire process, starting with the setting of the input bits for row and column buffers, is repeated for a total of 16 times. This allows bits to propagate through 16 by-passed cells. It should be recognized that if a cell is by-passed, cells which use its output as their input will not receive the correct bit if this is not done.

Referring back to Figure 4.4, the next step in the execution of an array instruction is to scan the processor cells. The flow chart for this procedure is shown in Figure 4.7.

The PROCESSOR CELL SCANNING procedure scans through the processor cells one at a time simulating the cell components. For each cell the data is moved from a large data block, called ARRAY, to an area used by the cell currently being simulated. Part of the data is stored 32 bits per word of Sigma 7 memory to conserve memory space but is expanded to one bit per word for ease of manipulation in the simulator. The data for a cell is expanded at the beginning of the simulation for that cell.

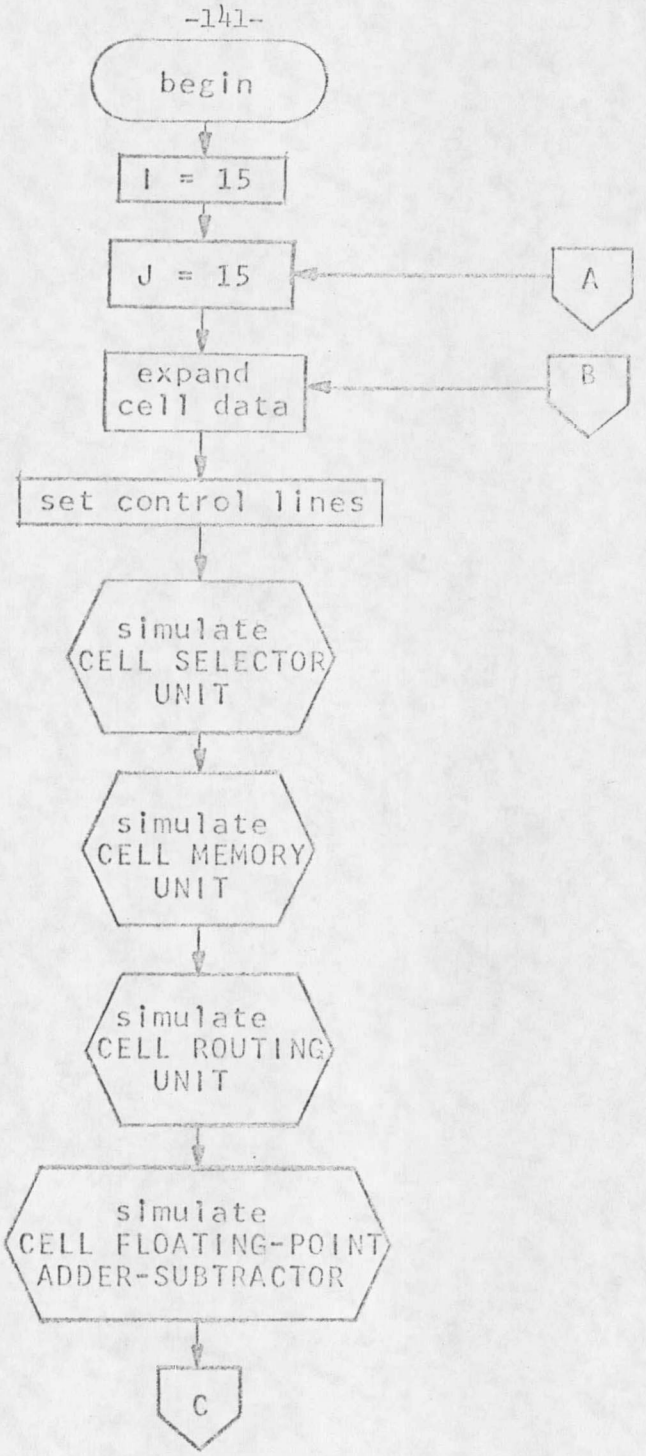


Figure 4.7. PROCESSOR CELL SCANNING Flow Chart

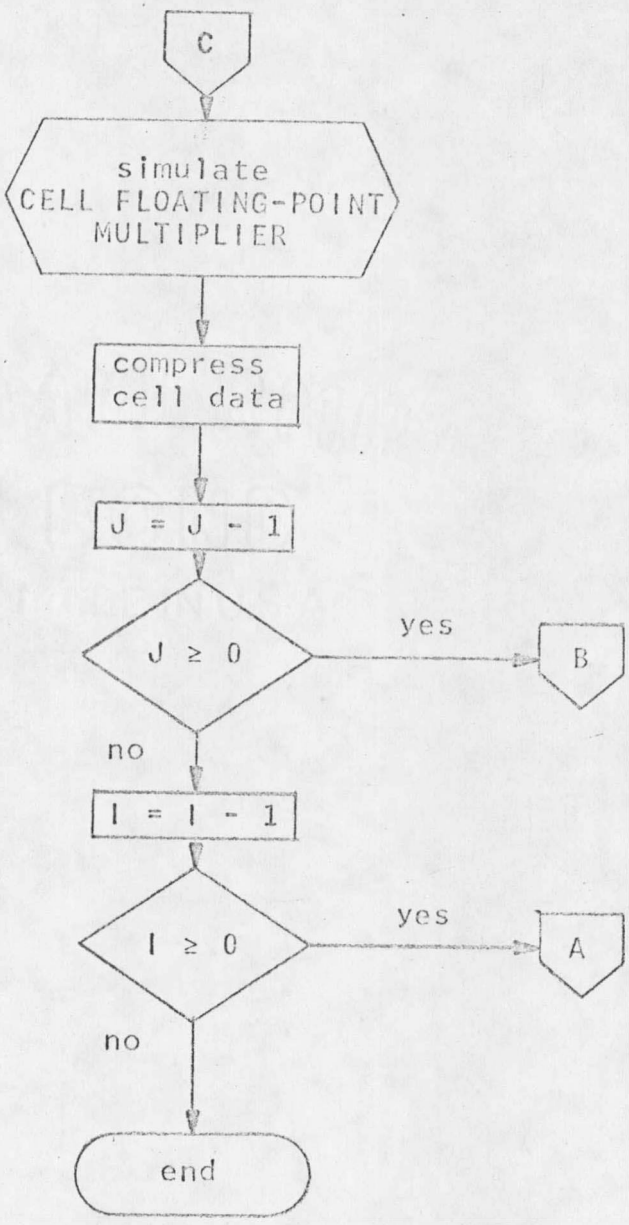


Figure 4.7. (Continued)

The next step in the cell simulation as indicated in Figure 4.7 is the setting of control lines. The variable called CONTROL is expanded to one bit per word to set all of the cell control lines. The set of control lines and their use is given by Table 4.4. It should be noted that several of these lines may be set at once. Thus, for example, it is possible to shift two registers of a cell at the same time.

The CELL SELECTOR unit is now simulated as shown in the flow chart of Figure 4.8. SEL0 and SEL1 are used to select data for the memory write lines, called WLINE, according to Table 4.5.

The next step in Figure 4.7 is the CELL MEMORY UNIT SIMULATION. This routing shown in Figure 4.9 stores the data on the WLINE in the cell memory location specified by ADDRESS if STORE = 1. The data in the specified cell memory location is stored in the word representing the memory read lines, called RLINE.

Figure 4.10 gives a flow chart for the CELL ROUTING UNIT SIMULATION of Figure 4.7. The first step in this routine is to lead the routing register from the RLINE if ARTLOAD = 1. Then the routing B register, called RBREG, is loaded from RLINE if BRTLOAD = 1. New values of output bits for the registers are set in OUTBIT using the SAVEBIT routine. If ARCLOCK = 1 and the register is not by-passed the routing A register is shifted right one bit using the appropriate bit from INBIT obtained using the GETBIT routine. The new output bit

Table 4.4. Cell Control Lines.

Line	Purpose
SEL0	First selector control.
SEL1	Second selector control.
STORE	Memory store.
ARTLOAD	Load routing A register.
ARCLOCK	Shift routing A register.
BRTLOAD	Load routing B register.
BRCLOCK	Shift routing B register.
ALDADA	Load adder-subtractor A register.
ALOADB	Load adder-subtractor B register.
ALIGN	Adder-subtractor alignment clock.
ALOADC	Load adder-subtractor C register.
ASLINE	Add/Subtract select line.
NORMCLK	Adder-subtractor normalize clock.
MLOADA	Load multiplier A register.
MSHIFTA	Shift multiplier A register.
MLOADB	Load multiplier B register.
MSHIFTB	Shift multiplier B register.
MLOADC	Load multiplier C (exp) register.
ZEROMC	Clear multiplier C (fraction) register.
MADDCYC	Multiplier add cycle clock.
MNORMCLK	Multiplier normalize clock.

Table 4.5. Cell Selector Control Line Table

SEL0	SEL1	Data Source
0	0	Routing A register
0	1	Routing B register
1	0	Adder-Subtractor C register
1	1	Multiplier C register

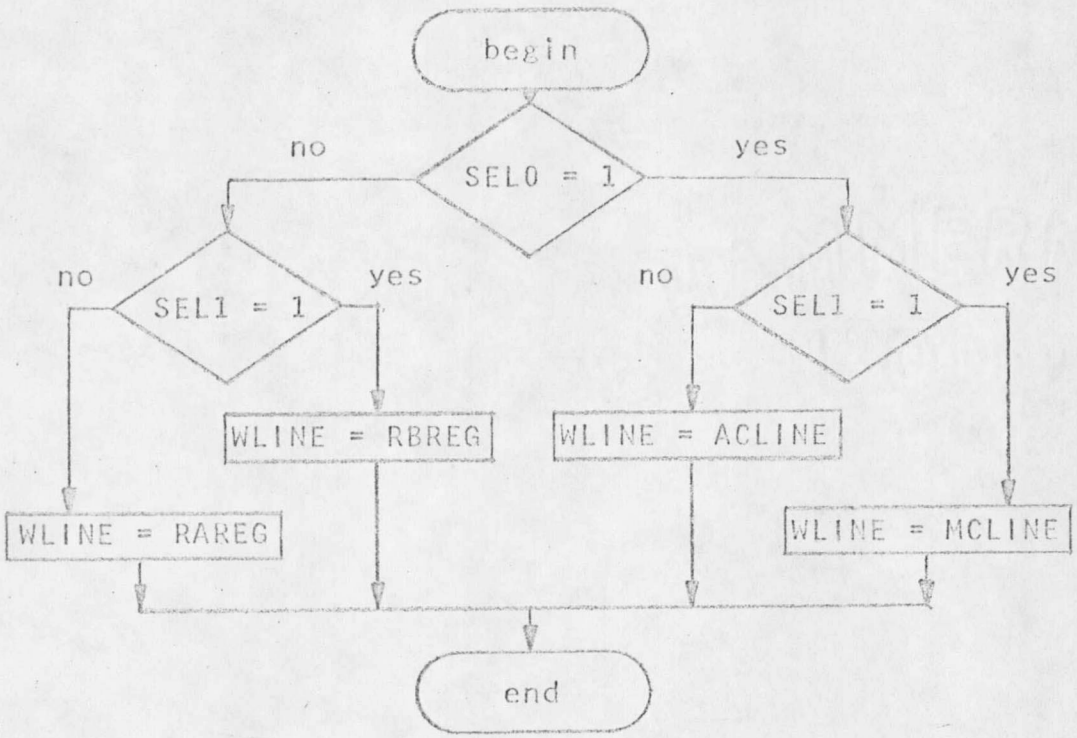


Figure 4.8. CELL SELECTOR UNIT Flow Chart

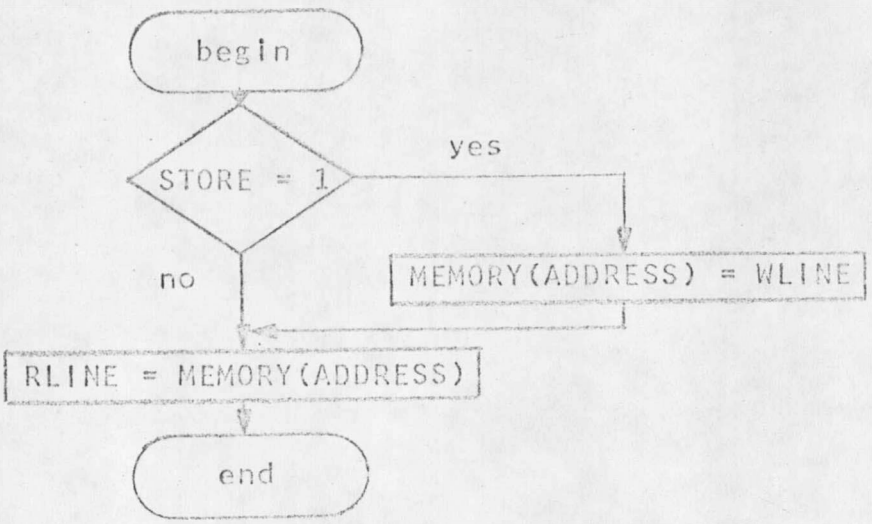


Figure 4.9. CELL MEMORY UNIT Flow Chart

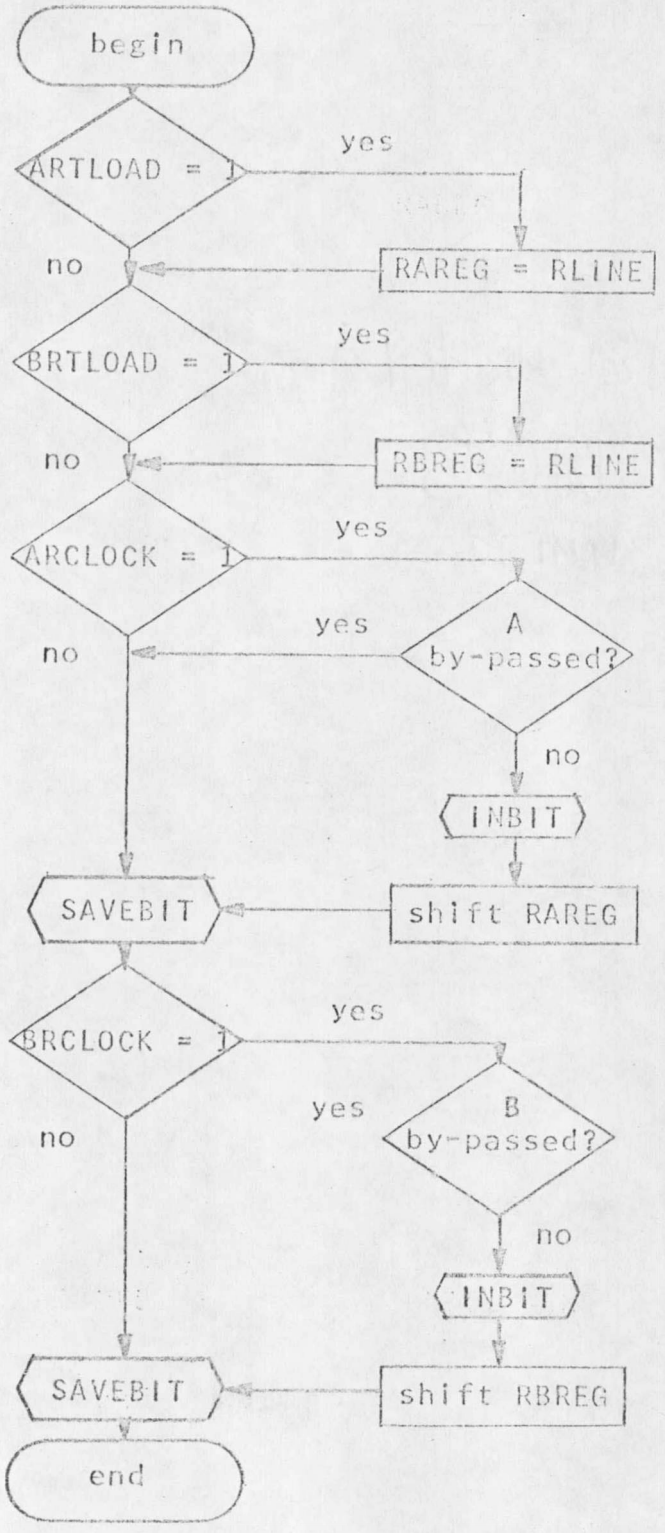


Figure 4.10. CELL ROUTING UNIT Flow Chart

is set in OUTBIT using the SAVEBIT routine. Similarly the routing B register is shifted right one bit if BRCLOCK = 1 and the register is not by-passed.

As shown in Figure 4.7 the next step in the simulation of the cell components is the CELL FLOATING-POINT ADDER-SUBTRACTOR SIMULATION. This simulation is described by the flow chart of Figure 4.11. The adder-subtractor A register is loaded from RLINE if ALOADA = 1, and the B register is loaded from RLINE if ALOADB = 1. If ALIGNA = 1, the A and B exponent registers are compared and if they are not equal, the smaller one is incremented, and its corresponding fraction register is shifted right one bit. If LOADC = 1, the sum, if ASLINE = 0, or the difference, if ASLINE = 1, of the A and B fraction registers is loaded into the C fraction register and the B exponent register plus one is loaded into the C exponent register. If the normalize conditions are not satisfied by the contents of the C register and NORMCLK = 1, the C fraction register is shifted left one bit and the C exponent register is decremented by one.

The last cell component to be simulated is the CELL FLOATING-POINT MULTIPLIER as indicated in Figure 4.7. A flow chart for the multiplier simulation is given in Figure 4.12. The multiplier A register is loaded from RLINE if MLOADA = 1, and the B register is loaded from RLINE if MLOADB = 1. If MSHIFTA = 1 the A fraction register is shifted right one bit. Similarly, if MSHIFTB = 1 the B fraction register is shifted

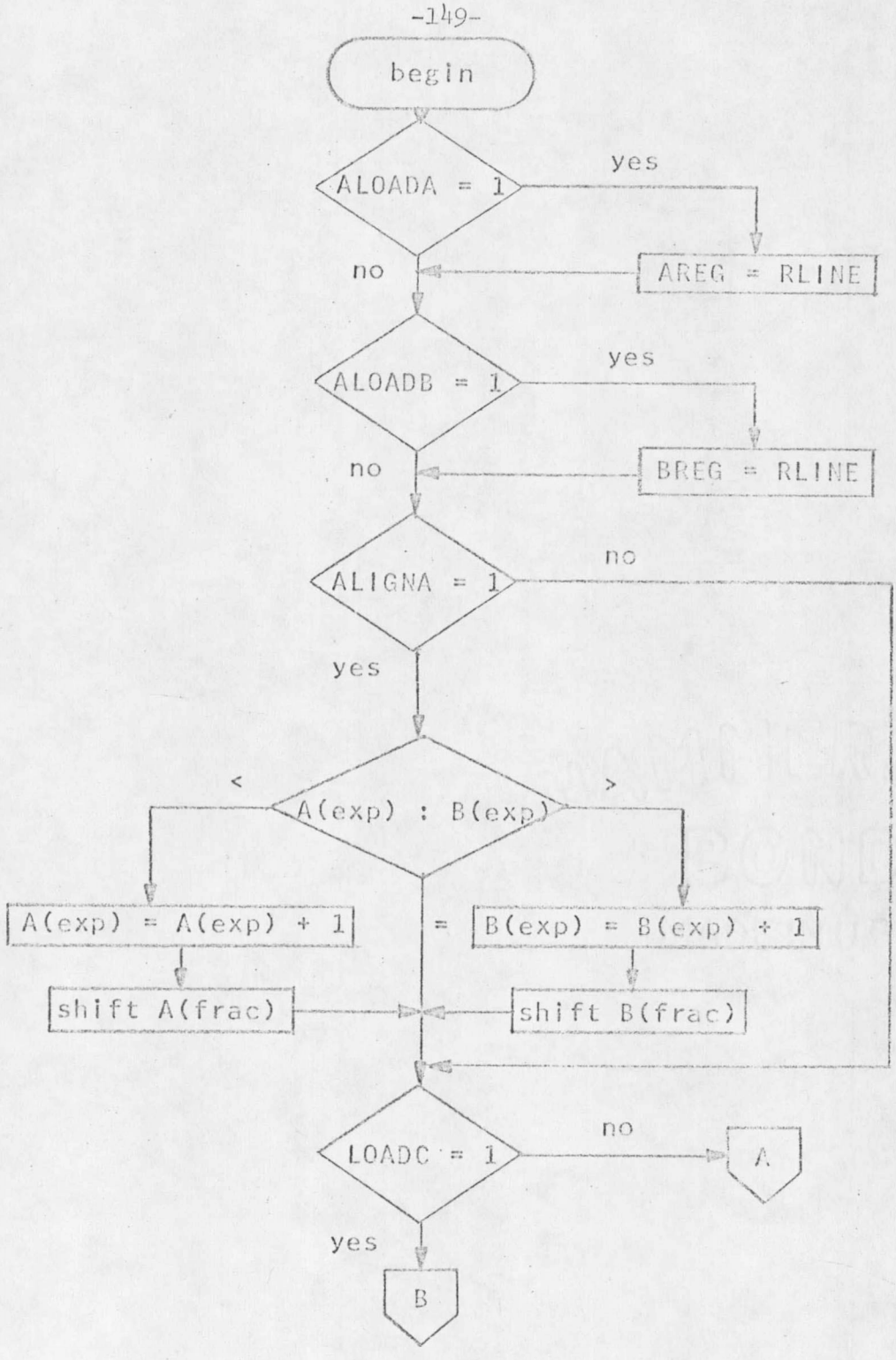


Figure 4.11. CELL FLOATING-POINT ADDER-SUBTRACTOR Flow Chart

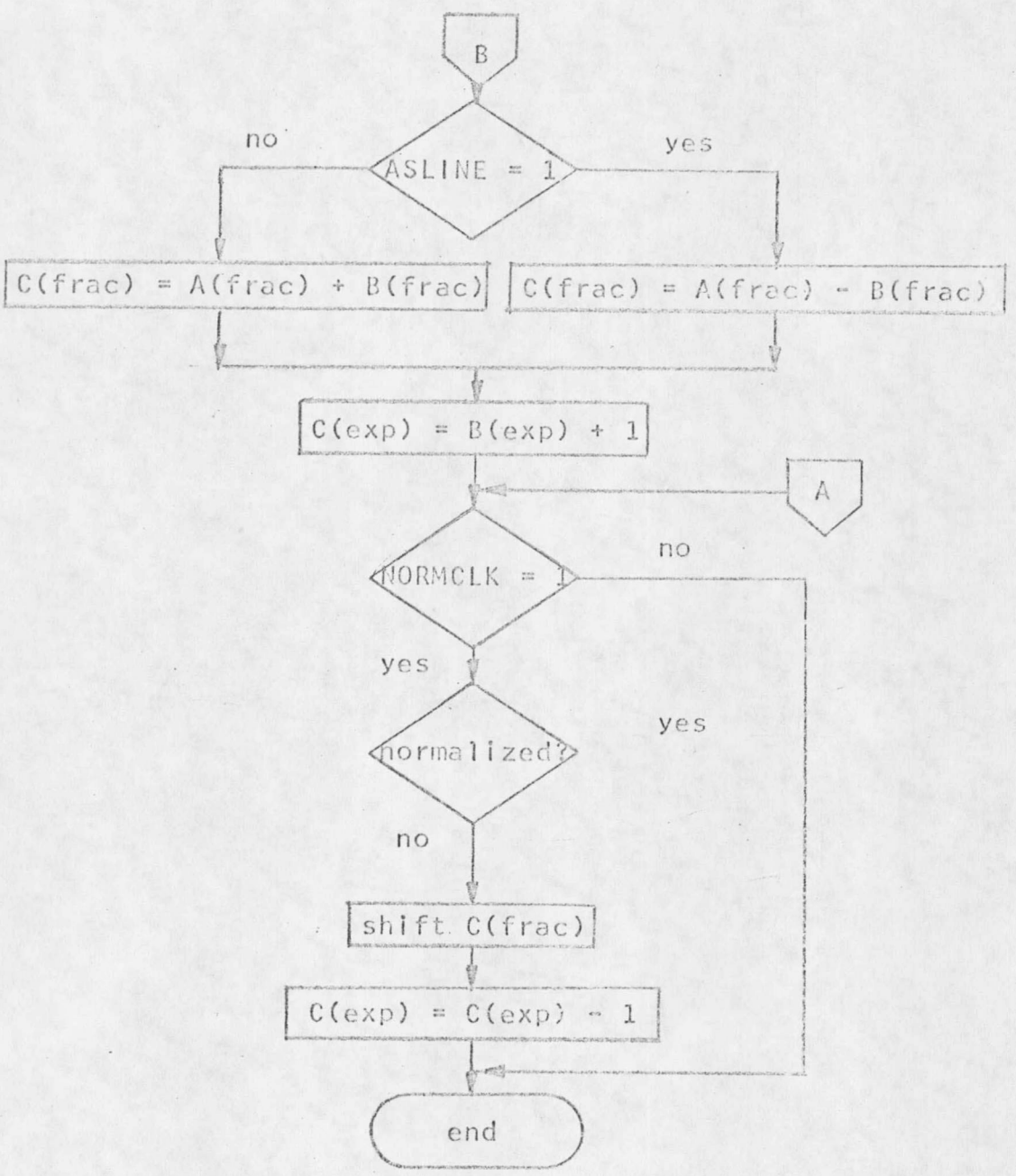


Figure 4.11. (Continued)

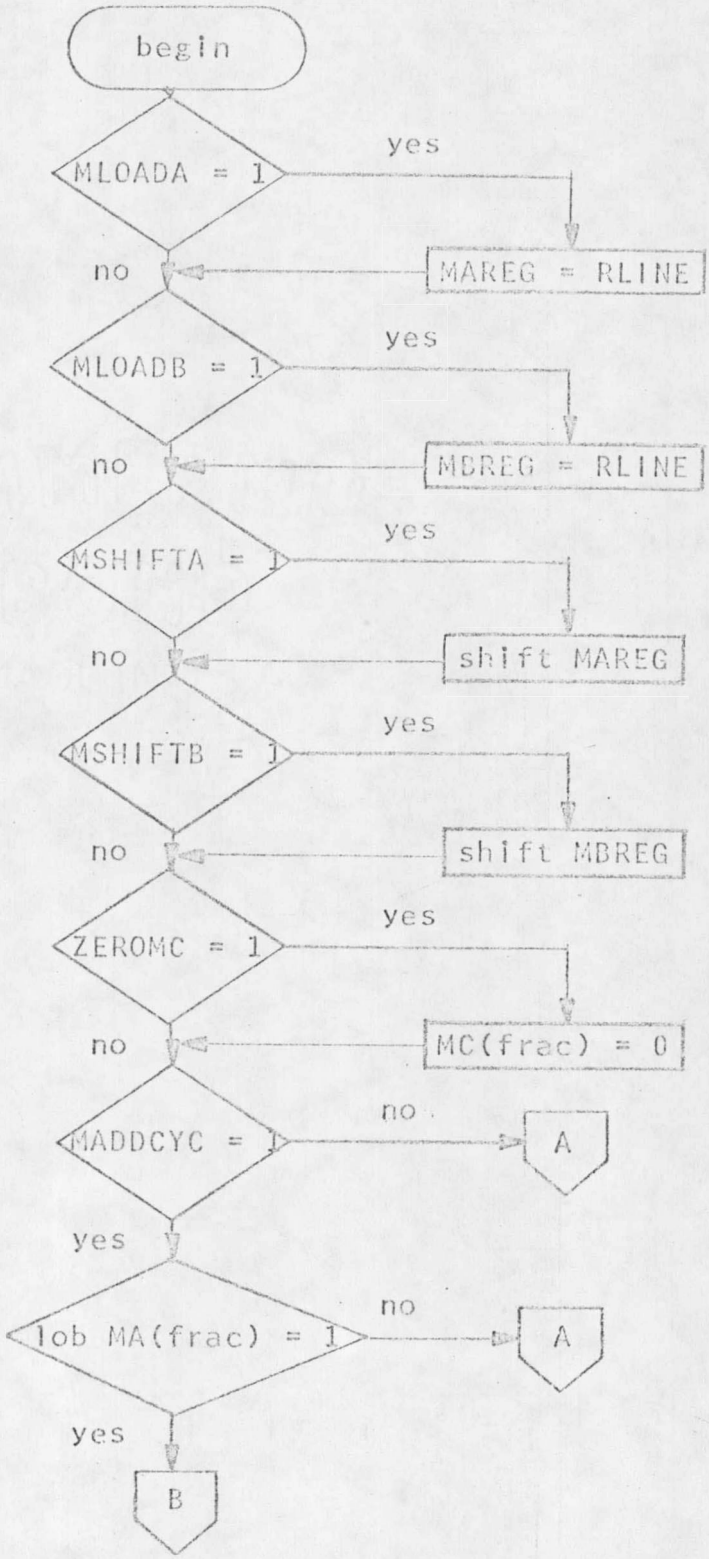


Figure 4.12. CELL FLOATING-POINT MULTIPLIER Flow Chart

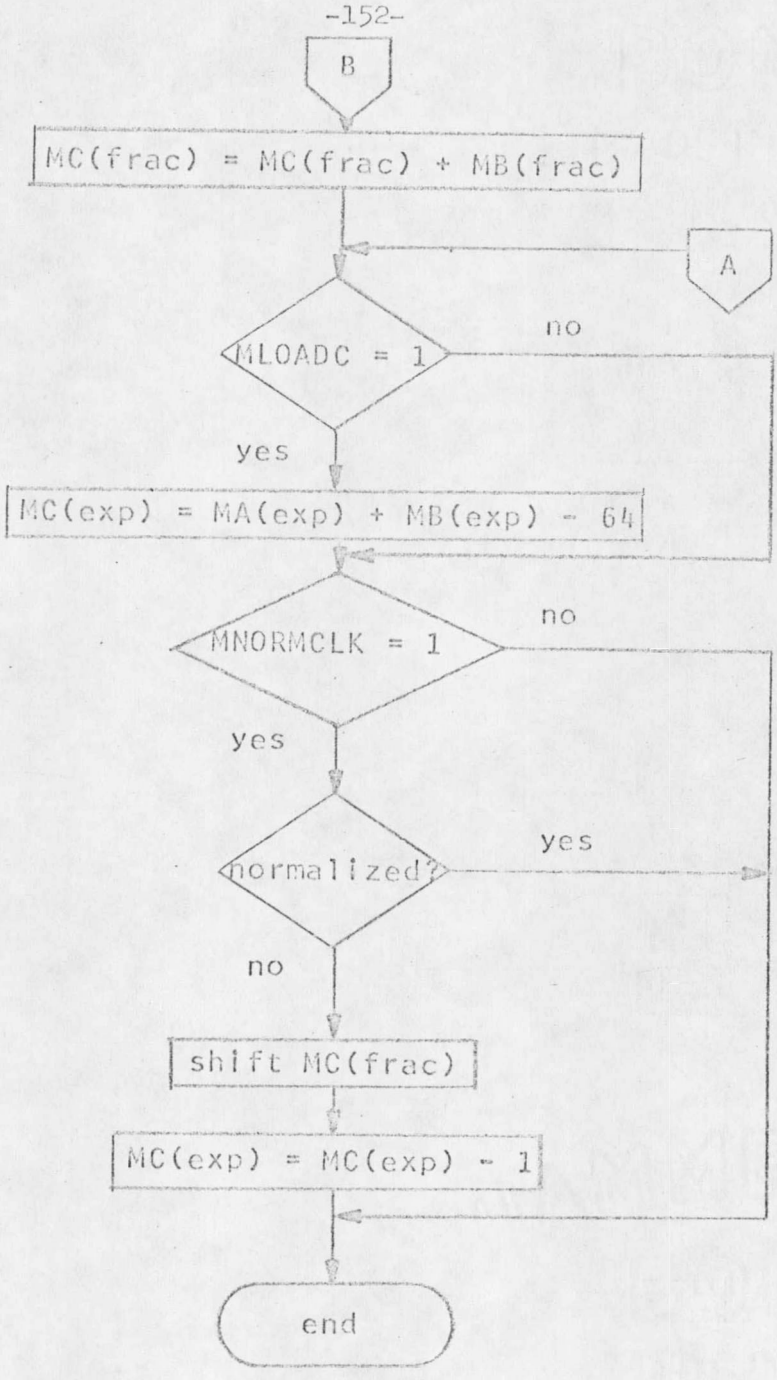


Figure 4.12. (Continued)

left one bit and a zero is stored in the low-order bit position. The C fraction register is cleared to zero if ZEROMC = 1. For the add cycle of the multiplier, if MADDCYC = 1, and the low-order bit of the A fraction register is 1, the sum of the C fraction register and the B fraction register is placed in the C fraction register. For the exponent portion of the numbers, if MLOADC = 1, the sum of the A exponent register and the B exponent register is loaded into the C exponent register and the high-order bit is complemented. If the conditions for normalization are not satisfied by the contents of the C register and if MNORMCLK = 1, the C fraction register is shifted left one bit and the C exponent register is decremented by one.

The cell simulation is now complete for the current cell. Data which were expanded previously to one bit per word are now compressed back to 32 bits per word.

When the last processor cell has been simulated the simulation program returns to the FETCH INSTRUCTION block of the flow chart of Figure 4.1.

It should be noted again that in the adder-subtractor and multiplier as well as other cell components the adding, shifting, comparing and other operations are done at the gate level. Thus, any desired circuit changes in these units should be fairly easy to implement in the logic simulation program.

4.2 Sample Run of Simulation Program

This section describes a sample run of the KF machine simulation program discussed in Section 4.1. The sample is intended to demonstrate both the routing and the arithmetic portions of the machine. Since all cells are identical, the simulation program was modified so that arithmetic operations were simulated in only one cell of the array. The cell in the last row and the last column was used.

Table 4.6 gives the sequence of instructions for this run. The function of each of the instructions may be found either in Table 4.1 or in Table 4.2. The purpose of the sequence of instructions is as follows. (1) Set up the routing control lines so that data may be transferred from the two buffers to the last column of the array, bypassing all other columns. (2) Read data into the row buffers. (3) Transfer the data to the cell in the array in which the arithmetic operations are to be simulated. (4) Store this data in cell memory location 0. (5) Repeat steps 2 and 3. (6) Store the new data in cell memory location 1. (7) Load data from cell memory location 0 into the adder-subtractor A register. (9) Align the adder-subtractor A and B registers. (10) Add and normalize the result in the adder-subtractor C register. (11) Store the result in cell memory location 2. (12) Repeat steps 7, 8 and 9. (13) Subtract and normalize the result in the adder-subtractor C register. (14) Store the result in cell memory location 3. (15) Load data from cell memory location 0 into the multiplier A register. (16) Load data from cell memory location 1

Table 4.6. Instruction Sequence for Sample Simulation Run

```
RBON          ALIN          LDMB 1        STMC 5
CCAM 8        BDIB 0 02     LDMC          RBOF
CCOM F 0      LDAC          SETI 0 31     LDRA 2
GLON          SETI 0 18     MADD          LDRB 3
RION          ANOR          SMAB          SETI 0 20
INRO          BDIB 0 02     BDIB 0 03     RTAB
SETI 0 20     STAC 2        SETI 0 18     BDIB 0 02
RTAB          LDAA 0        MNOR          OURO
BDIB 0 02     LDAB 1        BDIB 0 02     OUCO
STRA 0        SETI 0 80     STMC 4        LDRA 4
INRO          ALIN          LDMB 0        LDRB 5
SETI 0 20     BDIB 0 02     LDMA 1        SETI 0 20
RTAB          SUBT          SETI 0 31     RTAB
BDIB 0 02     SETI 0 18     SETI 0 31     BDIB 0 02
STRA 1        ANOR          SMAB          OURO
LDAA 0        BDIB 0 02     SETI 0 18     OUCO
LDAB 1        STAC 3        MNOR          $$$$
SETI 0 80     LDMA 0        BDIB 0 02
```

Table 4.7. Output from Sample Simulation Run

```
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000

0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  4.000

0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000 -3.750

0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000 -3.750
```

into the multiplier B register. (17) Multiply and normalize the result in the Multiplier C register. (18) Store the result in cell memory location 4. (19) Load data from cell memory location 1 into the multiplier A register. (20) Load data from cell memory location 0 into the multiplier B register. (21) Multiply and normalize the result in the multiplier C register. (22) Store the result in cell memory location 5. (23) Set up the routing control lines so that data may be transferred from the array to the row and column buffers. (24) Load data from cell memory location 2 into the routing A register. (25) Load data from cell memory location 3 into the routing B register. (26) Transfer the data from the cell in the array in which the arithmetic operations were simulated to the row and column buffers. (27) Output data from the row and column buffers. (28) Load data from cell memory location 4 into the routing A register. (29) Load data from cell memory location 5 into the routing B register. (30) Repeat steps 26 and 27. (31) Stop.

The input data for this sample run consisted of two column vectors. Since the arithmetic operations were simulated in only one cell of the array, all but the last component of each vector was zero. The last component of the first vector was 2.5 and that of the second was -1.5. The output from the sample run is shown in Table 4.7. This output consists of four vectors which represent the sum, difference and two products of the input vectors. Two products were taken to illustrate that the multiplication of signed numbers does not require that the

numbers be multiplied in any particular order. That is, either the multiplier or the multiplicand or both may be negative.

4.3 Results from the Simulation Studies

Using the simulation program, all parts of the KF machine which have to do with matrix operations were simulated. These include the array of processing cells, the interconnection structure and the row and column registers. The global control unit was not simulated because it performs conventional general purpose operations identical to those found in existing computers. However, some of the functions of the global control were simulated in effect if not in fact. The tasks of fetching instructions, decoding them, executing non-array instructions and setting control lines for array instructions are performed, but the actual logic which would be required to do these things was not simulated.

Various runs of the simulation program were made, performing operations such as matrix rotation, column interchange and transpose. These runs verified that the proposed interconnection structure and associated routing logic does perform these operations as expected. The same runs also verify the operation of the row and column registers, because these are used in conjunction with the routing system.

In other runs of the simulation program, sequences of instructions were used which confirmed that all of the parts of the cells functioned properly. Data were transferred to and from the cell memory and the cell registers and the arithmetic operations; add, subtract and multiply

were performed. The sample run which was discussed in Section 4.2 serves to verify these operations and to demonstrate the use of the instructions for the simulation program. The simulation program also allows sequences of the instructions to be checked for programming errors.

Since the simulation program represents the operation of the array portion of the KF machine at the gate level, variations on the logical design and the system organization may be checked without hardware realization. If, for example, it was desired to consider a different design for the cell multiplier; all that would be necessary would be to modify that portion of the simulation which simulated the multiplier. Such a modification to a hardware realization of the machine would require the modification to be made in every cell.

Since every gate is simulated for each cell of the array, and since inherently parallel operations are difficult to simulate directly in a serial fashion, the simulation program is unusually slow. In order to overcome this problem the program is capable of being modified so that only one cell is simulated. When this is done it is necessary to by-pass the cells of all but the last row or column when doing routing operations. This is because the routing portion of other cells will not be simulated. By-passing the cells can be done because this operation is done in the interconnection simulation which is independent of the cell simulation.

The simulation process could be speeded up somewhat by replacing the detailed simulation of the interconnection structure and some of

the logic by a simulation which performs the same operations but not at the gate level. This could also be done for new sections of the simulation once they have proved to work properly.

Chapter 5

SUMMARY AND CONCLUSIONS

5.1 Introduction

The problem considered in this thesis is the design of a specially organized, general-purpose computer which is able to perform any computation as is a general purpose computer but is highly efficient for operations which involve matrices. The general-purpose feature of the computer would allow it to be adapted to fit a variety of problems. The computer would probably be most appealing in situations where high-speed operation is required but where cost and size requirements prevent the use of special-purpose or high-speed general-purpose computers.

5.2 Kalman Filter Example

One application for such a computer is the class of problems which involves the use of the discrete Kalman filter. Since the discrete Kalman filter has many matrix operations, this algorithm has been chosen to serve as an example of the type of problem for which the computer is intended to be efficient. However, other applications involving matrix operations would be just as reasonable.

5.3 Cellular Computer

Based on previous work on microcellular and macrocellular arrays, the proposed computer consists of a two-dimensional, square array of processing cells and a global control unit. The global control unit contains a memory for storing instructions and data, an arithmetic unit and control circuitry for the array of processing cells. Instructions are obtained from the memory of the global control unit and

executed. The execution of an instruction is done either within the control unit or by the array of processing cells under the supervision of the control unit.

The processing cells, themselves, are each small computers which share the instruction decoding and control of the global control unit. When instructions are executed in the array of processing cells, all cells receive the same set of control signals and execute the same instruction, although each cell operates on its own set of data. Matrices are stored in the array of cells with the ij^{th} element of a matrix being stored in the ij^{th} cell of the array.

The processing cells each have a memory for storing sixteen 32-bit words, a floating-point adder-subtractor, a floating-point multiplier and logic for the routing of data to and from the cell.

Cells are connected in the array with a uniform interconnection structure which allows data to be transferred to and from the array through a set of buffer registers which connect the edges of the array to the global control unit. Matrices which are stored in the array may be rotated, skewed or transposed using the interconnection structure. Operations such as row interchange are also possible using this structure.

Since all processing cells in the array process data simultaneously when performing matrix operations, significant improvements in speed over conventionally organized computers may be achieved with the cellular computer. The multiplication of two 16 x 16 matrices, for example, is

about 250 times as fast in the cellular computer as in a conventional computer with a similar clock rate.

Each processor cell contains about 1000 logic units with approximately the same complexity as a shift register stage. In the near future it may be possible to realize a processor cell with a single LSI package. Since a major portion of the computer would consist of these identical packages, the cost per gate of the cellular computer should be considerably less than for a conventional computer.

5.4 Simulation Program

The cellular computer was simulated on a conventional general purpose computer. The simulation verifies that the logical design and the machine organization of the cellular computer work. It also allows modifications in the design of the computer to be tested without making actual hardware changes. The simulation also provides an example of the types of instruction sequences which may be written for the cellular computer.

5.5 Future Work

Before a complete hardware version of the cellular computer is attempted, several areas deserve further attention.

One of these areas involves the choice of data representation within the computer. Although some representations other than the 32-bit floating-point representation were considered, no detailed study was made of the effects of using them. Such a study would involve a comparison of the effects of several fixed-and floating-point representations on the amount of logic required for the arithmetic units

and the effects on the operating speed of the computer. Included in this study would be the effects of variations in the word size.

Other algorithms for the manipulation of matrices need to be considered. Among these would be several matrix inversion algorithms. Of particular interest is the effects of other algorithms on the interconnection and arithmetic structures of the computer.

Since the slowest operation in the cells is the multiplication, further study in the design of high-speed arithmetic units is in order. Other operations may also be improved. The routing, for example, may be done in a parallel-by-byte fashion with some additional circuitry.

An assembly language should be developed for the computer and linked to the simulation program. A complete Kalman filter problem should be programmed in this assembly language so that frequently used operations could be examined for possible further gains in operating speed.

A complete cell and its associated control circuits should be constructed of available integrated circuits to determine the feasibility of producing a hardware realization of the entire computer.

Appendix A

KALMAN FILTER EXAMPLE

COMPUTER PROGRAM LISTING

```
      DIMENSION P(4,4),Q(4,4),H(4,4),G(4,4),HI(4,4),PHI(4,4),PM1(4,4),
     1 XT(4),XHAT(4),GAMMA(4,4),U(2),XOBS(4),XSTAR(4),QGAM(4,4),XOLD(4),
     2 XACT(4),TEMP(4,4),TEMP1(4,4),TEMP2(4,4),PHIT(4,4),HT(4,4),R(4,4)
      LOGICAL IFF
 4321 READ 1000, MANY
      READ 1000,INCR
      IF(MANY)99,99,321
  321 MANY=MANY+1
      U(1)=0.
      U(2)=0.
C                     SET Q TO 0
      DO 2323 I=1,4
      DO 2323 J=1,4
 2323 Q(I,J)=0.
C                     GENERATE IDENTITY MATRIX
      DO 9090 I=1,4
      DO 9089 J=1,4
 9089 HI(I,J)=0.
 9090 HT(I,I)=1.
C                     READ STANDARD DEVIATION OF BEARING ERROR
      READ 1001,STDEV,STDEW
      PRINT 1020, STDEV
      PRINT 1019, STDEW
C                     SET TIME INCREMENT
      T=2
C                     GENERATE Q AND R COVARIANCE MATRICES
      G(1,1)=STDEW*STDEW
      Q(1,2)=0.
      Q(2,1)=0.
      Q(2,2)=STDEW*STDEW
      R(1,1)=STDEV*STDEV/(57.295780*57.295780)
C                  GENERATE PHI AND READ P MATRICES
      DO 3000 I=1,4
```

```
      DO 2000 J=1,4
      PHI(I,J)=0.
 2000 GAMMA(I,J)=0.
 3000 PHI(I,I)=1.
      PHI(1,2)=T
      PHI(3,4)=T
      DO 2001 I=1,4
 2001 READ 1025,(P(I,J),J=1,4)
C                    GENERATE GAMMA MATRIX
      GAMMA(1,1)=T**2*.5
      GAMMA(2,1)=T
      GAMMA(3,2)=T**2*.5
      GAMMA(4,2)=T
C                    GENERATE NEW Q  AS GAMMA * Q * GAMMA TRANSPOSE
      DO 777 I=1,2
      DO 777 J=1,4
      QGAM(I,J)=0.
      DO 777 K=1,2
  777 QGAM(I,J)=QGAM(I,J)+Q(I,K)*GAMMA(J,K)
      DO 888 I=1,4
      DO 888 J=1,4
      Q(I,J)=0.
      DO 888 K=1,2
  888 Q(I,J)=Q(I,J)+GAMMA(I,K)*QGAM(K,J)
C                         READ FIRST ESTIMATE OF X STAR
      READ 1001,(XSTAR(I),I=1,4)
C                    READ INITIAL X OBS AND X T
      READ 1001,(XOBS(I),I=1,4)
      READ 1001,(XT(I),I=1,4)
      DO 2707 J=1,4
 2707 XACT(J)=XT(J)-XOBS(J)
      PRINT 7000,XOBS
      PRINT 7001, XT
```

```
      PRINT 7002, XACT
      PRINT 1002, XSTAR
C                          CALCULATE ACTUAL RANGE, SPEED AND COURSE
      RACT=SQRT(XACT(1)**2+XACT(3)**2)
      ESACT=SQRT(XT(2)**2+XT(4)**2)
      IF(XT(2))7500,7501,7501
 7500 SEX1=0.
      GO TO 7502
 7501 SEX1=1.
 7502 IF(XT(4))7503,7504,7504
 7503 SEX3=0.
      GO TO 7505
 7504 SEX3=1.
 7505 CACT=ATAN(XT(2)/XT(4))*57.295780
     1+180.*(1.-SEX1*SEX3)*(1.+SEX3)
      PRINT 7550,RACT,ESACT,CACT
      PRINT 7560,U
      PRINT 7569
      DO 7580 I=1,4
 7580 PRINT 7570,(P(I,J),J=1,4)
      DO 1234 ITS=2,MANY
      ITSM=ITS-1
      IF=ITSM-INCR*(ITSM/INCR)
      IFF=.TRUE.
      IF(IF.EQ.0)IFF=.FALSE.
C                    CALCULATE X HAT AS
C                         PHI * XSTAR + GAMMA*U
      DO 4 I=1,4
      XHAT(I)=0.
      DO 3 K=1,4
    3 XHAT(I)=XHAT(I)+PHI(I,K)*XSTAR(K)
      DO 4 K=1,2
    4 XHAT(I)=XHAT(I)+GAMMA(I,K)*U(K)
```

```
C                    COMPUTATION OF ZHAT
      IF(XHAT(1))510,511,511
  510 SEX1=0.
      GO TO 512
  511 SEX1=1.
  512 IF(XHAT(3)) 513,514,514
  513 SEX3=0.
      GO TO 515
  514 SEX3=1.
  515 ANGLE=ATAN(XHAT(1)/XHAT(3))*57.295780
     1  +180.*(1.-SEX1*SEX3)*(1.+SEX3)
      ZHAT=ANGLE
C                    CALCULATE NEW X T
      DO 666 I=1,4
  666 XOLD(I)=XT(I)
      DO 8854 I=1,4
      XT(I)=0.
      DO 667 J=1,4
  667 XT(I)=XT(I)+PHI(I,J)*XOLD(J)
      GO TO 8854
      DO 668 J=1,2
  668 XT(I)=XT(I)+GAMMA(I,J)*GNOISE(.12345567)*STDEV
 8854 CONTINUE
C                    CALCULATE NEW X OBS
      DO 669 I=1,4
  669 XOLD(I)=XOBS(I)
      DO 670 I=1,4
      XOBS(I)=0.
      DO 671 J=1,4
  671 XOBS(I)=XOBS(I)+PHI(I,J)*XOLD(J)
      DO 670 J=1,2
  670 XOBS(I)=XOBS(I)+GAMMA(I,J)*U(J)
C                     CALCULATE ACTUAL X
```

```
      DO 672 J=1,4
  672 XACT(J)=XT(J)-XOBS(J)
      IF(ITS.EQ.0002)IFF=.FALSE.
      IF(ITS.EQ.0152)IFF=.FALSE.
      IF(ITS.EQ.0176)IFF=.FALSE.
      IF(ITS.EQ.0327)IFF=.FALSE.
      IF(ITS.EQ.0377)IFF=.FALSE.
      IF(ITS.EQ.0502)IFF=.FALSE.
      IF(ITS.EQ.0528)IFF=.FALSE.
      IF(ITS.EQ.0602)IFF=.FALSE.
      IF(ITS.EQ.151)GO TO 9933
      IF(ITS.EQ.176)GO TO 9933
      IF(ITS.EQ.326)GO TO 9933
      IF(ITS.EQ.376)GO TO 9933
      IF(ITS.EQ.501)GO TO 9933
      IF(ITS.EQ.526)GO TO 9933
      IF(ITS.EQ.527)GO TO 9933
      IF(ITS.EQ.601)GO TO 9933
      GO TO 9934
 9933 IFF=.FALSE.
      READ 1001,U
 9934 CONTINUE
C                    GENERATE ACTUAL TARGET TRUE BEARING
      IF(XACT(1)) 520,521,521
  520 SEX1=0.
      GO TO 522
  521 SEX1=1.
  522 IF(XACT(3)) 523,524,524
  523 SEX3=0.
      GO TO 525
  524 SEX3=1.
  525 ANGLE=ATAN(XACT(1)/XACT(3))*57.295780
     1  +180.*(1.-SEX1*SEX3)*(1.+SEX3)
```

```
      ZACT=ANGLE
C               COMPUTE OBSERVED TARGET BEARING
      ZNOISY=ANGLE+GNOISE(.12345567)*STDEV
C                    GENERATE NEW H TRANSFORMATION
      IF(XHAT(3))8002,8001,8002
 8001 H(1,1)=0.
      GO TO 8003
 8002 H(1,1)=XHAT(3)/(XHAT(3)**2+XHAT(1)**2)
 8003 H(1,2)=0.
      IF(XHAT(1))8005,8004,8005
 8004 H(1,3)=0.
      GO TO 8006
 8005 H(1,3)=-XHAT(1)/(XHAT(3)**2+XHAT(1)**2)
 8006 H(1,4)=0.
C                    CALCULATE NEW GAIN MATRIX
      CALL EMTRAN(PHI,4,4,PHIT)
      CALL EMPROD(P,PHIT,4,4,4,TEMP)
      CALL EMPROD(PHI,TEMP,4,4,4,TEMP1)
      CALL EMADD(TEMP1,Q,4,4,PM1)
      CALL EMTRAN(H,1,4,HT)
      CALL EMPROD(PM1,HT,4,4,1,TEMP)
      CALL EMPROD(H,TEMP,1,4,1,TEMP1)
      GAIN1=1./(TEMP1(1,1)+R(1,1))
      DO 110 I=1,4
  110 G(I,1)=TEMP(I,1)*GAIN1
      CALL EMPROD(G,H,4,1,4,TEMP)
      DO 111 I=1,4
      DO 111 J=1,4
  111 TEMP(I,J)=-TEMP(I,J)
      CALL EMADD(HI,TEMP,4,4,TEMP)
      CALL EMTRAN(TEMP,4,4,TEMP1)
      CALL EMPROD(PM1,TEMP1,4,4,4,TEMP2)
      CALL EMPROD(TEMP,TEMP2,4,4,4,TEMP1)
```

```
      CALL EMTRAN(G,4,1,TEMP)
      CALL EMPROD(P,TEMP,1,1,4,TEMP2)
      CALL EMPROD(G,TEMP2,4,1,4,TEMP)
      CALL EMADD(TEMP1,TEMP,4,4,P)
C                  CALCULATE X STAR AS
C                       (Z-ZHAT)*G + XHAT
      DIFF=ZNOISY-ZHAT
      IF(DIFF+180.)42,43,44
   42 DIFF=DIFF+360.
      GO TO 43
   44 IF(DIFF-180.)43,43,46
   46 DIFF=DIFF-360.
   43 DIFF=DIFF/57.295780
      DO 41 I=1,4
   41 XSTAR(I)=XHAT(I)+G(I,1)*DIFF
      IF(IFF)GOTO 8778
      X=XOBS(1)+XHAT(1)
      Y=XOBS(3)+XHAT(3)
      WRITE(107,107)XOBS(1),XOBS(3),XT(1),XT(3),X,Y
  107 FORMAT(6E12.5)
C                          CALCULATE ESTIMATED BEARING
      IF(XSTAR(1))4500,4501,4501
 4500 SEX1=0.
      GO TO 4502
 4501 SEX1=1.
 4502 IF(XSTAR(3))4503,4504,4504
 4503 SEX3=0.
      GO TO 4505
 4504 SEX3=1.
 4505 ZSTAR=ATAN(XSTAR(1)/XSTAR(3))*57.295780
     1+180.*(1.-SEX1*SEX3)*(1.+SEX3)
C                    CALCULATE ACTUAL RANGE SPEED AND COURSE
      RACT=SQRT(XACT(1)**2+XACT(3)**2)
```

```
      ESACT=SQRT(XT(2)**2+XT(4)**2)
      IF(XT(2)) 500,501,501
  500 SEX1=0.
      GO TO 502
  501 SEX1=1.
  502 IF(XT(4)) 503,504,504
  503 SEX3=0.
      GO TO 505
  504 SEX3=1.
  505 CACT=ATAN(XT(2)/XT(4))*57.295780
     1   +180.*(1.-SEX1*SEX3)*(1.+SEX3)
C                    CALCULATE ESTIMATED RANGE SPEED AND COURSE
      RSTAR=SQRT(XSTAR(1)**2+XSTAR(3)**2)
      ESSTAR=SQRT((XSTAR(2)+XOBS(2))**2+(XSTAR(4)+XOBS(4))**2)
C                COMPUTATION OF CSTAR
      IF(XSTAR(2)+XOBS(2)) 530,531,531
  530 SEX1=0.
      GO TO 532
  531 SEX1=1.
  532 IF(XSTAR(4)+XOBS(4)) 533,534,534
  533 SEX3=0.
      GO TO 535
  534 SEX3=1.
  535 CSTAR=ATAN((XSTAR(2)+XOBS(2))/(XSTAR(4)+XOBS(4)))*57.295780+
     1180.*(1.-SEX1*SEX3)*(1.+SEX3)
 8904 CONTINUE
      PRINT 1026, ITSM
   88 PRINT 1003,(XOBS(I),I=1,4)
      PRINT 1014,(XT(I),I=1,4)
      PRINT 1024,(XACT(I),I=1,4)
      PRINT 1016,(XSTAR(I),I=1,4)
      PRINT 1017,(XHAT(I),I=1,4)
      PRINT 1013, ZHAT, ZACT, ZNOISY, ZSTAR
```

```
      RERR=PACT-RSTAR
      RPER=100.*RERR/RACT
      ESERR=ESACT-ESSTAR
      ESPER=100.*ESERR/ESACT
      CERR=CACT-CSTAR
      CPER=100.*CERR/CACT
      PRINT 1009, PACT
      PRINT 1006, RSTAR,RERR ,RPER
      PRINT 1010, ESACT
      PRINT 1007, ESSTAR,ESERR, ESPER
      PRINT 1011, CACT
      PRINT 1008,CSTAR, CERR ,CPER
      PRINT 7021,(PM1(1,J),J=1,4)
      DO 7871 I=2,4
 7871 PRINT 1022,(PM1(I,J),J=1,4)
      PRINT 1018,(G(I,1),I=1,4)
      PRINT 1023,(H(1,I),I=1,4)
      PRINT 1021,(P(1,J),J=1,4)
      DO 878 I=2,4
  878 PRINT 1022,(P(I,J),J=1,4)
      PRINT1004,(U(I),I=1,2)
 8778 CONTINUE
 1234 CONTINUE
      GO TO 4321
   99 CONTINUE
      ENDFILE(107)
      END FILE(107)
      REWIND 107
      CALL EXIT
 1000 FORMAT(I4)
 1001 FORMAT(8F10.0)
 1002 FORMAT(20H  INITIAL XSTAR IS    8F14.6)
 1003 FORMAT(20H  PRESENT XOBS IS   8F14.6)
```

```
1004 FORMAT(20H  VECTOR U IS              8F14.6)
1005 FORMAT(22H  TARGET TRUE BEARING F14.6)
1006 FORMAT(25H  TARGET RANGE  ESTIMATE  F14.6,9H   ERROR  F14.6,
    XF14.8,9H PER CENT   )
1007 FORMAT(25H  TARGET SPEED  ESTIMATE  F14.6,9H   ERROR  F14.6,
    XF14.8,9H PER CENT   )
1008 FORMAT(25H  TARGET COURSE ESTIMATE  F14.6,9H   ERROR  F14.6,
    XF14.8,9H PER CENT    )
1009 FORMAT(25H  TARGET RANGE  ACTUAL    F14.6)
1010 FORMAT(25H  TARGET SPEED  ACTUAL    F14.6)
1011 FORMAT(25H  TARGET COURSE ACTUAL    F14.6)
1013 FORMAT(30H  TARGET BEARING, PREDICTION  F12.6,9H  ACTUAL
    XF12.6,9H  NOISY   F12.6,6H  EST   F12.6)
1014 FORMAT(20H  PRESENT XT  IS     8F14.6)
1016 FORMAT(20H  PRESENT XSTAR IS   8F14.6)
1017 FORMAT(20H  PRESENT XHAT  IS   8F14.6)
1018 FORMAT(20H  PRESENT GAIN IS    /4F25.10)
1019 FORMAT(38H  STANDARD DEVIATION OF   XT    NOISE  F14.6)
1020 FORMAT(38H1 STANDARD DEVIATION OF BEARING NOISE  F14.6)
1021 FORMAT(20H  PRESENT   P   IS   /4F25.10)
7021 FORMAT(20H  PRESENT  PM1  IS   /4F25.10)
1022 FORMAT(1H    4F25.10)
1023 FORMAT(20H  PRESENT H IS       /4F25.10)
1024 FORMAT(20H  PRESENT XACT IS    8F14.6)
1025 FORMAT(4E15.8)
1026 FORMAT(25H0**********STEP15,20H**********)
7000 FORMAT(20H  INITIAL XOBS  IS    8F14.6)
7001 FORMAT(20H  INITIAL XT    IS    8F14.6)
7002 FORMAT(20H  INITIAL XACT  IS    8F14.6)
7550 FORMAT(16H  INITIAL RANGE  F14.6,9H   SPEED  F14.6,10H   COURSE
   1F14.6)
7560 FORMAT(12H  INITIAL U  2F14.6)
7569 FORMAT(12H  INITIAL P   )
```

```
 7570 FORMAT(4F25.10)
      END



C         SUBROUTINE TO GENERATE N(0,1) NOISE
      REAL FUNCTION GNOISE(X)
      DATA J/987654321/
      GNOISE=0
      DO 7 K=1,12
      I=J
      J=I*65539
      IF(J)5,6,6
    5 J=J+2147483647+1
    6 Y=J
      Y=Y*.4656613E-9
    7 GNOISE=GNOISE+Y
      GNOISE=GNOISE-6.
      END



      SUBROUTINE EMADD(A,B,N,M,C)
      DIMENSION A(4,4),B(4,4),C(4,4)
      DO 1 I=1,N
      DO 1 J=1,M
    1 C(I,J)=A(I,J)+B(I,J)
      END
```

```
      SUBROUTINE EMTRAN(A,N,M,C)
      DIMENSION A(4,4),C(4,4)
      DO 1 I=1,N
      DO 1 J=1,M
    1 C(J,I)=A(I,J)
      END
```

```
      SUBROUTINE EMPROD(A,B,N,M,L,C)
      DIMENSION A(4,4),B(4,4),C(4,4)
      DO 1 I=1,N
      DO 1 J=1,L
      C(I,J)=0.
      DO 1 K=1,M
    1 C(I,J)=C(I,J)+A(I,K)*B(K,J)
      END
```

Appendix B

CELLULAR COMPUTER SIMULATION

COMPUTER PROGRAM LISTING

```
!FORTRAN
          COMMON/DATA/DATA(16)
          CALL KFSIM
          END



!FORTRAN
          SUBROUTINE INPUT
          COMMON/DATA/DATA(16)
          READ(105,105)DATA
          CALL TO
          RETURN
  105     FORMAT(8F10.0)
  108     FORMAT(8E12.5)
          ENTRY OUTPUT
          CALL FROM
          WRITE(108,108)DATA
          RETURN
          ENTRY INVERT(N)
          CALL FROM
          DATA(N)=1./DATA(N)
          CALL TO
          END



!SYMBOL
          DEF       KFSIM,BASEADR
          DEF       TIMER
          DEF       ONCE
          DEF       BUFFERS
          REF       INPUT,OUTPUT,DATA
          REF       F:IN,F:OUT,F:RAD
          REF       INVERT
SET COM,12,20,32,12,20 X'222',0,X'25200101',X'352',AF(1)
EXECUTE   COM,12,20 X'6AE',AF(1)
*****
BASEADR   EQU       $
INSTNUM   EQU       64
          BOUND     8
INSTRBUF  RES       20
          TEXT      '* ERROR'
```

```
INSTR     GEN,8,24  X'10',F:IN
          GEN,8,24  X'30',16
          DATA      INSTRBUF
          DATA      80
PUSH      GEN,8,24  X'11',F:RAD
          GEN,8,24  X'38',16
          DATA      INSTRBUF
          DATA      80
          DATA      LOADLOC
PULL      GEN,8,24  X'10',F:RAD
          GEN,8,24  X'38',16
          DATA      INSTRBUF
          DATA      80
          DATA      EXCLOC
ECHOINST  GEN,8,24  X'11',F:OUT
          GEN,8,24  X'30',16
          DATA      INSTRBUF
          DATA      80
MESSAGE   GEN,8,24  X'11',F:OUT
          GEN,8,24  X'30',16
          DATA      INSTRBUF
          DATA      88
EBCDIC    TEXT      '0123456789ABCDEF'
ERROR     CAL1,1    MESSAGE
          B         START
LOADLOC   RES       1
EXCLOC    RES       1
INDEX     RES       16
DONE      DATA      0
TEMP      DATA      0
*
** ARRAY MICRO-INSTRUCTION SET
*
*LDRA X             LOAD ROUTING A REGISTER
*LDRB X             LOAD ROUTING B REGISTER
*LDAA X             LOAD ADDER A REGISTER
*LDAB X             LOAD ADDER B REGISTER
*LDAC               LOAD ADDER C REGISTER (ADD)
*LDMA X             LOAD MULTIPLIER A REGISTER
*LDMB X             LOAD MULTIPLIER B REGISTER
*LDMC               LOAD MULTIPLIER C (EXP) REGISTER
*STRA X             STORE ROUTING A REGISTER
*STRB X             STORE ROUTING B REGISTER
*STAC X             STORE ADDER C REGISTER
*STMC               STORE MULTIPLIER C REGISTER
```

```
*ALIN                ALIGNMENT CLOCK
*ZERC                ZERO MULTIPLIER C REGISTER
*MADD                MULTIPLIER ADD CLOCK
*ANOR                ADDER NORMALIZE CLOCK
*MNOR                MULTIPLIER NORMALIZE CLOCK
*SFMA                SHIFT MULTIPLIER A REGISTER
*SFMB                SHIFT MULTIPLIER B REGISTER
*SMAB                SHIFT MULTIPLIER A AND B REGISTERS
*RTAO                ROUTING A REGISTER CLOCK
*RTAB                ROUTING B REGISTER CLOCK
*RTAB                ROUTING AND
*RTAB                ROUTING A AND B REGISTER CLOCKS
*
** NON-ARRAY INSTRUCTION SET
*DUMP X Y            DUMP MEMORY LOCATION (X,Y)
*INVR X              INVERT ROW BUFFER X
*INVC X              INVERT COLUMN BUFFER X
*RBON                ROWBACK LINE ON
*RBOF                ROWBACK LINE OFF
*CBON                COLBACK LINE ON
*CBOF                COLBACK LINE OFF
*GLON                GLOBAL LINE ON
*GLOF                GLOBAL LINE OFF
*RION                ROW INPUT LINE ON
*CION                COLUMN INPUT LINE ON
*RIOF                ROW INPUT LINE OFF
*CIOF                COLUMN INPUT LINE OFF
*TRON                TRANSPOSE LINE ON
*TROF                TRANSPOSE LINE OFF
*BDIB X Y            BACK YY ON DECREMENTING INDEX X
*BFWD XX             AHEAD XX
*BBAK XX             BACK XX
*SETI X YY           SET INDEX X TO YY
*BDZF XX             BRANCH DONE ZERO AHEAD YY
*BDZB XX             BRANCH DONE ZERO BACK XX
*RCON X Y            SET ROWCON X TO Y
*RCAM X              SET ALL ROWCOM TO X
*CCOM X Y            SET COLCOM X TO Y
*CCAM X              SET ALL COLCOM TO X
*INRO                INPUT TO ROW BUFFERS
*OURO                OUTPUT FROM ROW BUFFERS
*INCO                INPUT TO COLUMN BUFFERS
*OUCO                OUTPUT FROM COLUMN BUFFERS
*$$$$                STOP
*
```

```
*
**  ARRAY INSTRUCTIONS
INSTABLE EQU        $-1
         TEXT       'LDRA'
         TEXT       'LDRB'
         TEXT       'LDAA'
         TEXT       'LDAB'
         TEXT       'LDAC'
         TEXT       'LDMA'
         TEXT       'LDMB'
         TEXT       'LDMC'
         TEXT       'STRA'
         TEXT       'STRB'
         TEXT       'STAC'
         TEXT       'STMC'
         TEXT       'ALIN'
         TEXT       'ZERC'
         TEXT       'MADD'
         TEXT       'SUBT'
         TEXT       'ANDR'
         TEXT       'MNDR'
         TEXT       'SFMA'
         TEXT       'SFMB'
         TEXT       'SMAB'
RTA0     TEXT       'RTA0'
RTB0     TEXT       'RTB0'
RTAB     TEXT       'RTAB'
*
*
**  ARRAY INSTRUCTION CONTROL LINE TABLE
OPTABLE  EQU        $-1
LDRA     DATA       X'10000000'
LDRB     DATA       X'04000000'
LDAA     DATA       X'00800000'
LDAB     DATA       X'00400000'
LDAC     DATA       X'00100000'
LDMA     DATA       X'00020000'
LDMB     DATA       X'00008000'
LDMC     DATA       X'00002000'
STRA     DATA       X'20000000'
STRB     DATA       X'60000000'
STAC     DATA       X'A0000000'
STMC     DATA       X'E0000000'
ALIN     DATA       X'00200000'
ZERC     DATA       X'00001000'
```

```
MADD      DATA      X'00000800'
SUBT      DATA      X'00180000'
ANDR      DATA      X'00040000'
MNDR      DATA      X'00000400'
SFMA      DATA      X'00010000'
SFMB      DATA      X'00004000'
SMAB      DATA      X'00014000'
RTA0      DATA      X'08000000'
RTB0      DATA      X'02000000'
RTAB      DATA      X'0A000000'
*
**  NON-ARRAY INSTRUCTIONS
DUMP      TEXT      'DUMP'
INVR      TEXT      'INVR'
INVC      TEXT      'INVC'
RB0N      TEXT      'RB0N'
RB0F      TEXT      'RB0F'
CB0N      TEXT      'CB0N'
CB0F      TEXT      'CB0F'
GL0N      TEXT      'GL0N'
GL0F      TEXT      'GL0F'
RI0N      TEXT      'RI0N'
CI0N      TEXT      'CI0N'
RI0F      TEXT      'RI0F'
CI0F      TEXT      'CI0F'
TR0N      TEXT      'TR0N'
TR0F      TEXT      'TR0F'
BDIB      TEXT      'BDIB'
BFWD      TEXT      'BFWD'
BBAK      TEXT      'BBAK'
SETI      TEXT      'SETI'
BDZF      TEXT      'BDZF'
BDZB      TEXT      'BDZB'
RC0M      TEXT      'RC0M'
RCAM      TEXT      'RCAM'
CC0M      TEXT      'CC0M'
CCAM      TEXT      'CCAM'
INR0      TEXT      'INR0'
0UR0      TEXT      '0UR0'
INC0      TEXT      'INC0'
0UC0      TEXT      '0UC0'
STOP      TEXT      '$$$$'
*
*
**  NON-ARRAY INSTRUCTION EXECUTION
```

```
ARGGEN   LI,6      4
         SLD,2     8
         LI,5      15
         CB,2      EBCDIC,5
         BE        $+2
         BDR,5     $-2
         SCS,5     -4
         SLD,4     4
         BDR,6     $-7
         SLS,4     16
         B         *10
*
INVARG   DATA      0
INVROW   LI,7      16
         LW,2      ROWBUFF-1,7
         STW,2     DATA-1,7
         BDR,7     $-2
         BAL,10    ARGGEN
         SLD,4     -28
         AI,4      1
         STW,4     INVARG
         BAL,13    INVERT
         DATA      1
         PZE,4     INVARG
         LI,7      16
         LW,2      DATA-1,7
         STW,2     ROWBUFF-1,7
         BDR,7     $-2
         LI,2      0
         STW,2     CONTROL
         EXECUTE   CYCLE
         B         START
INVCOL   LI,7      16
         LW,2      COLBUFF-1,7
         STW,2     DATA-1,7
         BDR,7     $-2
         BAL,10    ARGGEN
         SLD,4     -28
         AI,4      1
         STW,4     INVARG
         BAL,13    INVERT
         DATA      1
         PZE,4     INVARG
         LI,7      16
         LW,2      DATA-1,7
```

```
            STW,2     COLBUFF-1,7
            BDR,7     $-2
            LI,2      0
            STW,2     CONTROL
            EXECUTE   CYCLE
            B         START
ROWBON      LI,2      1
            STW,2     ROWBACK
            B         START
ROWBOFF     LI,2      0
            STW,2     ROWBACK
            B         START
COLBON      LI,2      1
            STW,2     COLBACK
            B         START
COLBOFF     LI,2      0
            STW,2     COLBACK
            B         START
GLOBON      LI,2      1
            STW,2     GLOBAL
            STW,2     CHANGE
            B         START
GLOBOFF     LI,2      0
            STW,2     GLOBAL
            LI,2      1
            STW,2     CHANGE
            B         START
ROWON       LI,2      1
            STW,2     ROWINP
            B         START
ROWOFF      LI,2      0
            STW,2     ROWINP
            B         START
COLON       LI,2      1
            STW,2     COLINP
            B         START
COLOFF      LI,2      0
            STW,2     COLINP
            B         START
TRANSON     LI,2      1
            STW,2     TRANSP
            B         START
TRANSOFF    LI,2      0
            STW,2     TRANSP
            B         START
```

```
BRNDIB    BAL,10    ARGGEN
          SLD,4     =28
          MTW,=1    INDEX,4
          BLEZ      START
          SLS,5     =24
          LCW,5     5
          AWM,5     EXCL0C
          B         START
BRNFWD    BAL,10    ARGGEN
          SLD,4     =24
          AWM,4     EXCL0C
          B         START
BRNBAK    BAL,10    ARGGEN
          SLD,4     =24
          LCW,4     4
          AWM,4     EXCL0C
          B         START
SETINDEX  BAL,10    ARGGEN
          SLD,4     =28
          SLS,5     =24
          STW,5     INDEX,4
          B         START
BRNDZB    MTW,0     D0NE
          BNEZ      START
          BAL,10    ARGGEN
          SLS,4     =24
          LCW,4     4
          AWM,4     EXCL0C
          B         START
BRNDZF    MTW,0     D0NE
          BNEZ      START
          BAL,10    ARGGEN
          SLS,4     =24
          AWM,4     EXCL0C
          B         START
*
INPUTR0W  BAL,13    INPUT
          DATA      0
          LI,7      16
          LW,2      DATA=1,7
          STW,2     R0WBUFF=1,7
          BDR,7     $=2
          LI,2      0
          STW,2     C0NTR0L
          EXECUTE   CYCLE
```

```
            B           START
INPUTCOL    BAL,13      INPUT
            DATA        0
            LI,7        16
            LW,2        DATA-1,7
            STW,2       COLBUFF-1,7
            BDR,7       $-2
            LI,2        0
            STW,2       CONTROL
            EXECUTE     CYCLE
            B           START
OUTROW      LI,7        16
            LW,2        ROWBUFF-1,7
            STW,2       DATA-1,7
            BDR,7       $-2
            BAL,13      OUTPUT
            DATA        0
            B           START
OUTCOL      LI,7        16
            LW,2        COLBUFF-1,7
            STW,2       DATA-1,7
            BDR,7       $-2
            BAL,13      OUTPUT
            DATA        0
            B           START
SETRCOM     BAL,10      ARGGEN
            SLD,4       -26
            SLS,5       -2
            SLD,4       -2
            SLS,5       -26
            SLS,5       2
            STB,5       ROWCOM,4
            LI,5        1
            STW,5       CHANGE
            B           START
*
ALLRCOM     BAL,10      ARGGEN
            SLD,4       -30
            SLS,5       -2
            SLD,4       -28
            SLS,5       2
            LW,4        5
            LI,5        15
            STB,4       ROWCOM,5
            BDR,5       $-1
```

```
            STB,4       ROWCOM
            B           START
SETCCOM     BAL,10      ARGGEN
            SLD,4       =26
            SLS,5       =2
            SLD,4       =2
            SLS,5       =26
            SLS,5       2
            STB,5       COLCOM,4
            LI,5        1
            STW,5       CHANGE
            B           START
ALLCCOM     BAL,10      ARGGEN
            SLD,4       =30
            SLS,5       =2
            SLD,4       =28
            SLS,5       2
            LW,4        5
            LI,5        15
            STB,4       COLCOM,5
            BDR,5       $-1
            STB,4       COLCOM
            B           START
*
TRAPFPT     GEN,8,24    X'14',0
            DATA        1
KFSIM       EQU         $
            CAL1,3      TRAPFPT
BEGIN       LI,2        0
            STW,2       LOADLOC
            LI,7        64
            STW,2       RES-1,7
            BDR,7       $-1
            LI,7        8192
            STW,2       ARRAY-1,7
            BDR,7       $-1
            LI,2        X'5B'
PROG        MTW,1       LOADLOC
            CAL1,1      INSTR
            CAL1,1      ECHOINST
            CAL1,1      PUSH
            CB,2        INSTRBUF
            BNE         PROG
            LI,2        0
            STW,2       EXCLOC
```

```
START   MTW,1   EXCLØC
        CAL1,1  PULL
        EXU     TIMER
        EXU     BUFFERS
        LI,6    0
        LI,7    0
        LB,2    INSTRBUF,7
        CI,2    X'40'
        BE      $+3
        STB,2   INSTRBUF,6
        AI,6    1
        AI,7    1
        CI,7    79
        BLE     $-7
        LD,2    INSTRBUF
        CW,2    STØP
        BNE     $+2
        CAL1,9  3
        CW,2    DUMP
        BE      CØREDUMP
        CW,2    INVR
        BE      INVRØW
        CW,2    INVC
        BE      INVCØL
        CW,2    RBØN
        BE      RØWBØN
        CW,2    RBØF
        BE      RØWBØFF
        CW,2    CBØN
        BE      CØLBØN
        CW,2    CBØF
        BE      CØLBØFF
        CW,2    GLØN
        BE      GLØBØN
        CW,2    GLØF
        BE      GLØBØFF
        CW,2    RIØN
        BE      RØWØN
        CW,2    CIØN
        BE      CØLØN
        CW,2    RIØF
        BE      RØWØFF
        CW,2    CIØF
        BE      CØLØFF
        CW,2    TRØN
```

```
                BE          TRANSØN
                CW,2        TRØF
                BE          TRANSØFF
                CW,2        BDIB
                BE          BRNDIB
                CW,2        BFWD
                BE          BRNFWD
                CW,2        BBAK
                BE          BRNBAK
                CW,2        SETI
                BE          SETINDEX
                CW,2        BDZB
                BE          BRNDZB
                CW,2        BDZF
                BE          BRNDZF
                CW,2        INRØ
                BE          INPUTRØW
                CW,2        INCØ
                BE          INPUTCØL
                CW,2        ØURØ
                BE          ØUTRØW
                CW,2        ØUCØ
                BE          ØUTCØL
                CW,2        RCØM
                BE          SETRCØM
                CW,2        RCAM
                BE          ALLRCØM
                CW,2        CCØM
                BE          SETCCØM
                CW,2        CCAM
                BE          ALLCCØM
                LI,7        INSTNUM
                CW,2        INSTABLE,7
                BE          ØPFØUND
                BDR,7       $-2
                B           ERRØR
*
ØPFØUND         BAL,10      ARGGEN
                SLS,4       *16
                SLS,4       *8 *
                LW,3        ØPTABLE,7
                ØR,3        4
                STW,3       CØNTRØL
                EXECUTE     CYCLE
                B           START
```

```
*****
*
*
BLOKSIZE EQU        14
N        EQU        16
LOCATION RES        1
I        RES        1
J        RES        1
*        ARRAY      DATA
ARRAY    RES        8192
GLOBAL   DATA       0
BITADR   DATA       0
TOADR    DATA       0
CHANGE   DATA       1
CELLCOM  RES        64
*        NON-COMPRESSIBLE CELL DATA
MEMORY   D01        16
         DATA       0
RAREG    DATA       0
RBREG    DATA       0
CONTROL  DATA       0
ADDRESS  DATA       0
WLINE    DATA       0
RLINE    DATA       0
CLINE    DATA       0
MCLINE   DATA       0
COLCOM   D01        4
         DATA       0
ROWCOM   D01        4
         DATA       0
OUTBIT   D01        17
         DATA       0
INBIT    D01        17
         DATA       0
COLINP   DATA       0
ROWINP   DATA       0
ROWBUFF  D01        16
         DATA       0
COLBUFF  D01        16
         DATA       0
TRANSP   DATA       0
ACON     DATA       0
BCON     DATA       0
BCONSAVE DATA       0
ROWBACK  DATA       0
```

```
COLBACK   DATA      0
*
ALINE     EQU       RLINE
BLINE     EQU       RLINE
MALINE    EQU       RLINE
MBLINE    EQU       RLINE
RALINE    EQU       RLINE
RBLINE    EQU       RLINE
*
*         COMPRESSIBLE CELL DATA
DATABLOK  EQU       $
AREG      RES       33
BREG      RES       33
ADSUBOUT  RES       34
MAREG     RES       32
CREG      RES       34
MBREG     RES       56
MCREG     RES       57
MADOUT    RES       56
HORLINE   RES       1
INHIBIT   RES       1
NORMOUT   RES       1
NORMZERO  RES       1
STORE     RES       1
ALOADA    RES       1
ALOADB    RES       1
AASCLOCK  RES       1
ASHIFT    RES       1
BSHIFT    RES       1
ZEROMC    RES       1
MSHIFTA   RES       1
MSHIFTB   RES       1
MNORMOUT  RES       1
ALIGN     RES       1
ASLINE    RES       1
ALOADC    RES       1
NORMCLK   RES       1
MLOADA    RES       1
MLOADB    RES       1
MADDCYC   RES       1
MLOADC    RES       1
MNORMCLK  RES       1
SEL0      RES       1
SEL1      RES       1
ARTLOAD   RES       1
```

```
BRTLOAD  RES      1
ARCLOCK  RES      1
BRCLOCK  RES      1
RES      RES      100
SCD2     SCD,2    0
*
*
**  EXECUTE ARRAY INSTRUCTIONS
RETURN   RES      1
CYCLE    EQU      $
         EXU      TIMER
         LI,3     X'F0'
         AND,3    CONTROL
         SLS,3    -4
         STW,3    ADDRESS
         STW,14   RETURN
*
**  SET ROUTING CONTROLS FOR ALL CELLS
         MTW,0    CHANGE
         BE       NOCHANGE
         LI,7     15
ISETCOM  LI,6     15
JSETCOM  LB,2     ROWCOM,7
         MTW,0    GLOBAL
         BEZ      $+2
         LB,2     COLCOM,6
         LW,5     7
         SLS,5    4
         OR,5     6
         STB,2    CELLCOM,5
         AI,6     -1
         BGEZ     JSETCOM
         AI,7     -1
         BGEZ     ISETCOM
         LI,7     0
         STW,7    CHANGE
NOCHANGE EQU      $
         LI,2     X'A'
         CB,2     CONTROL
         BAZ      PAST
*
**  SET OUTPUT BITS OF ROW BUFFERS
         LI,7     16
         LW,2     ROWBUFF-1,7
         SLD,2    -1
```

```
          BDR,7       $-2
          SLS,3       -16
          LI,7        32
          STH,3       OUTBIT,7
*
**  SET OUTPUT BITS OF COLUMN BUFFERS
          LI,7        16
          LW,2        COLBUFF-1,7
          SLD,2       -1
          BDR,7       $-2
          SLS,3       -16
          LI,7        33
          STH,3       OUTBIT,7
*
          LW,2        ROWINP
          SLS,2       1
          OR,2        TRANSP
          STW,2       ACON
          LW,2        COLINP
          SLS,2       1
          OR,2        TRANSP
          STW,2       BCON
*
          LI,13       16
*
**  SET INPUT BITS FOR ROW AND COL BUFFERS
INTERCON  EQU         $
*
          LI,7        31
          LH,2        OUTBIT,7
          AI,7        2
          STH,2       INBIT,7
          AI,7        -1
          MTW,0       TRANSP
          BNEZ        $+8
          LI,6        15
          LH,2        OUTBIT,6
          SLD,2       -1
          AI,6        -1
          BGEZ        $-3
          SLS,3       -16
          LW,2        3
          STH,2       INBIT,7
          MTW,0       ROWBACK
          BEZ         $+3
```

```
          LH,2        OUTBIT,7
          STH,2       INBIT,7
          MTW,0       COLBACK
          BEZ         $+4
          AI,7        1
          LH,2        OUTBIT,7
          STH,2       INBIT,7
*
          LI,7        15
ILOOP     LI,6        15
JLOOP     LW,5        7
          SLS,5       4
          OR,5        6
          LB,2        CELLCOM,5
          SLD,2       =4
          SLS,3       =28
          AND,2       L(7)
          AND,3       L(7)
          OR,2        ACON
          OR,3        BCON
          LB,2        ACONFIG,2
          LB,3        BCONFIG,3
          STW,3       BCONSAVE
*
** FIND INPUT SOURCE FOR ROUTING A REGISTER
          B           $+1,2
          B           ACONFIG0
          B           ACONFIG1
          B           ACONFIG2
          B           ACONFIG3
*
GETINA    STW,2       BITADR
          LW,4        7
          SLS,4       4
          OR,4        6
          STW,4       TOADR
          BAL,8       MOVEBIT
*
**  FIND INPUT SOURCE FOR ROUTING B REGISTER
          LW,2        BCONSAVE
          B           $+1,2
          B           BCONFIG0
          B           BCONFIG1
          B           BCONFIG2
          B           BCONFIG3
```

```
GETINB   STW,2     BITADR
         LW,4      7
         AI,4      16
         SLS,4     4
         OR,4      6
         STW,4     TOADR
         BAL,8     MOVEBIT
*
**  ACCOUNT FOR BY-PASSED CELLS
         LW,5      7
         SLS,5     4
         OR,5      6
         LB,2      CELLCOM,5
         SLD,2     -7
         SLS,3     -28
         AND,3     L(1)
         STW,3     12
         CI,2      0
         BEZ       $+7
         LW,2      7
         SLS,2     4
         OR,2      6
         STW,2     BITADR
         BAL,8     GETBIT
         BAL,8     SAVEBIT
         CI,12     0
         BEZ       $+8
         LW,2      7
         AI,2      16
         SLS,2     4
         OR,2      6
         STW,2     BITADR
         BAL,8     GETBIT
         BAL,8     SAVEBIT
*
         AI,6      -1
         BGEZ      JLOOP
         AI,7      -1
         BGEZ      ILOOP
         BDR,13    INTERCON
ROUTING  EQU       $
*
**  SHIFT ROW BUFFERS FOR ROUTING
         LI,5      32
         LH,3      INBIT,5
```

```
          LI,7      16
          LW,2      ROWBUFF+1,7
          SCD,2     +1
          STW,2     ROWBUFF+1,7
          BDR,7     $-3
*
**  SHIFT COLUMN BUFFERS FOR ROUTING
          LI,5      33
          LH,3      INBIT,5
          LI,7      16
          LW,2      COLBUFF+1,7
          SCD,2     +1
          STW,2     COLBUFF+1,7
          BDR,7     $-3
*
PAST      EQU       $
          EXU       TIMER
          MTW,0     CONTROL
          BEZ       *RETURN
          B         DOARRAY
*
**  INTERCONNECTION STRUCTURE SERVICE ROUTINES
GETBIT    LW,4      BITADR
          SLD,4     +5
          SLS,5     +27
          LW,2      INBIT,4
          AI,5      1
          BR,5      SCD2
          EXU       5
          AND,3     L(1)
          B         *E
*
SAVEBIT   LW,4      BITADR
          SLD,4     +5
          SLS,5     +27
          LW,2      3
          LI,3      1
          SLD,2     31
          LCW,5     5
          AND,5     L(X'7F')
          BR,5      SCD2
          EXU       5
          STS,2     OUTBIT,4
          B         *E
*
```

```
MOVEBIT  LW,4      BITADR
         SLD,4     =5
         SLS,5     =27
         LW,2      OUTBIT,4
         AI,5      1
         OR,5      SCD2
         EXU       5
         AND,3     L(1)
         LW,4      TOADR
         SLD,4     =5
         SLS,5     =27
         LW,2      3
         LI,3      1
         SLD,2     31
         LCW,5     5
         AND,5     L(X'7F')
         OR,5      SCD2
         EXU       5
         STS,2     INBIT,4
         B         *8
*
**  ROUTING A REGISTER ROUTING CONFIGURATIONS
ACONFIG  DATA      X'00010202'
         DATA      X'03030303'
*
**  DIAG = 0,  ROWINP = 0,  TRANSP = 0
ACONFIGO LW,2      7
         SLS,2     4
         LW,4      6
         AI,4      =1
         BGEZ      $+2
         LI,4      X'F'
         OR,2      4
         B         GETINA
*
**  DIAG = 0,  ROWINP = 0,  TRANSP = 1
ACONFIG1 LW,2      7
         SLS,2     4
         LW,4      6
         AI,4      =1
         BGEZ      $+4
         LI,2      X'1F0'
         OR,2      7
         B         GETINA
         OR,2      4
```

```
          B         GETINA
*
**  DIAG = 0,  ROWINP = 1,  TRANSP = =
ACONFIG2  LW,2      7
          SLS,2     4
          LW,4      6
          AI,4      =1
          BGEZ      $+4
          LI,2      X'200'
          OR,2      7
          B         GETINA
          OR,2      4
          B         GETINA
*
**  DIAG = 1,  ROWINP = =,  TRANSP = =
ACONFIG3  LW,2      7
          AI,2      =1
          BGEZ      $+4
          LI,2      X'1FC'
          OR,2      6
          B         GETINA
          SLS,2     4
          LW,4      6
          AI,4      1
          AND,4     L(X'F')
          OR,2      4
          B         GETINA
*
** ROUTING B REGISTER ROUTING CONFIGURATIONS
*
BCONFIG   DATA      X'00000101'
          DATA      X'02030203'
**  DIAG = 0,  COLINP = 0,  TRANSP = =
*
BCONFIG0  LW,2      7
          AI,2      =1
          AND,2     L(X'F')
          AI,2      16
          SLS,2     4
          OR,2      6
          B         GETINB
*
**  DIAG = 0,  COLINP = 1,  TRANSP = =
BCONFIG1  LW,2      7
          BEZ       $+5
```

```
          AI,2      15
          SLS,2     4
          OR,2      6
          B         GETINB
          LI,2      X'210'
          OR,2      6
          B         GETINB
*
**  DIAG = 1,  COLINP = =,  TRANSP = 0
CONFIG2   LW,2      6
          BEZ       $+8
          LW,2      7
          AI,2      1
          AND,2     L(X'F')
          AI,2      16
          SLS,2     4
          OR,2      6
          B         GETINB
          LW,2      7
          SLS,2     4
          AI,2      15
          B         GETINB
*
**  DIAG = 1,  COLINP = =,  TRANSP = 1
CONFIG3   LW,2      6
          BEZ       $+8
          LW,2      7
          AI,2      1
          AND,2     L(X'F')
          AI,2      16
          SLS,2     4
          OR,2      6
          B         GETINB
          LI,2      X'1F0'
          OR,2      7
          B         GETINB
*
**  SCAN THROUGH PROCESSOR CELLS
DOARRAY   EQU       $
          LI,2      15
          STW,2     I
ISCAN     EQU       $
          LI,2      15
          STW,2     J
JSCAN     EQU       $
```

```
          LW,7      I
          MI,7      N
          AW,7      J
          MI,7      BLOKSIZE*N*2
          AI,7      ARRAY
          STW,7     LOCATION
*
**  EXPAND COMPRESSIBLE CELL DATA
          EXU       TIMER
          LI,7      BLOKSIZE-1
EXPAND    LW,6      7
          SLS,6     5
          LW,2      *LOCATION,7
          LI,5      32
ELOOP     LW,4      6
          AW,4      5
          SLD,2     -1
          SLS,3     -31
          STW,3     DATABLOK-1,4
          BDR,5     ELOOP
          AI,7      -1
          BGEZ      EXPAND
          LW,2      LOCATION
          AI,2      BLOKSIZE
          LI,7      0
          LW,3      *2
          STW,3     MEMORY,7
          AI,2      1
          AI,7      1
          CI,7      16
          BL        $-5
          LW,3      *2
          STW,3     RAREG
          AI,2      1
          LW,3      *2
          STW,3     RBREG
          EXU       TIMER
*
**  SET CELL CONTROL LINES
          LW,3      CONTROL
          SET       SEL0
          SET       SEL1
          SET       STORE
          SET       ARTLOAD
          SET       ARCLOCK
```

```
          SET       BRTLOAD
          SET       BRCLOCK
          SET       TEMP
          SET       ALOADA
          SET       ALOADB
          SET       ALIGN
          SET       ALOADC
          SET       ASLINE
          SET       NORMCLK
          SET       MLOADA
          SET       MSHIFTA
          SET       MLOADB
          SET       MSHIFTB
          SET       MLOADC
          SET       ZEROMC
          SET       MADDCYC
          SET       MNORMCLK
*
** CELL SELECTOR UNIT
SELECT    LW,2      SELO
          SLS,2     1
          OR,2      SEL1
          SLS,2     1
          B         $+1,2
          LW,2      RAREG
          B         SELOUT
          LW,2      RBREG
          B         SELOUT
          BAL,10    ASETCL
          B         SELOUT
          BAL,10    MCLSET
SELOUT    STW,2     WLINE
*
** CELL MEMORY UNIT
CMU       LW,5      ADDRESS
          MTW,0     STORE
          BEZ       READONLY
WRITE     LW,2      WLINE
          STW,2     MEMORY,5
READONLY  LW,2      MEMORY,5
          STW,2     RLINE
          LI,3      0
          CI,2      0
          BGEZ      $+2
          LI,3      1
```

```
            STW,3       HORLINE
*
**  CELL ROUTING UNIT
LOADAR      MTW,0       ARTLOAD
            BEZ         LOADBR
            LW,3        RALINE
            STW,3       RAREG
            LW,2        3
            SLD,2       =1
            SLS,3       =31
            LW,4        I
            SLS,4       4
            OR,4        J
            STW,4       BITADR
            BAL,8       SAVEBIT
LOADBR      MTW,0       BRTLOAD
            BEZ         ROUTER
            LW,3        RBLINE
            STW,3       RBREG
            LW,2        3
            SLD,2       =1
            SLS,3       =31
            LW,4        I
            AI,4        16
            SLS,4       4
            OR,4        J
            STW,4       BITADR
            BAL,8       SAVEBIT
*
ROUTER      EQU         $
SHIFTA      MTW,0       ARCLOCK
            BEZ         SHIFTB
            LW,4        I
            SLS,4       4
            OR,4        J
            STW,4       BITADR
            BAL,8       GETBIT
            LW,4        I
            SLS,4       4
            OR,4        J
            LB,4        CELLCOM,4
            SLS,4       =7
            CI,4        0
            BNEZ        $+6
            LW,2        3
```

```
            LW,3    RAREG
            SLD,2   -1
            STW,3   RAREG
            AND,3   L(1)
            BAL,8   SAVEBIT
SHIFTB      MTW,0   BRCLOCK
            BEZ     ADDER
            LW,4    I
            AI,4    16
            SLS,4   4
            OR,4    J
            STW,4   BITADR
            BAL,8   GETBIT
            LW,4    I
            SLS,4   4
            OR,4    J
            LB,4    CELLCOM,4
            SLS,4   -3
            AND,4   L(1)
            BNEZ    $+6
            LW,2    3
            LW,3    RBREG
            SLD,2   -1
            STW,3   RBREG
            AND,3   L(1)
            BAL,8   SAVEBIT
*
** CELL FLOATING-POINT ADDER-SUBTRACTOR
ADDER       MTW,0   ALOADA
            BEZ     ANOLOADA
            LW,2    ALINE
            LI,7    32
ASETA       LI,3    0
            SAD,2   -1
            SCS,3   1
            STW,3   AREG,7
            BDR,7   ASETA
            STW,3   AREG
ANOLOADA    MTW,0   ALOADB
            BEZ     ALIGNAB
            LW,2    BLINE
            LI,7    32
ASETB       LI,3    0
            SAD,2   -1
            SCD,3   1
```

```
            STW,3     BREG,7
            BDR,7     ASETB
            STW,3     BREG
ALIGNAB     MTW,0     ALIGN
            BEZ       LOADC
            LI,4      0
            LI,5      0
            LI,7      26
COMP        LW,2      AREG,7
            LW,3      BREG,7
            LW,1      4
            LW,6      5
            EOR,1     L(1)
            EOR,6     L(1)
            AND,1     3
            AND,6     2
            EOR,2     L(1)
            EOR,3     L(1)
            AND,1     2
            AND,6     3
            OR,4      6
            OR,5      1
            AI,7      1
            CI,7      32
            BLE       COMP
            STW,5     ASHIFT
            STW,4     BSHIFT
            OR,4      5
            EOR,4     L(1)
            STW,4     INHIBIT
            AND,4     ALIGN
            LW,2      ASHIFT
            LW,3      BSHIFT
            AND,2     4
            AND,3     4
ALIGNA      LW,4      ASHIFT
            BEZ       ALIGNB
            LI,7      24
MOVEA       LW,2      AREG,7
            STW,2     AREG+1,7
            BDR,7     MOVEA
            LI,7      32
AADD        LW,2      AREG,7
            LW,3      2
            EOR,3     4
```

```
                STW,3     AREG,7
                AND,4     2
                BEZ       ALIGNB
                AI,7      -1
                CI,7      26
                BGE       AADD
ALIGNB          LW,5      BSHIFT
                BEZ       L0ADC
                LI,7      24
M0VEB           LW,2      BREG,7
                STW,2     BREG+1,7
                BDR,7     M0VEB
                LI,7      32
BADD            LW,2      BREG,7
                LW,3      2
                E0R,3     5
                STW,3     BREG,7
                AND,5     2
                BEZ       L0ADC
                AI,7      -1
                CI,7      26
                BGE       BADD
L0ADC           MTW,0     AL0ADC
                BEZ       N0RM
ADDSUB          LI,3      0
                LI,7      26
NEXTHIGH        LW,4      BREG-1,7
                AND,4     3
                LW,5      AREG-1,7
                E0R,5     ASLINE
                LW,6      BREG-1,7
                AND,6     5
                0R,4      6
                LW,6      3
                AND,6     5
                0R,4      6
                E0R,3     AREG-1,7
                E0R,3     BREG-1,7
                STW,3     ADSUB0UT-1,7
                LW,3      4
                BDR,7     NEXTHIGH
                LI,3      1
                LI,7      7
EXPP            LW,2      BREG+25,7
                LW,4      2
```

```
            EOR,4     3
            STW,4     ADSUBOUT+25,7
            AND,3     2
            BDR,7     EXPP
            LI,7      26
CFRAC       LW,3      ADSUBOUT-1,7
            STW,3     CREG-1,7
            BDR,7     CFRAC
            LI,7      26
CEXP        LW,3      ADSUBOUT,7
            STW,3     CREG+1,7
            AI,7      1
            CI,7      32
            BLE       CEXP
NORM        MTW,0     NORMCLK
            BEZ       ACLSET
            LI,7      2
            LW,3      CREG
            AND,3     CREG,7
            BDR,7     $-1
            LI,7      2
            LW,4      CREG
            OR,4      CREG,7
            BDR,7     $-1
            EOR,4     L(1)
            OR,3      4
            STW,3     NORMOUT
            AND,3     NORMCLK
            BEZ       NOVFL
NORMSHFT    LI,7      0
            LW,3      CREG+2,7
            STW,3     CREG+1,7
            AI,7      1
            CI,7      23
            BLE       NORMSHFT+1
CEXPDOWN    LI,7      33
            LW,3      CREG,7
            EOR,3     L(1)
            STW,3     CREG,7
            CI,3      0
            BE        NOVFL
            AI,7      -1
            CI,7      26
            BGE       CEXPDOWN+1
NOVFL       LI,7      26
```

```
            LW,3        CREG,7
            STW,3       NORMZERO
ACLSET      BAL,10      ASETCL
            B           MLTPLIER
*
ASETCL      LI,7        7
SETCEXP     LW,2        CREG+26,7
            SLD,2       *1
            BDR,7       SETCEXP
SETCLINE    LI,7        25
            LW,2        CREG-1,7
            SLD,2       *1
            BDR,7       SETCLINE+1
            STW,3       CLINE
            LW,2        CLINE
            B           *10
*
**  CELL FLOATING-POINT MULTIPLIER
MLTPLIER    MTW,0       MLOADA
            BEZ         MNOLOADA
            LI,7        32
            LW,2        MALINE
MSETA       SLD,2       *1
            SLS,3       *31
            STW,3       MAREG-1,7
            BDR,7       MSETA
MNOLOADA    MTW,0       MLOADB
            BEZ         MASHIFT
            LI,7        32
            LW,2        MBLINE
MSETB       SLD,2       *1
            SLS,3       *31
            STW,3       MBREG+23,7
            BDR,7       MSETB
            LI,7        24
            STW,3       MBREG-1,7
            BDR,7       $-1
MASHIFT     MTW,0       MSHIFTA
            BEZ         MBSHIFT
            LI,7        24
            LW,3        MAREG-1,7
            STW,3       MAREG,7
            BDR,7       $-2
MBSHIFT     MTW,0       MSHIFTB
            BEZ         MCLEARC
```

```
            LI,7    0
            LW,3    MBREG+1,7
            STW,3   MBREG,7
            AI,7    1
            CI,7    47
            BLE     $-4
            LI,3    0
            STW,3   MBREG,7
MCLEARC     MTW,0   ZER0MC
            BEZ     ACCUM
            LI,7    49
            LI,3    0
            STW,3   MCREG-1,7
            BDR,7   $-1
ACCUM       MTW,0   MADDCYC
            BEZ     ML0ADCEX
            LI,3    0
            LI,7    49
ACCUMIT     LW,4    MBREG-1,7
            E0R,4   3
            E0R,4   MCREG-1,7
            STW,4   MAD0UT-1,7
            LW,4    MBREG-1,7
            AND,4   MCREG-1,7
            LW,5    MBREG-1,7
            AND,5   3
            0R,4    5
            LW,5    MCREG-1,7
            AND,5   3
            0R,4    5
            LW,3    4
            BDR,7   ACCUMIT
ADDCYC      LW,2    MADDCYC
            AND,2   MAREG+24
            BEZ     ML0ADCEX
            LI,7    49
ADDCYC0N    LW,3    MAD0UT-1,7
            STW,3   MCREG-1,7
            BDR,7   ADDCYC0N
ML0ADCEX    MTW,0   ML0ADC
            BEZ     MN0RM
MEXP        LI,3    0
            LI,7    7
MEXPIT      LW,4    MBREG+48,7
            E0R,4   3
```

```
            EOR,4       MAREG+24,7
            STW,4       MADOUT+48,7
            LW,4        MBREG+48,7
            AND,4       MAREG+24,7
            LW,5        MBREG+48,7
            AND,5       3
            OR,4        5
            LW,5        MAREG+24,7
            AND,5       3
            OR,4        5
            LW,3        4
            BDR,7       MEXPIT
            LW,3        MADOUT+49
            EOR,3       L(1)
            STW,3       MADOUT+49
            LI,7        7
CEXPON      LW,3        MADOUT+48,7
            STW,3       MCREG+49,7
            BDR,7       CEXPON
MNORM       MTW,0       MNORMCLK
            BEZ         MSETCL
            LI,7        2
            LW,3        MCREG
            AND,3       MCREG,7
            BDR,7       $-1
            LI,7        2
            LW,4        MCREG
            OR,4        MCREG,7
            BDR,7       $-1
            EOR,4       L(1)
            OR,3        4
            STW,3       MNORMOUT
            AND,3       MNORMCLK
            CI,3        0
            BEZ         MSETCL
MNORMSHF    LI,7        2
            LW,3        MCREG,7
            STW,3       MCREG-1,7
            AI,7        1
            CI,7        49
            BLE         MNORMSHF+1
MCEXPDN     LI,7        56
            LW,3        MCREG,7
            EOR,3       L(1)
            STW,3       MCREG,7
```

```
            BE          MSETCL
            AI,7        -1
            CI,7        49
            BGE         MCEXPDN+1
MSETCL      BAL,10      MCLSET
            B           ENDMLT
*
MCLSET      EQU         $
            LI,7        7
MSETCEXP    LW,2        MCREG+49,7
            SLD,2       -1
            BDR,7       MSETCEXP
            LI,7        25
MSETCFR     LW,2        MCREG-1,7
            SLD,2       -1
            BDR,7       MSETCFR
            STW,3       MCLINE
            LW,2        MCLINE
            B           *10
ENDMLT      EQU         $
*
**  COMPRESS COMPRESSIBLE CELL DATA
            EXU         TIMER
            LI,7        BLOKSIZE-1
COMPRESS    LW,6        7
            SLS,6       5
            LI,5        32
CLOOP       LW,4        6
            AW,4        5
            LW,2        DATABLOK-1,4
            SLD,2       -1
            BDR,5       CLOOP
            STW,3       *LOCATION,7
            AI,7        -1
            BGEZ        COMPRESS
            LW,2        LOCATION
            AI,2        BLOKSIZE
            LI,7        0
            LW,3        MEMORY,7
            STW,3       *2
            AI,2        1
            AI,7        1
            CI,7        16
            BL          $-5
            LW,3        RAREG
```

```
            STW,3       *2
            AI,2        1
            LW,3        RBREG
            STW,3       *2
*
            EXU         TIMER
            EXU         ONCE
            MTW,-1      J
            BGEZ        JSCAN
            MTW,-1      I
            BGEZ        ISCAN
            B           *RETURN
DUMPDATA    DATA        0
DUMPSTRT    DATA        0
DUMPEND     DATA        0
            TEXT        'MEMORY'
COREDUMP    BAL,10      ARGGEN
            SLD,4       -28
            SLS,5       -28
            XW,4        5
            MI,5        N
            AW,5        4
            MI,5        BLOKSIZE+N+2
            AI,5        ARRAY
            STW,5       DUMPSTRT
            AI,5        31
            STW,5       DUMPEND
            CAL1,3      DUMPDATA
            B           START
ONCE        B           *RETURN
SNAP        DATA        0
            DATA        OLDTIME,OLDTIME+1
            TEXT        'TIMER'
OLDTIME     DATA        -1
TIMER       BAL,12      TIME
TIME        PSW,8       *0
            LW,8        *X'4E'
            CAL1,3      SNAP
            STW,8       OLDTIME
            PLW,8       *0
            B           *12
BUFFERS     CAL1,3      BUFFER
BUFFER      DATA        0
            DATA        OUTBIT
            DATA        COLBUFF+15
```

```
          TEXT      'BUFFER'
*
*  FOR NORMAL OPERATION
* USE MODIFY TIMER,69000000
*      MODIFY ONCE,69000000
*      MODIFY BUFFERS,69000000
*
          END



!SYMBOL
          DEF       TO,FROM
          REF       DATA
TEST      DATA      X'00C00000'
RETURN    RES       1
**  CONVERT DATA TO MACHINE
**  FLOATING-POINT REPRESENTATION
TO        AI,13     1
          STW,13    RETURN
          LI,7      16
TOLOOP    LW,2      DATA-1,7
          BNEZ      $+3
          LI,5      0
          B         ENDTO
          LAW,4     2
          LI,5      0
          SCD,4     8
          AI,5      -64
          SLS,5     2
          AI,5      64
          SLS,4     -8
          LI,1      24
          CW,4      TEST
          BCS,4     $+4
          SLS,4     1
          AI,5      -1
          BDR,1     $-4
          CI,2      0
          BGEZ      $+2
          LCW,4     4
          SLS,4     7
          OR,5      4
ENDTO     STW,5     DATA-1,7
```

```
          BDR,7     T0L00P
          B         *RETURN
**  C0NVERT DATA FR0M MACHINE
**  FL0ATING-P0INT REPRESENTATION
FR0M      AI,13     1
          STW,13    RETURN
          LI,7      16
FR0ML00P  LW,2      DATA-1,7
          BNEZ      $+3
          LI,6      0
          B         ENDFR0M
          SAD,2     -7
          SLS,3     -25
          AI,3      -64
          LW,4      3
          SLD,4     -2
          SLS,5     -30
          CI,5      0
          BEZ       $+4
          AI,4      1
          AI,5      -5
          BIR,5     $+2
          B         $+3
          SAS,2     -1
          B         $-3
          AI,4      64
          LAW,6     2
          SLS,4     24
          BR,6      4
          CI,2      0
          BGEZ      $+2
          LCW,6     6
ENDFR0M   STW,6     DATA-1,7
          BDR,7     FR0ML00P
          B         *RETURN
          END
```

LITERATURE CITED

1. Barnes, G. H., et al; "The ILLIAC IV Computer," IEEE Transactions on Computers, Vol. 17, August 1968, pp. 746-757.

2. Comfort, W. T., "A Modified Holland Machine," Proceedings, Fall Joint Computer Conference, 1963, pp. 481-488.

3. Kalman, R. E., "New Methods and Results in Linear Prediction and Filtering Theory," RIAS Technical Report 61-1.

4. Brooking, M. E., W. F. King III and A. Giusti, "Survey of In-House Work on Cellular Logic," Unpublished Memorandum, AFCRL, 1965.

5. Gonzales, R. A., "A Multi-Layer Iterative Circuit Computer," IEEE Transactions on Electronic Computers, Vol. 12, December 1963, pp. 781-790.

6. Hennie, F. C., Iterative Arrays of Logical Circuits, MIT Press and John Wiley and Sons Inc., New York, 1961.

7. Holland, J. H., "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," Proceedings, Eastern Joint Computer Conference, 1959, pp. 108-113.

8. King, W. F., and A. Giusti, "Can logic arrays be kept flexible?," Electronic Design 13, May 24, 1965, pp. 57-61.

9. Kolb, R. C., and F. H. Hollister, "Bearings-Only Target Motion Estimation," Presented at the First Asilomar Conference on Circuits and Systems, Pacific Grove, California, November 1-3, 1967.

10. Lee, E. S., Associative Techniques with Complementing Flip-Flops," Proceedings, Spring Joint Computer Conference, 1963, pp. 381-394.

11. Maitra, K. K., "Cascaded Switching Networks of Two-input Flexible Cells," IRE Transactions on Electronic Computers, Vol. 11, April 1962, pp. 136-143.

12. McKeever, B. T., "The Associative Memory Structure," Proceedings, Fall Joint Computer Conference, 1965, pp. 371-388.

13. Minnick, R. C., "A Survey of Microcellular Research," Journal of the Association for Computing Machinery, Vol. 14, April 1967, pp. 203-241.

14. Minnick, R. C., et al; "Cellular Logic," Final Report, SRI Project 5087, AFCRL Contract AF 19(628)-4233; AFCRL 66-613, April 1966.

15. Minnick, R. C., "Cobweb Cellular Arrays," Proceedings Fall Joint Computer Conference, 1965, pp. 327-341.

16. Minnick, R. C., "Cutpoint Cellular Logic," IEEE Transactions on Electronic Computers, Vol. 13, December 1964, pp. 685-698.

17. Murtha, J. C., "Highly Parallel Information Processing Systems," Advances in Computers, Vol. 7, 1966, pp. 1-116.

18. Rapp, A. K., "MOS Integrated Logic Nets," Microelectronics and Large Systems, Spartan Books, Washington, 1965, pp. 129-155.

19. Slotnick, D. L., W. C. Borck and R. C. McReynolds, "The SOLOMON Computer," Proceedings, Fall Joint Computer Conference, 1962, pp. 97-107.

20. Spandorfer, L. M., "Large Scale Integration -- an Appraisal," Advances in Computers, Vol. 9, 1968, pp. 179-235.

21. Spandorfer, L. M., and J. V. Murphy, "Synthesis of Logic Functions on an Array of Integrated Circuits," Scientific Report No. 1 for UNIVAC Project 4645, AFCRL-63-528, Contract AF 19(628)2907, Sperry Rand Corp., UNIVAC Engineering Center, Bluebell, Pa., October 31, 1963.

22. Timothy, L. K. and B. E. Bona, State Space Analysis, McGraw Hill, New York, 1968.

23. Unger, S. H., "A Computer Oriented Toward Spatial Problems," IRE Proceedings, Vol. 46, October 1958, pp. 1744-1750.

24. Unger, S. H., "Pattern Detection and Recognition," IRE Proceedings, Vol. 47, October 1959, pp. 1737-1752.