

Register-transfer-level design and software simulation of a parallel processor (The three-dimensional computer)
by William Aziz Hanna

A thesis submitted to the Graduate Faculty in partial fulfillment of the requirement for the degree of DOCTOR OF PHILOSOPHY in Electrical Engineering
Montana State University
© Copyright by William Aziz Hanna (1974)

Abstract:
This thesis describes a parallel processor whose principal features are a set of primary memory, arithmetic, and I/O planes arranged to form a three-dimensional (3-D) structure.

The memory planes are also used for data manipulation but not for arithmetic operations. The structure is so arranged that data manipulation arithmetic or logic operations, and I/O operations can be significantly overlapped. Control, necessary language translation, and non-parallel functions are performed by a general purpose supervisory machine which is interfaced with the 3-D structure.

An interpreter-simulator called SIM3D has been developed to interpret and execute 3-D instructions. FORTRAN used along with in-line SYMBOL assembly language were used to write SIM3D.

The 3-D instructions have a five-field fixed format and are strongly correlated with the actual hardware structure and capability. Ordinary sequential (non-parallel) instructions are also implemented and may be intermixed in the instruction stream although they are executed by the supervisory machine.

Some popular matrix oriented problems were programmed and executed in SIM3D language; these programs illustrate the 3-D machine as an effective computation tool for real-time problems and as a means of gaining more throughput over conventional general purpose computers.

Fault-tolerant computing modes and a system design applicable to the 3-D machine architecture have been established from state-of-the-aft literature on fault-tolerant design fortified by some ideas peculiar to the specific 3-D structure.

© William Aziz Hanna 1974

All Rights Reserved

REGISTER-TRANSFER-LEVEL DESIGN AND SOFTWARE SIMULATION OF A PARALLEL PROCESSOR

(THE THREE-DIMENSIONAL COMPUTER)

by

WILLIAM AZIZ HANNA

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

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Paul E. DeKlerk
Head, Major Department

Roy M. Johnson
Chairman, Examining Committee

Henry L. Parsons
Graduate Dean

MONTANA STATE UNIVERSITY
Bozeman, Montana

March, 1974

ACKNOWLEDGEMENT

The Author wishes to extend his gratitude for the guidance, help, and encouragement rendered to him by Dr. Roy M. Johnson during the course of his graduate work. Special thanks and deep gratitude is given to Dr. Donald A. Rudberg for needed guidance and sincere help; also, Dr. Rudberg deserves credit for writing part of the simulator program included as Appendix C. The other members of the committee, Dr. Norman A. Shyne, Dr. Donald A. Pierre and Professor Robert Lord, deserve thanks for their valuable guidance. Special thanks go to Dr. Robert C. Minnick of Rice University for valuable suggestions. The Author appreciates the sincere support extended by Dr. Paul E. Uhlrich, the Head of the Electrical Engineering Department. Thanks to Mr. Douglas James for writing the Quine-McClusky program. Thanks also goes to Mrs. Marjean Smith for typing the manuscript and to Mr. Chad Groth for the drafting work. The financial support provided by the Electrical Engineering Department of Montana State University and the Office of Naval Research under contract Number N00014-67-C-0477 is gratefully acknowledged.

The Author continually thanks the Lord Jesus Christ for meeting all his needs.

To a family whose continuous encouragement in times of uncertainty gave hope, the author dedicates this thesis.

TABLE OF CONTENTS

Page

VITA..... ii

ACKNOWLEDGEMENTS..... iii

TABLE OF CONTENTS..... iv

LIST OF TABLES..... ix

LIST OF FIGURES..... x

TABLE OF ABBREVIATIONS AND NOTATIONS..... xiii

ABSTRACT..... xvii

Chapter 1: INTRODUCTION..... 1

1.1 On the Evolution of Digital Computers..... 2

1.2 A survey of Parallel Processors..... 6

1.2.1 The Unger Computer..... 6

1.2.2 The Holland Computer..... 8

1.2.3 The Comfort Computer..... 9

1.2.4 The SOLOMON Computer..... 14

1.2.5 The Gonzalez Computer (A Multilayer Iterative Circuit Computer)..... 16

1.2.6 The ILLIAC IV Computer..... 18

1.2.7 Canon's Cellular Computer..... 21

1.2.8 Mulder's Computer..... 24

1.2.9 The PEPE Computer..... 27

1.3 Why a 3-D Computer?..... 31

1.4 An Outline of Remaining Chapters..... 35

Chapter 2: HARDWARE ORGANIZATION OF THE 3-D COMPUTER..... 38

2.1 General Organization..... 39

2.2 Technological Assumptions..... 43

2.2.1 Integrated Circuits Assumptions..... 43

2.2.2 Non-conductive Interplane Transfer..... 44

2.3 Subsystem Description..... 44

2.3.1 The Memory Cube (MC)..... 44

2.3.2 The Computation Plane (CP)..... 50

2.3.3 Input-Output-Plane (IOP)..... 56

2.3.4 Control Unit and Supervisory Machine... 61

2.3.4.1 Memory - Cell - Control..... 64

2.3.4.2 The X, Y, and Z Decoders and their controlling action..... 70

2.3.4.3 Miscellaneous Counters and Registers..... 72

2.3.4.4 IOP Control..... 73

Chapter 3: MACHINE LANGUAGE FOR THE 3-D COMPUTER..... 86

3.1 Assembly Language Structure..... 87

3.2 Memory Cube Configuration and Instruction..... 89

3.2.1 Bussing and Transfer Connections..... 91

3.2.2 Instruction Inhibition, Cancellation Numbers and Cancellation Codes..... 99

3.2.3 Operational Instructions..... 103

Page

3.2.3.1 Instruction Format..... 103

3.2.3.2 Field Description..... 104

3.2.4 3-D Instruction Set..... 108

3.2.4.1 Input/Output (I/O) Instructions..... 108

3.2.4.2 Transfer Instructions..... 109

3.2.4.3 Branching Instructions..... 112

3.3 The Simulator (SIM3DMIX)..... 113

3.4 Program Organization and System Considerations 115

3.5 Use of the Simulator..... 117

3.6 Typical Program Arrangements..... 119

Chapter 4: APPLICATION OF THE 3-D MACHINE TO MATRIX ORIENTED PROBLEMS..... 122

4.1 Problem definition..... 123

4.2 A Matrix Transpose Algorithm (MTA)..... 123

4.2.1 The Algorithm..... 124

4.2.2 Example.....

4.3 A Matrix Multiplication Algorithm (MMA)..... 125

4.3.1 The Algorithm..... 131

4.4 A Matrix Inversion Algorithm (MIA)..... 134

4.4.1 The Algorithm..... 134

4.5 Search for a Maximum or a Minimum Element, of an Array, Algorithm..... 136

4.5.1 The Algorithm..... 137

Page

Chapter 5: FAULT-TOLERANT DESIGN AND OPERATION OF THE 3-D COMPUTER 139

5.1 Introduction 140

5.2 The Necessity of Redundancy 140

5.3 Evolution of Fault-Tolerant Techniques 141

5.4 Hardware Redundancy Methods 142

5.4.1 Parity Checking and Information Coding 142

5.4.2 Vote Takers and Decision Devices 144

5.5 Software Redundancy Methods 145

5.5.1 Program Status Array of Words (PSAW) 151

5.6 Software/Hardware (Hybrid) Fault-Tolerant Technique Pertinent to the 3-D Machine Organization 155

5.6.1 Hybrid by Subplane Fault-Tolerant Computing 156

5.6.2 Hybrid by Plane Fault-Tolerant Computing 157

5.7 Fault Detection and Diagnostic Routines 158

5.7.1 Fault Detection 158

5.7.1.1 Parity Error Detection 158

5.7.1.2 Vote Disagreement Detection 160

5.7.1.3 Unsuccessful Rolling Back 160

5.7.2 Diagnostic Routines 161

5.8 3-D Machine Reconfiguration 162

Page

Chapter 6: CONCLUSIONS AND RECOMMENDATIONS..... 164

6.1 Conclusions..... 165

6.1.1 Summary..... 165

6.1.2 System Hardware Design..... 165

6.1.3 Software Considerations..... 166

6.1.4 Fault-Tolerant Computing and Design..... 167

6.2 Recommendations for Future Research..... 168

Appendix A: LEGAL CONNECTION CODES OF MEMORY CUBE INTRAPLANE TRANSFER AND THEIR CORRESPONDING GATE SETTINGS..... 170

Appendix B: A QUINE-McCLUSKY MINIMIZATION ALGORITHM APPLIED TO THE GATE SETTING AS OUTPUT AND CONNECTION CODES AS INPUT TO A CONNECTION CODE COMBINATION LOGIC DECODER..... 179

Appendix C: A LISTING OF THE 3-D MACHINE SIMULATION (SIM3DMIX)..... 218

Appendix D: A PARALLEL MATRIX TRANSPOSE ALGORITHM AS IMPLEMENTED ON SIM3SMIX..... 273

Appendix E: A PARALLEL MATRIX MULTIPLICATION ALGORITHM AS IMPLEMENTED ON SIM3DMIX..... 279

Appendix F: A PARALLEL MATRIX INVERSION ALGORITHM ACCORDING TO THE GAUSS-GORDAN ELIMINATION PRINCIPLE AS IMPLEMENTED ON SIM3DMIX..... 301

Appendix G: 3-D MACHINE'S COST ESTIMATES..... 328

Appendix H: NON-CONDUCTIVE DATA TRANSFER TECHNOLOGIES..... 334

Bibliography.....340

LIST OF TABLES

Table		Page
2.1	Arithmetic Logic Operation Codes	51
3.1	Cancellation Codes	102
3.2	Execution Via Card Input	120
3.3	Execution Via Terminal	121
5.1	Number of Correcting Bits Required for Different Values of Word Size	143

LIST OF FIGURES

Figure		Page
1.1	The Unger Spatially-Oriented Computer	7
1.2	A Module of the Holland Machine	10
1.3	Modules Used in the Execution of an Instruction	11
1.4	The General Organization of the Comfort Computer	13
1.5	The Basic Organization of the SOLOMON Computer	15
1.6	The General Organization of the 4-Quadrant ILLIAC IV Computer	20
1.7	General Organization of the KF Machine	22
1.8	General Organization of Mulder's Computers	25
1.9	General Organization of the PEPE Computer	30
2.1	Physical Relations of Major 3-D Machine Subsystem	42
2.2	Intraplane Transfer Possibilities for MR1 and MR2	48
2.3	A 2x2 Memory Plane	49
2.4	Arithmetic Unit of One Zone/Cell of CP	54
2.5	Input/Output Processor	58
2.6	VCSR Clock Sequence Signals for 1, 2, 4, 8 and 16 Column I/O Operation	59
2.7	Control and Data Flow Diagram of the 3-D Machine	62
2.8	Control and Supervisory Structure	65
2.9	The Horizontal, Vertical, Minor-Diagonal and Inter-Cell Connections of MR1	67
2.10	Bus to Bus Connections	69

Figure		Page
2.11	Cancellation Logic	71
2.12	Sequence of MCSR Signals	74
2.13	Sequence of HCSR Signals, Showing Column IC only having a Value of 1	76
2.14	Schematic Diagram of a Typical VCSR or HCSR Connection	77
2.15	A typical Logic Diagram for VCSR or HCSR	79
2.16	The Logic Diagram for the Control Signals of IOP	80
2.17	A Transducer/Cell IOP	81
2.18	An n Times Faster IOP than shown in Figure 2.7	82
2.19	Time-Multiplexed IOP Organization	84
2.20	Schematic Diagram of a Zones IOP	85
3.1	A 2x2 Memory Plane	89
3.2		90
3.3	Register Connection Codes	92
3.4	A 4x4 Matrix Transpose Pattern	98
3.5a	Operational Instruction Format	103
3.5b	A Typical Operational Instruction	103
4.1	First Step of Matrix Transpose Data Flow	126
4.2	Second Step of Matrix Transpose Data Flow	127
4.3	Third Step of Matrix Transpose Data Flow	128
4.4	Fourth Step of Matrix Transpose Data Flow	129
4.5	Fifth Step of Matrix Transpose Data Flow	130

Figure		Page
4.6	Step 3 of MMA	132
4.7	Step 4 of MMA	133
5.1	TMR Subunit	146
5.2	TMR Subunit with TMR Vote Takers	147
5.3	NMR Subunit	148
5.4	An NMR Subunit with N Vote Takers	149
5.5	An HMR (3, 5) Subunit	150
5.6	PSAW Format	153
5.7	Flow Chart of a Hybrid-by-plane Fault-Tolerant Computing System	159
5.8	Fault-Tolerant Active Subplane of the 3-D machine	163
H.1a	Non-conductive Transfer System Structure	337
H.1b	Laser Writing Scheme	337
H.1c	Laser Reading Scheme	338

TABLE OF ABBREVIATIONS AND NOTATIONS

Notation	Meaning	Page first encountered
ALO	Arithmetic logic operation	88
ALOC	Arithmetic logic operation code	47
BRNF	Branch on a floating point value, (branching is accomplished if each element is less or equal to that value	113
BRNU	Branch unconditionally	112
BRNZ	Branch on zero, (branching is accomplished if each element is equal to zero)	112
CC	Cancellation code	47
CN	Cancellation number	41
CONC	Condition code of a computation plane cell	152
CP	Computation plane	40
CPA	Computation plane's A register	53
CPB	Computation plane memory buffer register	53
CPQ	Computation plane's Q register	53
CPU	Central processing unit	151
CR	Connection register read mostly store	40
CTRN	Conductive intraplane transfer	109
D	Diagonal intraplane transfer	124

Notation	Meaning	Page first encountered
DTRN	Conductive intercell transfer	109
FFT	Fast Fourier transform	3
G_i	Gate i for bidirectional bussing control	66
G_{id}	Gate i for data bussing control into the cell	66
G_{is}	Gate i for data bussing from the cell control	66
GP	General purpose computer	114
HCSR	Horizontal control shift register	57
HCSR(I)	Gate I driven by the horizontal control shift register	73
IBS	Input buffer stack	57
IC	Index of the first column of an array	70
ICC	Interconnection code	47
II	Index of the source plane	70
INPT	A parallel input instruction	105
IOBR	Input/output buffer register	57
IOP	Input/output plane	40
IR	Index of the first row of an array	70
I/O	Input/output	4
JC	Index of the last column of an array	70
JJ	Index of the destination plane	70

Notation	Meaning	Page first encountered
JR	Index of the last row of an array	70
LED	Light emitting diode	335
LSI	Large scale integration (certain integrated circuit techniques)	43
MAX	A call for a processor to find the maximum element of an array	220
MC	Memory cube	39
MIA	Matrix inversion algorithm	134
MMA	Matrix multiplication algorithm	125
MMESA	Search for a maximum or a minimum element algorithm	136
MP	Memory plane	39
MRC	Memory cell's control register	47
MRT	Memory cell's interplane transfer buffer register	47
MR1	Memory cell's register number one	46
MR2	Memory cell's register number two	46
MTA	Matrix transpose algorithm	123
OBS	Output buffer stack	57
OP	Operation	41
OUTP	A parallel output instruction	109
PEPE	Parallel element processing ensemble	27
PSAW	Program status array of words	151

Notation	Meaning	Page first encountered
PSDW	Program status doubleword	151
SIM3D	Assembly language for the three dimensional computer	87
SIM3DMIX	A working version of SIM3D which allows inline FORTRAN and/or assembly language intructions	113
SPSD	Supervisor machine program status doubleword	151
TRAN	Interplane transfer instruction	70
VCSR	Vertical control shift register	57
VCSR(I)	Gate I driven by the vertical control shift register	73
X	Cell contents are not subject to intra-plane transfer	125
X-Decoder	Row activation decoder	63
Y-Decoder	Column activation decoder	63
Z-Decoder	Plane activation decoder	63
3-D	Three dimensional	31
o	Cell contents are interplane transferred using a rectangular path	124
$\lceil x \rceil$	Ceiling of x	124

ABSTRACT

This thesis describes a parallel processor whose principal features are a set of primary memory, arithmetic, and I/O planes arranged to form a three-dimensional (3-D) structure. The memory planes are also used for data manipulation but not for arithmetic operations. The structure is so arranged that data manipulation arithmetic or logic operations, and I/O operations can be significantly overlapped. Control, necessary language translation, and non-parallel functions are performed by a general purpose supervisory machine which is interfaced with the 3-D structure.

An interpreter-simulator called SIM3D has been developed to interpret and execute 3-D instructions. FORTRAN used along with in-line SYMBOL assembly language were used to write SIM3D. The 3-D instructions have a five-field fixed format and are strongly correlated with the actual hardware structure and capability. Ordinary sequential (non-parallel) instructions are also implemented and may be intermixed in the instruction stream although they are executed by the supervisory machine.

Some popular matrix oriented problems were programmed and executed in SIM3D language; these programs illustrate the 3-D machine as an effective computation tool for real-time problems and as a means of gaining more throughput over conventional general purpose computers.

Fault-tolerant computing modes and a system design applicable to the 3-D machine architecture have been established from state-of-the-art literature on fault-tolerant design fortified by some ideas peculiar to the specific 3-D structure.

CHAPTER 1

INTRODUCTION

1.1 On The Evolution of Digital Computers

The evolutionary development of electronic computing machines has been strongly influenced by the state-of-the-art in electronics and other components that make up the machines, coupled with the demands of the applications to which the machines will be put.

Limitations on applications of machines have usually reflected limitations in the state-of-the-art and a high cost-to-performance ratio. Components must be available, dependable, inexpensive, and reproducible with a high degree of uniformity. The history of computing machinery reflects this state of affairs quite clearly. In the evolution of what are called first generation computers, memories were slow(5) [1] (magnetic drums with 200 ms access time), arithmetic units were slow (mainly relay type logic), and because of the expense, general registers were not in wide use. In addition, the size of the components used in the first generation machines and their heat generation precluded the construction of large, fast machines with great capabilities. Such machines were not forthcoming until technological advances led to second generation machines.

Second generation machines had a construction which was typically a tube-transistor combination, thereby making smaller

---

1. Numbers in parenthesis refer to references listed in the Bibliography.

machines with less heat generation. Memories were faster and more compact because random access magnetic core memory had been recently introduced (5). Input/Output devices became faster and more reliable, and the use of multiple, general-purpose registers came about. Speed increases thus made the use of such computers for real-time, on-line problems practical in some instances. Organizationally, the basic Von Neuman programming concept and memory allocation concept were still employed. Machines were serially organized, and memory was accessed sequentially, one word at a time. Data transfers between various locations in memory were also accomplished serially; throughput increases were possible only through increases in individual element speeds. It was not that parallel processing was not considered but rather the problem was still one of excessive element size and the expense of modular replication. Examples (22) of real-time applications are: air traffic control in congested areas, the steering of electronic radar systems, filtering and control applications in complex systems, and signal identification of such nature as is shown in the Fast Fourier Transform (FFT). During the mid-1960's, rapid advances in solid-state technology occurred, and the computers constructed during the period profited as a result. The IBM 360 series came into being as did the RCA Spectra 70 and the XDS Sigma series. These as well as other machines represent the so-called third generation of computers, and have usually the

same sort of characteristic as the original Von Neuman machines; that is, a single processing unit which has access to a sequential memory. Some spreading of functions throughout the structure of the computing machine was however, becoming evident. Tasks that previously had been handled by the central processor were now being handled by peripheral devices, e.g., input/output processors were now doing data formatting. Multiport access to memory allowed overlapping I/O with other operations. It was also during this period that multiprocessing gained popularity. The sharing of resources of the entire machine (memory resources, central processor resources and input/output resources) between different programs (Multiprogramming) was employed. This stripping away of tasks from the central processor was indicative of an overworked processor, an obvious bottleneck in throughput. No matter how much the speed of this single processor element was increased, there were natural limitations on execution speeds. Thus, many workers in the field became interested in parallel processing, that is, the execution of more than one arithmetic or logical operation simultaneously. The idea was not particularly new, but it had not seen great application in earlier computer designs, principally due to cost and size problems, but new technology was making modular replication more feasible. Although certainly not all problems to which one would address an automatic calculating machine have sufficient parallelism inherent

in them to make a parallel machine cost effective, there is a large enough number of such problems to make the study of parallel operating machines a worthwhile undertaking. In fact, major portions of the aforementioned applications in radar, signal processing, control, and filtering fall in this category. Therefore, a large amount of interest came about in the design and evaluation of parallel operating machines. The parallel processors which are usually considered as milestones in the field are: The Unger Spatially Oriented Computer (45), The Holland Computer (24), The Comfort Computer (12), The SOLOMON Computer (44), The Gonzales Iterative Circuit Computer (17), The ILLIAC IV Computer (4), and the Cannon-Mulder Array Computers (9, 32). In essence, a parallel processor is viewed as a fast machine from slow parts. Although the number of operations carried out per second by each of the individual arithmetic units may not be particularly high, when one considers the totality of operations being carried out, the equivalent sequential instruction rate can be exceptionally high. It turns out, therefore, that when appropriate problems are addressed, a parallel processor can be an effective means of attacking a problem if one utilizes a large number of identical cells which may have elementary arithmetic and logic capabilities built into them, and which are interconnected in such fashion as appropriate to the class of problems under consideration.

1.2 A Survey of Parallel Processors

The following is a chronological survey of parallel processors which are relevant to the philosophy of the 3-D machine's organization and is not intended to be an exhaustive survey of array machines.

1.2.1 The Unger Computer

The Unger Computer is a computer oriented towards spatial problems(45). By a spatial problem it is meant a problem which can be broken down into a multidimensional array of identical subproblems, e.g. partial differential equations, pattern recognition, and random signal processing. Unger's computer is a very effective tool for parallel processing binary patterns in two dimensions. The Unger computer consists of a master control unit, and a rectangular array of processing modules, as in Figure 1.1. Each module communicates with its four nearest neighbors and receives orders from the master control unit. In order to simplify the hardware, the master control unit cannot address the modules individually, but issues general orders which go to all of the modules. A module unit consists of a one-bit accumulator, a small amount of random access memory (six bits in one bit words), and some associated logic. Inputs to each module come from the master control and from the accumulators of the nearest neighbor modules. If the accumulators of all modules contain zero, a logical adder (or gate) with an input

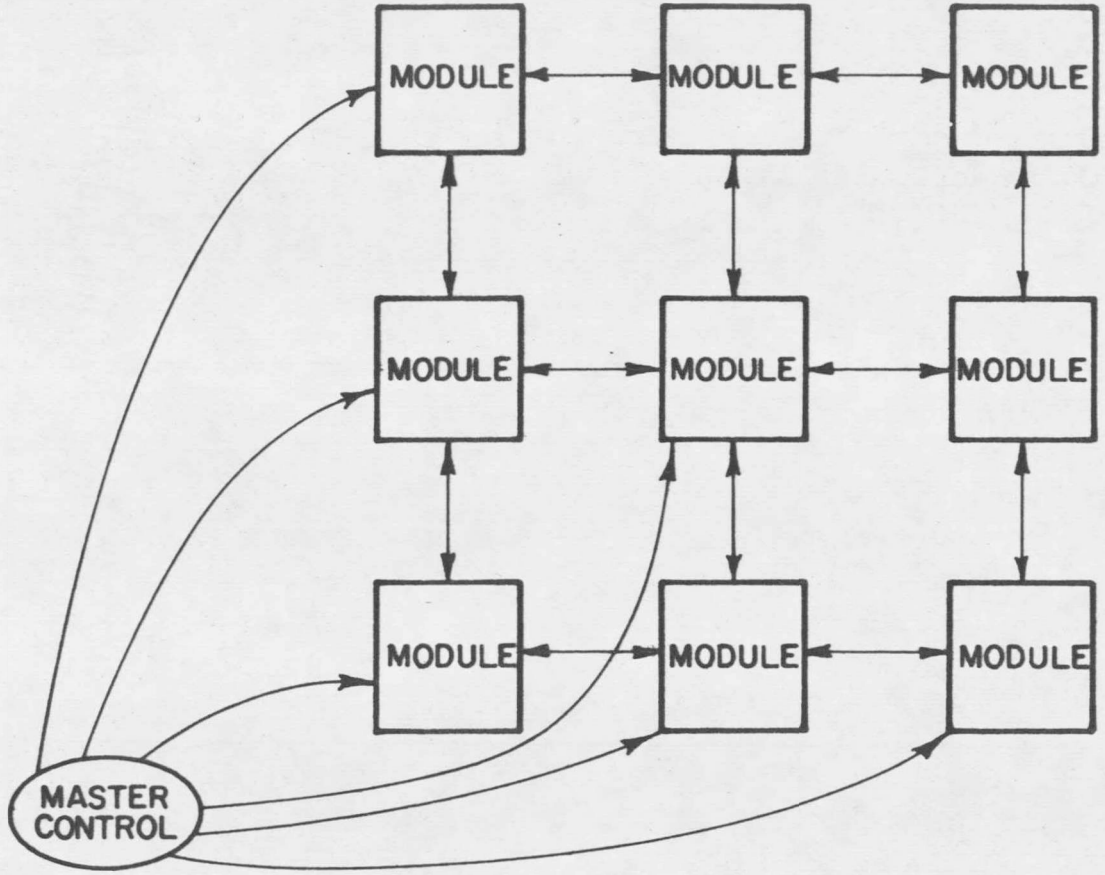


Figure 1.1 The Unger Spatially-Oriented Computer.

from each module signals that condition back to the master control unit. This instruction is analogous to the conditional transfer order used in conventional machines.

The master control is capable of performing a single instruction in all cells at once which simplifies its hardware design and structure. This simple design of the master control unit is costly in that lengthy, complicated programming of the machine is subsequently necessary.

1.2.2 The Holland Computer

The purpose of the Holland computer organization was to provide a basis for investigation into computability and the theory of automata. A computer organization described by Holland (24), established system control at the local level in the processing array. This organization is in direct contrast to the central control concept suggested by Unger.

It consists of a two-dimensional array of identical modules, each module containing a storage register, routing logic, and auxiliary registers. At any given time a module will be active or inactive. If the module is active, it treats its contents as an instruction and proceeds to execute 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

in the array. With this computer structure, sequences of instructions are arranged spatially throughout the array of modules, with an arbitrary number of sequences being executed at any given time. The operation cycle of this computer consists of the following three phases:

1. Module storage registers are set to value provided by an external source at the same time that a logical path is established. A path is a chain of modules which links two non-adjacent active modules.
2. Active modules receive operands dependent upon the topology of the logical path enabled. See Figure 1.2.
3. Instructions stored in all active modules are executed.

The Holland machine is difficult to program efficiently, and a large amount of hardware is required to accomplish a non-trivial computing task. However, it introduces the concept of an array of identical, locally controlled modules which are interconnected to form a computer.

Figure 1.2 shows a single module of the Holland computer, while Figure 1.3 shows modules used in the execution of an instruction.

1.2.3 The Comfort Computer

Comfort's proposed computer (12) is a modification of the Holland computer, described above, which attempts to accomplish

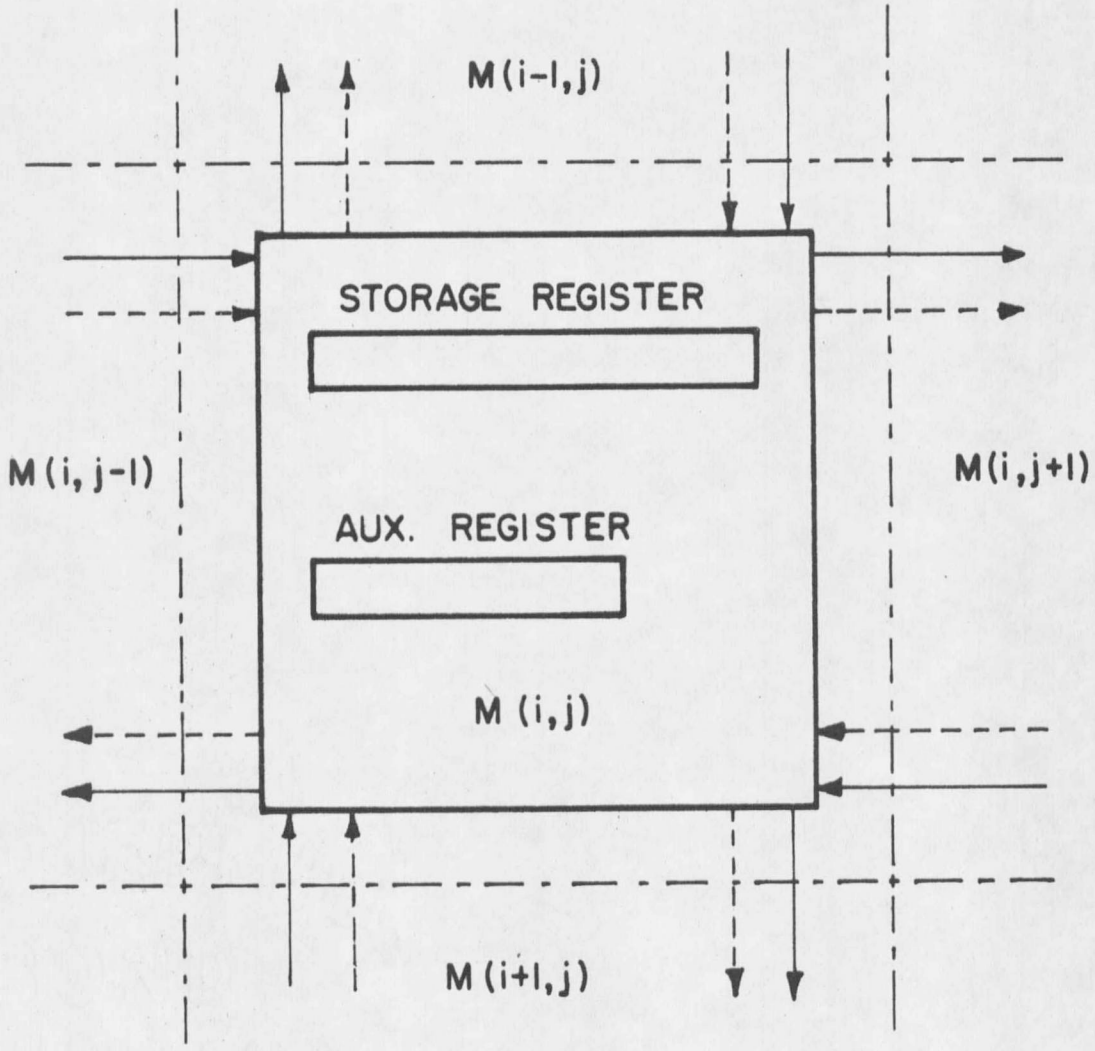


Figure 1.2 A Module of Holland Machine.

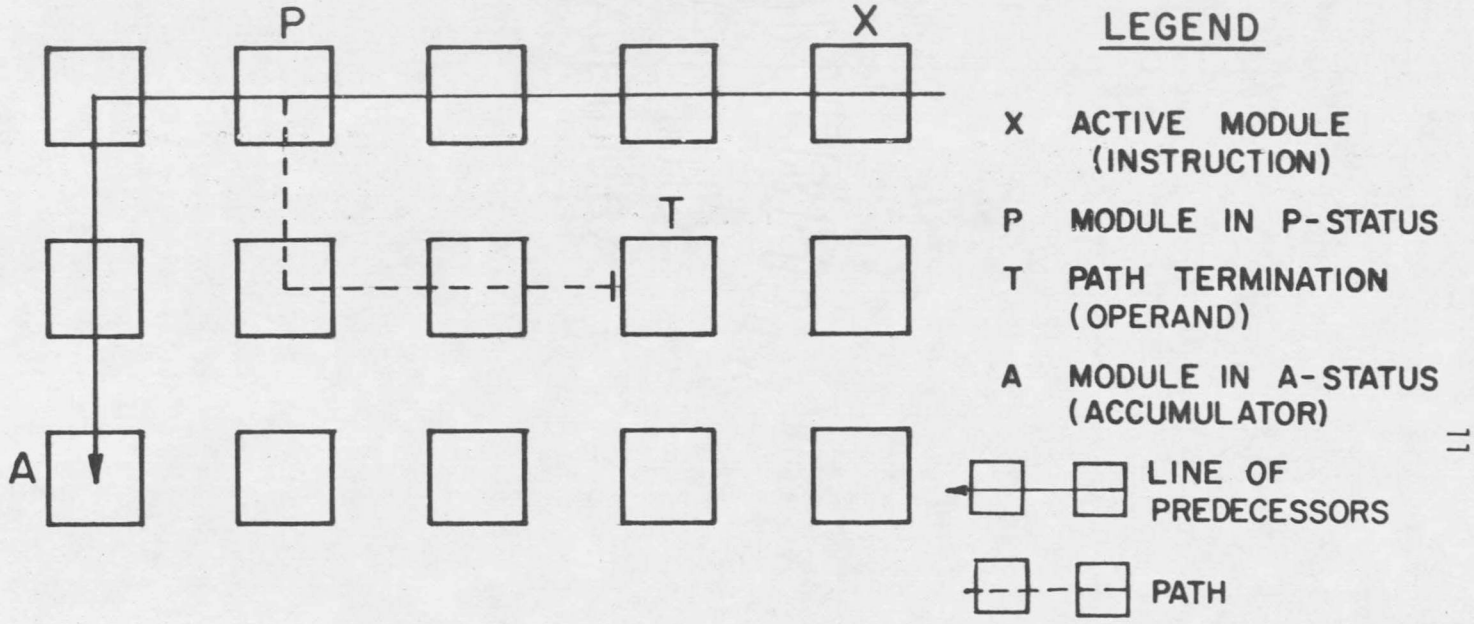


Figure 1.3 Modules Used in the Execution of an Instruction.

the following goals:

1. Reduction in machine programming complexity, thereby increasing its practical ability to solve problems.
2. Reduction of the hardware required to implement the machine without sacrificing its theoretical computing power.

The Comfort machine retains three basic features of the Holland computer which are:

1. A parallel network of computing elements.
2. Sequencing of instructions which are arranged spatially throughout the array of modules with an arbitrary number of sequences being executed at any given time.
3. The method of data accessing.

The Comfort computer is fundamently an array of modules each of which consists of two essential parts:

1. A memory and control section,
2. A communication section which controls communication with the four nearest neighbors.

The array is rectangular with a set of arithmetic units at one side. See Figure 1.4.

The A-Boxes in Figure 1.4 are basically fixed-point arithmetic units containing an accumulator, multiplier, and operation decoder. An instruction sequence must start from an A-box. The results of an operation are returned to a specified element of the array or

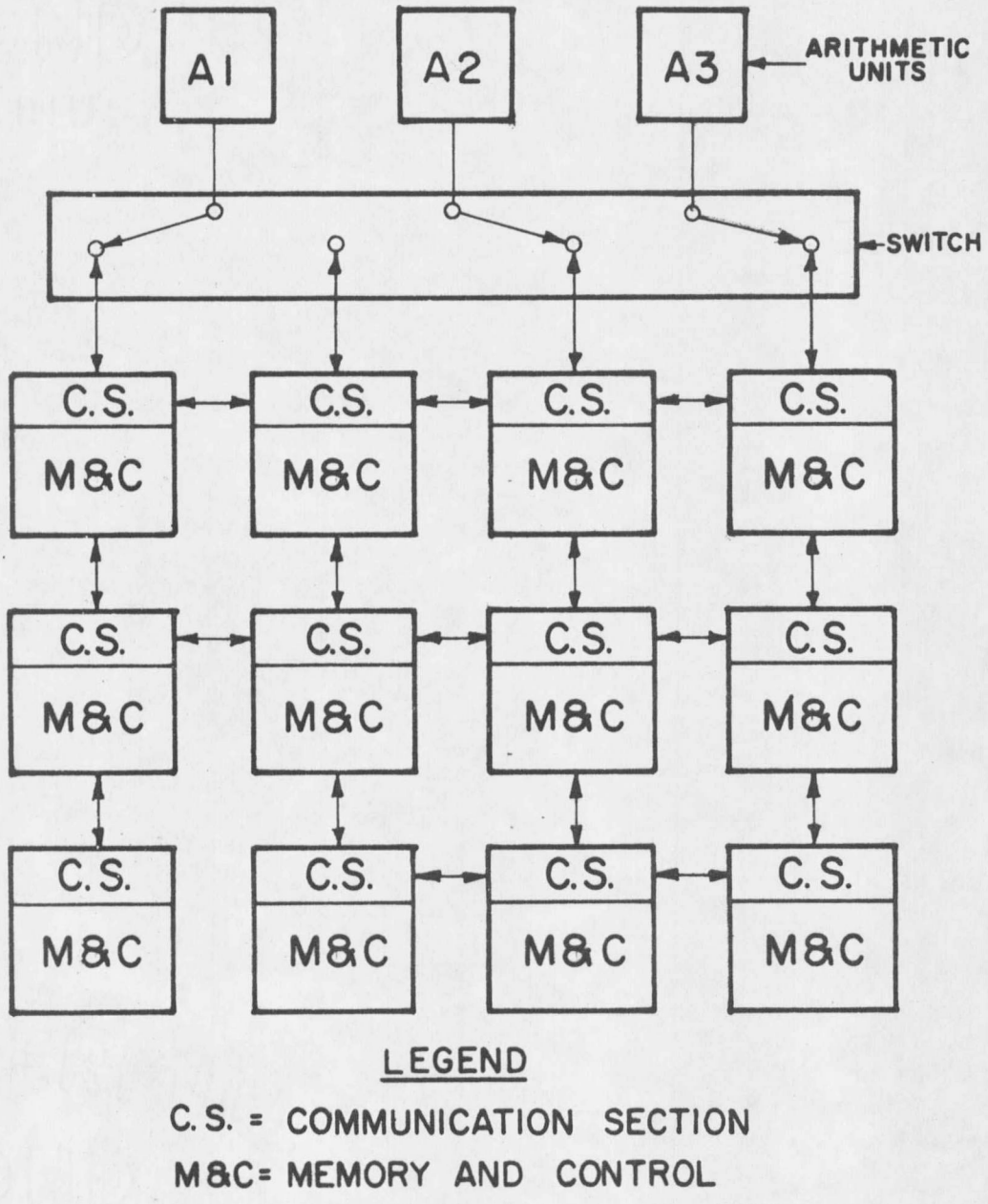


Figure 1.4 The General Organization of the Comfort Computer.

retained in the A-box. Comfort utilized the notion of the line of successors and the line of predecessors which was introduced by Holland. The notion of path building first introduced by Holland prevails in the Comfort machine also.

A basic weakness of the Comfort computer is the fact that the path building operation is done one segment at a time. Also, a large number of the modules are serving the sole purpose of being part of a path to the termination module.

1.2.4 The SOLOMON Computer

The SOLOMON (Simultaneous Operation Linked Ordinal Modular Network) Computer is one of the most interesting parallel processors to study because it was not only proposed, but also a prototype has been built (44).

The SOLOMON computer consists of three major units (7), as shown in Figure 1.5. These units are: the processing element (PE) network, the network control unit (NCU), and the input-output unit (IOU). The basic system described consists of: an array of 32 x 32 PE networks, an NCU with two control and arithmetic subunits, and an IOU which consists of five data channels. The system can be reconfigured by the programmer into four 16 x 16 arrays or sixteen 8 x 8 arrays, etc.

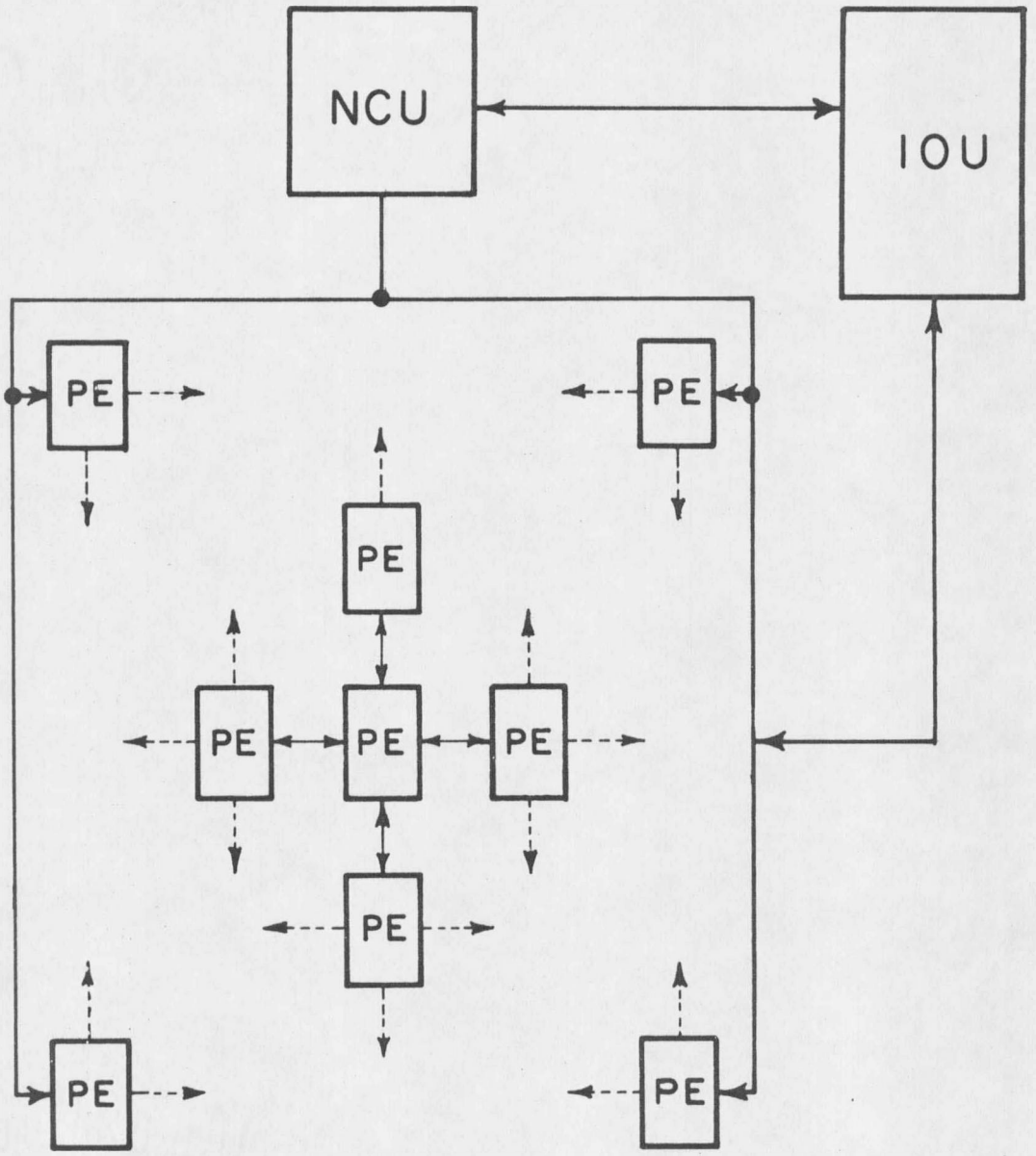


Figure 1.5 The Basic Organization of the SOLOMON Computer.

A PE possesses complete arithmetic capabilities; it also has two memory frames of storage with 4096 bits of storage per frame, which is expandable to 16,384 bits. Each PE is capable of performing serial-by-bit arithmetic and logic operations. Information is transferred to or from any of the four nearest neighbors of a PE serially-by-bit.

The SOLOMON computer was the first attempt (1962) to construct a machine specifically oriented towards the solution of the general class of matrix problems. It presented modular distributed computer design by introducing the concept of a module or a PE with its own storage and control capability. The PE operates under the overall control of the NCU and IOU over the whole array of modules or PE's with the possibility of communication between the four nearest neighbor modules only.

The SOLOMON computer, since it was constructed from discrete components available at the time, was organized as a serial-by-bit machine. The same machine could be constructed as a parallel-by word computer with only minor architecture changes necessary.

1.2.5 The Gonzalez Machine (A Multilayer Iterative Circuit Computer)

A machine was proposed by Rodolfo Gonzalez (17) in 1963 for dealing with problems involving spatial relationships between

variables. The Gonzalez machine has the following features:

1. Path building which is similar to that proposed by Unger (3) and Holland (24).
2. Three stacked layers of processing elements which are referred to as program layer, control layer, and computing layer. Every layer has a specialized function, separating the flow of control signals from the flow of information.
3. A three-phase sequence of operations in which initially the program layer stores the data instructions, subsequently the control layer decodes the instructions and the computing layer executes the instructions.

In steady state operation, these three layers activate simultaneously and operate on different instructions. This results in an effective instruction time equivalent to the time it takes one layer to perform its task.

According to Gonzalez, any individual module can function at various times as an accumulator, register, memory cell, or simply as a connecting link. It can be activated in any of these functions at any time during the execution of the program.

The throughput of the Gonzalez machine is restricted by the scheme used for selecting operands. This is due to the following two factors: 1) the path building procedure is essentially sequential, resulting in long effective execution time; and 2) for operands which

are not in two physically adjacent modules a link is built which uses the modules between the ones containing the operands as links. These linking modules are precluded from doing any other operation except linking.

1.2.6 The ILLIAC IV Computer

The ILLIAC IV computer which was introduced by Barnes et al. (4) of the University of Illinois at Urbana in 1968, is an array processor; it can be considered as a descendant of the SOLOMON computer, with some revision of the philosophy of the SOLOMON design. While the SOLOMON computer deviates significantly from the Von Neuman concept of digital computers, the ILLIAC IV designers tried to make a cost versus performance compromise by retaining some conventional Von Neuman concepts. By employing several processing elements (PE's), and several input/output channels (I/O) but with a single control unit (CU); the goal of building a superfast computer at a reasonable cost appeared to be realizable. A single CU stems from the fact that control units are much more expensive than PE's or I/O units.

It was originally proposed to use 256 PE's divided into four quadrants of 64 each. Each quadrant has a single CU which decodes instructions and generates the same control signals for all PE's in the quadrant. Each PE can execute or inhibit the instruction issued

by the CU. The four arrays may be connected together under program control to permit multiprocessing or single processing operation.

Each PE has a processing element memory (PEM), which consists of 2048 words of 64 bits each; a thin film random access memory of 240 nanosecond (ns) cycle time is used. Each PE is capable of a 64 bit floating point multiplication in 400 ns or an addition in 240 ns.

Figure 1.6 shows the general organization of the ILLIAC IV computer as it was proposed. As the figure indicates, the ILLIAC IV system is supervised by a general purpose computer; a B6500 computer is used in the machine as constructed. A parallel accessed high data rate (10^9 bits/sec.) disk is used as a backup memory.

The modular organization of the ILLIAC IV as an array of PE's simplifies maintenance problems. The design policy of plugable modules reduces the down time of the machine and simplifies diagnostic procedures.

The B6500 is capable of compiling programs to run on the ILLIAC IV, simulating the function of the ILLIAC IV when it is down, and locating a faulty subsystem or a faulty module of the subsystem. This last function of the B6500 assumes that the B6500 is fault free.

The ILLIAC IV proved the computational potential behind the SOLOMON computer concept of parallel processing when implemented as parallel-by-word. It also yielded information about the implementation

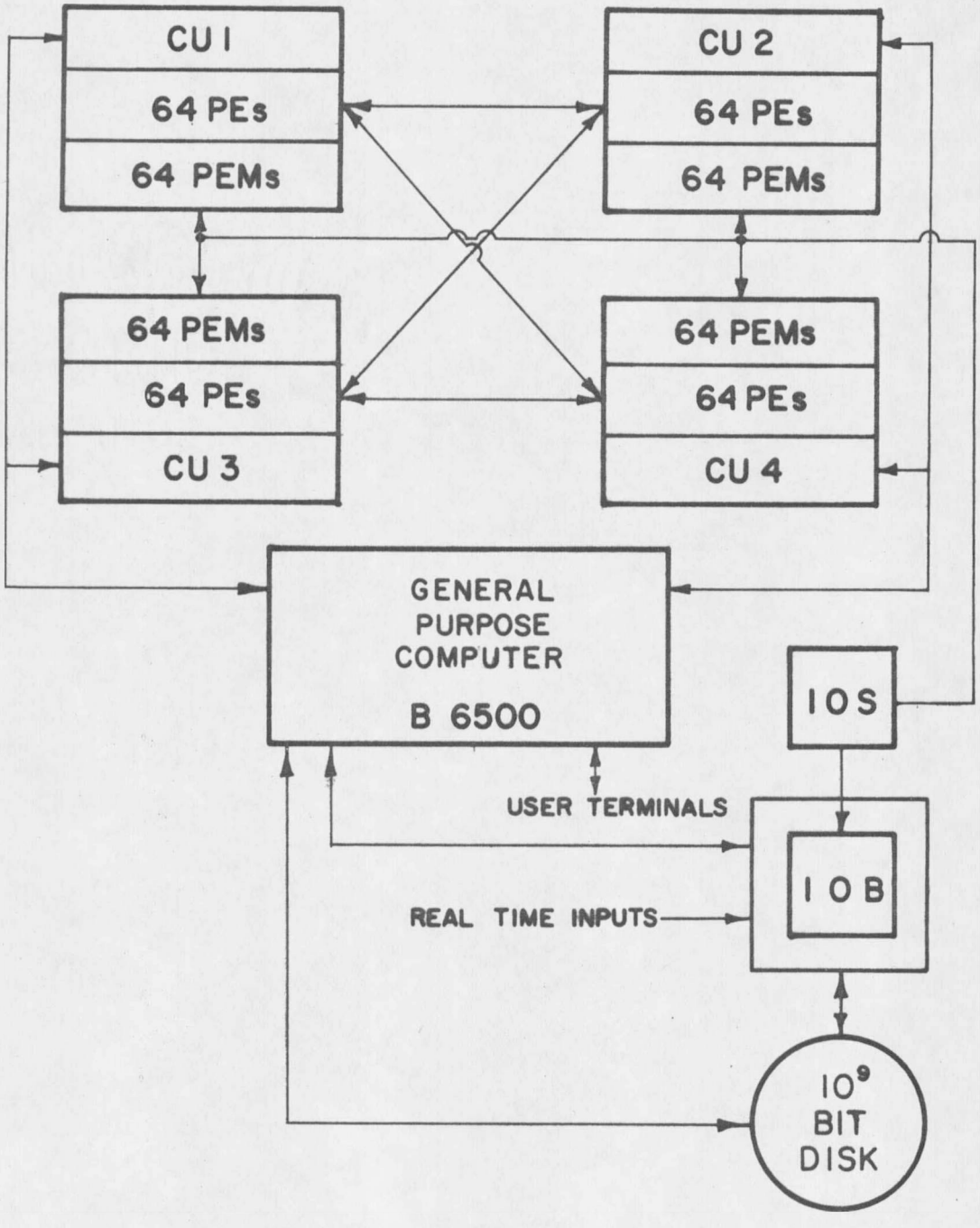


Figure 1.6 The General Organization of the 4 quadrant ILLIAC IV Computer.

of some modern techniques such as thin magnetic film memories and LSI logic. The goal of the ILLIAC IV project was to build a computer capable of executing on the order of 10^9 instructions/sec, but due to subcontractor problems (6), some technological changes were necessary. This led to cost escalation, reduction of execution rate to $2x10^8$ instructions/second, and limiting the system to one quadrant of 64 PE's only.

1.2.7 Cannon's Cellular Computer

Cannon's computer (9) is an array processor proposed in 1969, and was prompted by a search for machines capable of rapid solution of the discrete Kalman Filter Algorithm. Cannon called it the Kalman Filter (KF) machine. The KF machine is capable of performing matrix operations such as addition, subtraction, multiplication, transposition and inversion very efficiently.

The KF machine is composed of a square array of processing cells (PS's), a row control register, a column control register and a single global control unit (GCU).

The array of processing elements perform operations involving matrices and vectors. Data is stored in the array in a natural spatial distribution in the sense that the $i,j^{\underline{th}}$ element of the array is stored in the $i,j^{\underline{th}}$ cell of the processing array. See Figure 1.7.

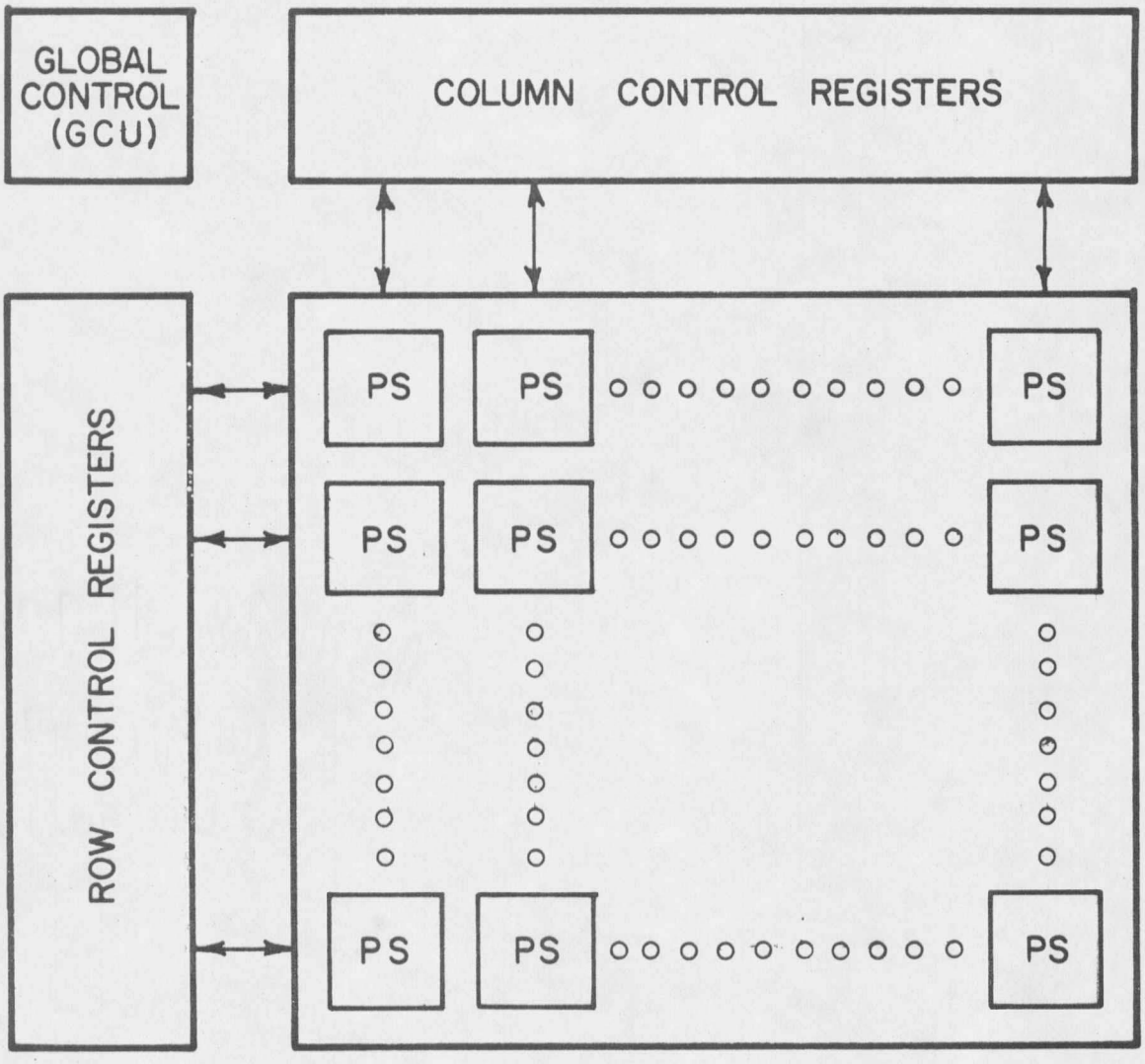


Figure 1.7 General Organization of the KF Machine.

The row and column control registers form an interface between the array of processing elements and the global control unit.

The global control unit (GCU) which resembles closely a supervisory machine, does the following:

1. Fetches instructions from its memory.
2. Decodes these instructions.
3. Sends control signals to the array processor for executing array type instructions.
4. Executes non-array type instructions.

Two types of instructions were suggested:

1. Array instructions which perform operations on vectors or matrices.
2. Non-array instructions which perform operations on scalars.

Cannon proposed specific logic units (9), as part of each processor cell, to do certain standard logic functions. He utilized AND, OR, and NOT as the basic set of elementary logic to build his special logic. Some of his special logic units are: add-one cell, One's to Two's complement cell, Comparator cell and Adder cell. It is to be noted that a PS of Cannon's machine is much simpler than a PE of the ILLIAC IV.

A reconfigureable array interconnection structure was designed to connect a cell to its neighbors vertically, horizontally, or minor-diagonally.

Cannon's machine proved to be significantly fast on matrix oriented problems. However, hardware requirements are high, since the computing array consists of N^2 computing units (N is the dimension of the array). An increase of execution rate proportional to N^2 is expected.

1.2.8 Mulder's Computer

A "Matrix-Oriented Cellular Computer Capable of Fault-Tolerant Operation" was proposed by Mulder (32) in 1970 as a continuation of the work done by Cannon. Mulder's machine reflects the ultimate Cannon style design, which has its origin in the SOLOMON computer and its descendant, the ILLIAC IV. The Mulder machine is a spatially oriented array of processing elements; the machine is two dimensional having control capabilities both in the cell (local), and over all the cells (global). See Figure 1.8. This organization yields more operational flexibility than that of Cannon's machine at the expense of extra hardware.

Mulder's design is different from Cannon's in the following areas: 1) processing cell organization and logical design; 2) speed of operation; 3) speed of data transfer within the processing array; 4) development of a computer assembly language instruction set; 5) complete array-structured computer/simulation (PAL-1);6) the introduction of hardwired microsequence control at the local

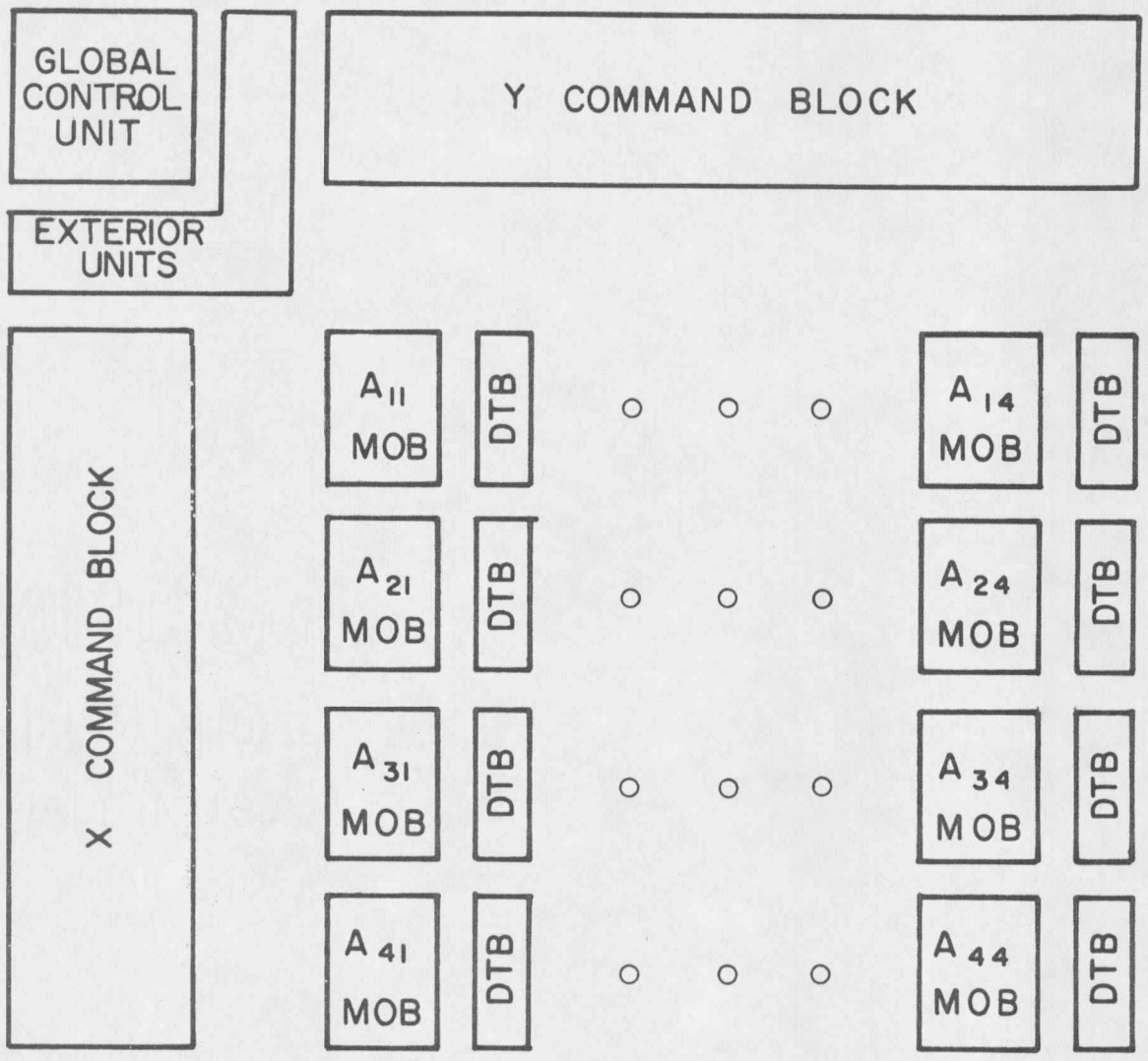


Figure 1.8 General Organization of Mulder's Computer

level; and 7) the introduction of additional parallel arithmetic operations.

A combination of hardware and software redundancies are used in the design to achieve a fault-tolerant operation of the computer elements or cells.

The GCU is a 10 Megahertz, 36-bit, general purpose computer, which corresponds to the B6500 in the ILLIAC IV system. This unit functions as a system executive, and initializes and executes assembly language, instruction sets. The GCU also contains a separate input/output processor of array data.

The exterior control units are the microprocessing unit used to control data transfer operation within the processing array and other units, the dynamic microprocessing unit used to facilitate execution of special instructions, and the input/output unit.

The X and Y command blocks route data and command signals from the GCU and exterior units to the processing array.

The processing array consists of NxN identical cells. Each cell consists of a Mathematical Operation Block (MOB), and a Data Transfer Block (DTB). A cell can be enabled or inhibited according to the X and Y commands. The MOB is capable of doing floating point addition, subtraction, and multiplication. The DTB is capable of routing information from one cell to another in a single operation regardless of the proximity of the two cells, which offers an advantage

over the Cannon machine (12) for which transfer is to the nearest neighbors only. The DTB is capable also of comparing two floating points words and has access to the memory module of the processing cell.

Mulder's machine represents a step forward in the array processors field. It solved some of the problems of Cannon's machine such as the faster routing of information and a more effective input/output processing unit. Mulder also explored the idea of macroinstructions and proposed a microprocessing unit as part of the exterior control unit to decode and execute the macros. Fault-Tolerant computing through hardware and software redundancy was also discussed in Mulder's thesis (32).

A drawback of the Mulder machine is the requirement of a complicated cell capable of arithmetic, logic, and data transfer operations. This complicated cell not only made the design and consequently the construction of the cell a hard task, but also degraded the potential performance of the machine, especially when the cell has to wait for the completion of an arithmetic instruction execution before initiating a data transfer instruction.

1.2.9 The PEPE Computer

PEPE (13) (Parallel Element Processing Ensemble), has evolved from work done in the early 1960's at Bell Laboratories

based on the premise that parallel data processors could gain increased performance through organization rather than circuit speed. Early work centered on a form of associative processing memory called Distributed Logic Memory (26) (DLM) by Lee and Paul, and later work by Crane and Githens (15) resulted in the eventual design of PEPE. Evidently more ongoing research is contemplated. According to Crane et al. (13) "PEPE should be considered more as a snapshot than as a final description".

PEPE is essentially a parallel processor peripheral of a host machine; it computes updated locations for multiple objects as required for real time radar processing. The highly parallel architecture of PEPE allows many object tracks to be processed simultaneously.

The PEPE-host combination can handle more objects simultaneously than is possible with the host computer alone. Although a cost versus performance study has indicated that PEPE can be cost effective only if it is built from MSI and LSI logic averaging about 256 bits per chip, the prototype which is already built is made of a low scale integration of logic flat-packs (averaging around three gates per package).

The prototype of PEPE already built has sixteen processing elements (PE's) arranged in two wings of eight PE's each. Each PE has three main parts: 1) an Arithmetic unit (AU), 2) a 512

word random access memory, of 32 bit words (M), and 3) a correlation unit (CU). A global control couples the ensemble of PE's to the host machine. The global control consists of an Arithmetic Control Unit (ACU) and a Correlation Control Unit (CCU). Figure 1.9 shows the general organization of the PEPE computer.

For array oriented problems each PE maintains information on a different element of the array, data is stored in M while the AU and CU provide the input, processing, and output functions for data maintenance.

The ACU and CCU together with the host computer store and control the programs specifying the operation of the AU's and CU's. The ACU broadcasts one program instruction at a time to all AU's simultaneously. The CCU executes a second program independently and concurrently with the ACU, broadcasting one instruction at a time to all CU's.

Each AU has an 8-bit content addressed tag register, that together with the AU accumulator register determines the status of the AU activity. PEPE executes the same instruction in all cells which limits the speed of the machine when it is applied to problems with different instructions at different locations in the array, e.g. the FFT algorithm (13). Also a host machine is essential for such tasks as compiling and scheduling of programs.

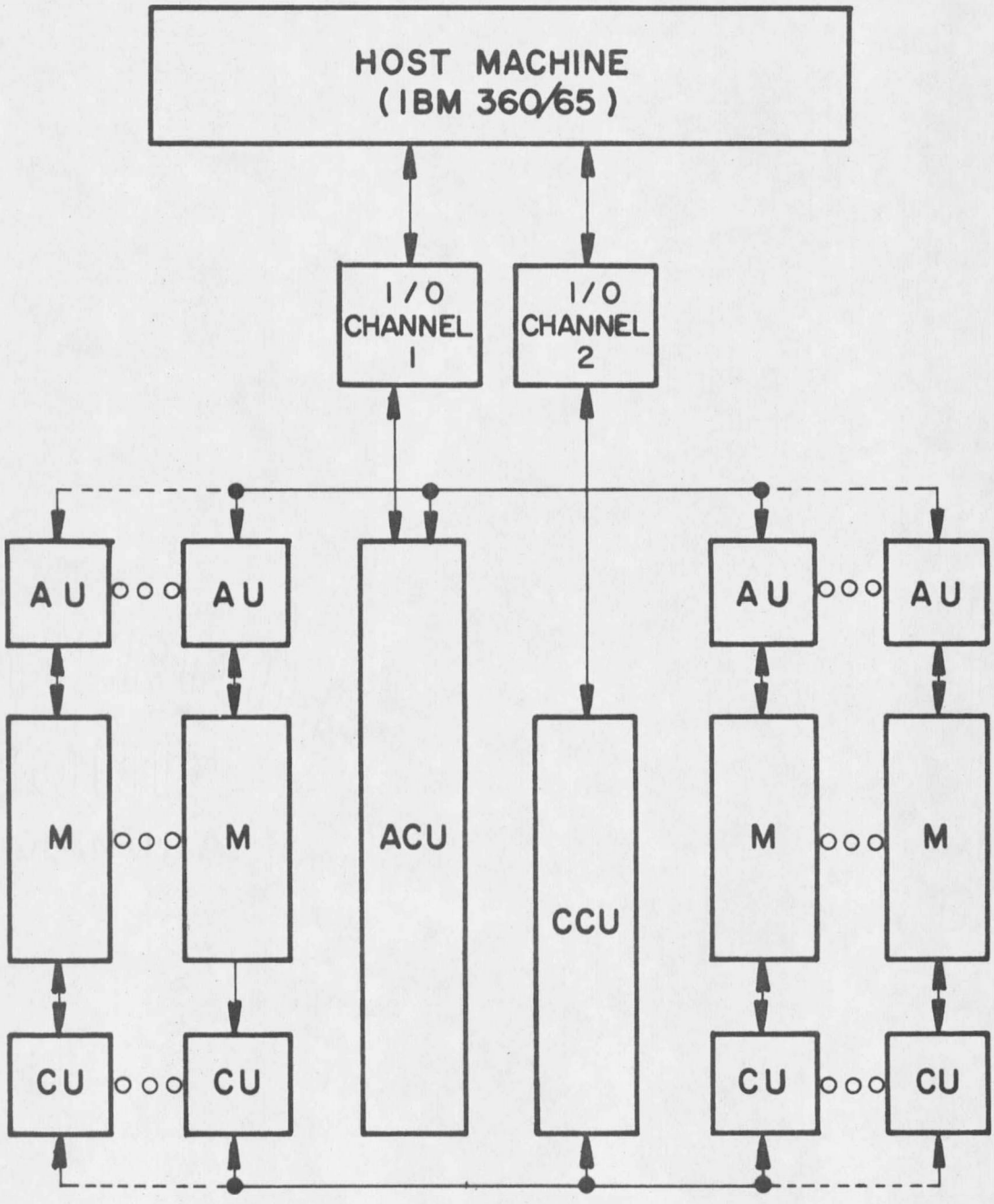


Figure 1.9 General Organization of the PEPE Computer.

According to Crane et al. (13): "PEPE can perhaps be best characterized as an associatively-organized, highly parallel computer capable of executing two instruction streams simultaneously. As such, PEPE bears similarities both to array processors such as ILLIAC IV and to associative processors.

PEPE differs from ILLIAC IV, however, in its use of a second instruction stream for correlation, and also in the way that PE's interact. ILLIAC IV is an array processor wherein each PE represents a different point in a two dimensional grid and communicates directly with its four nearest neighbors. This arrangement is powerful in applications such as heat flow analysis, weather forecasting, etc., which require much exchange of data among neighboring PE's. PEPE, on the other hand, provides a real-time data base representing objects having little interaction with one another and there is no need for the PE's to have direct intercommunication."

PEPE is therefore more of an ensemble computer than an array one. This lack of communication in PEPE provides for more reliability of the PE since less hardware is employed to build the PE.

1.3 Why a 3-D Computer?

All of the aforementioned machines are array processors; that is, the arrangement of the processing elements is in a

rectangular interconnection pattern. All of these machines suffered from some shortcoming or another.

Unger's machine (45) presented the first attempt known to the author to establish a computing machine which consisted of an array of identical modules or processing elements. Unger's machine suffered from a rigid design which required a module either to execute the same instruction executed in the rest of modules or to inhibit the execution and stay idle. The modules of Unger's machine implemented elementary operations only so that any non-elementary operation had to be macro-programmed out of the elementary operations available. These shortcomings made Unger's machine difficult to program efficiently so as to make the best use of the available hardware capabilities.

Holland's machine (24) as a follow up of Unger's computer attempted to spread the control through the array structure, thereby causing the modules to acquire more computational and communicational power. This complicated the hardware design of the individual modules and their interface hardware, thus making it still more difficult to program the machine effectively.

Comfort's computer (12) (called a Modified Holland Computer) came into existence to simplify the hardware design and programming effort relative to that of Holland's computer without sacrificing much of the computing power of a Holland-like machine. This was

achieved by divorcing the arithmetic function of the machine from the data transfer and movement functions by introducing the A-boxes. Cost reduction was achieved by sharing the A-boxes between the different modules of the machine through a switching network.

The SOLOMON computer (18) was proposed to alleviate the problems of its ancestors, in particular in regard to programming. The SOLOMON computer was much larger in scale than the previously mentioned computers; it has more computing power and higher speed. At the time it was proposed (1962) it was expensive to afford a word size bus, thus it was evident that achieving powerful control and computing capabilities were hindered by the serial-by-bit data transfer.

ILLIAC IV (4) came along to alleviate the SOLOMON computer's shortcomings. It did increase the throughput tremendously (10^9 instructions/sec. was proposed, but a 10^8 instructions/sec. rate was actually achieived) by operating on words instead of bits both in arithmetic and transfer operations. Although ILLIAC IV inherently has a powerful processing element capable of both arithmetic and transfer operations it still had a processing element communication capability with the four nearest neighbors only.

The Cannon (9) and Mulder (32) machines are descendants of the SOLOMON computer and the ILLIAC IV computer, (Cannon and

Mulder computers and the 3-D machine were never intended to have the same size of computing power of the ILLIAC IV computer). The Cannon and Mulder machines are rectangular arrays of identical cells which are interconnected in vertical, horizontal, and minor diagonal directions. This provides the capability of interchange of information between calls in a manner that is not comparable to the serial interchange of information to be found in the Von Neuman concept. It is also more powerful than the four nearest neighbors communication to be found in their ancestors. A much richer flow of information is possible in these machines. However, certain deficiencies were uncovered in the evaluation of these machines. The principal defects were that each cell (designed to handle arithmetic operations, data manipulation and transfer, and input/output operations) was firstly very complex, and secondly capable of performing only one of the three operations mentioned at a time. That is, data transfer could occur, arithmetic operations could occur, or input/output could occur, but only one of these at a time. It became apparent that there was an inefficient use of hardware in the machine and that ideally the structure of the cell should be simplified to provide a mechanism whereby overlapping of operations in the three areas could be affected. This gave rise to the notion of the Three-Dimensional Computer (3-D Computer) in which the array structure was expanded in space

from a two-dimensional arrangement. Memory is brought out to form a single section which is devoted solely to the storage and the transfer of data. The computing function is performed in another section of the machine, which is devoted solely to that operation. The input/output operations are performed in another section of the machine which is again solely dedicated to that function. By spreading the function over the structure of the machine, the memory cells are simplified because they no longer contain the arithmetic units, the arithmetic cells are simplified because they no longer contain memory or I/O, and the I/O cells are simplified because they contain neither of the other two functions. Thus, a larger number of cells appear, but each one is of a simpler structure with the additional advantage that each section of the computer can now operate in an overlapped fashion with the other sections, thereby increasing the throughput of the machine. A measure of the effectiveness of this spreading out of the function over the structure of the machine will very likely be in the sense of the cost per operation of the machine. As the price of hardware for these particular cells decreases, (and historically the price of hardware has come down), the effectiveness of the 3-D organization will become greater and greater.

1.4 An Outline of the Subsequent Chapters

The material covered by this thesis is:

1. A general description at the register-transfer-level of the 3-D machine. This description is very brief at some points, especially when it deals with subsystems which have been dealt with exhaustively in the literature. This general description is the theme of Chapter 2.
2. Chapter 3 is devoted to the presentation of an assembly language which has been devised and used to simulate the 3-D machine operation at the register-transfer-level.
3. Chapter 4 is devoted to some parallel algorithms for matrix operations. Some of these algorithms have been written especially for the 3-D machine; the algorithms for matrix transpose, matrix multiplication, and the search for maximum or minimum element of an array. The matrix inversion algorithm is an adaption of the Gauss-Jordan elimination algorithm (16) written for conventional serial machines.
4. Chapter 5 is mostly a survey of the literature on fault-tolerant computing and design considerations as applicable to the 3-D machine. Some fault-tolerant computing and design considerations which are peculiar to the 3-D machine architecture have been proposed.
5. The appendices are a collection of results of research work which has been done in the following areas:

Appendix A summarizes the work done to find the valid connection codes for the memory plane intraplane transfers and their corresponding gate settings.

Appendix B is a summary of the results of applying the Quine-McClusky algorithm (29) to the gate settings.

Appendix C is a listing of the simulator.

Appendix D is a listing of a parallel matrix transpose algorithm.

Appendix E is a listing of a parallel matrix multiplication algorithm.

Appendix F is a listing of a parallel matrix inversion algorithm.

Appendix G is a cost estimation procedure to find cost estimates of 3-D machines.

Appendix H is a survey of possible non-conductive data transfer technologies.

CHAPTER 2

HARDWARE ORGANIZATION OF THE 3-D COMPUTER

2.1 General Organization of the 3-D Computer

The 3-D machine organization follows the basic philosophy of its predecessors, the Cannon and Mulder machines, which in turn have their origin in the SOLOMON computer and its descendant, ILLIAC IV. As a result of applying the simulator of the Mulder machine to some matrix-oriented problems it was found that throughput of the machine could be improved by a factor of 2 or 3 if the processing element is partitioned into three general sections which are physically separate and are capable of operating simultaneously; the I/O processor, the data manipulation and movement processor (with its associated memory), and the computation (arithmetic and logic) processor (32, 37).

This led to an overall machine basically composed of five subsystems as shown in Figure 2.1. Much of the detail of the constituents of each subsystem is reserved for later sections so as not to obscure the general presentation of the architecture.

The five subsystems are:

1. The Memory Cube (MC), which is a parallelepiped of Memory Planes (MP), each arranged as a two-dimensional rectangular grid, with one memory cell at each crosspoint of the grid. The memory cells have a particular interconnection structure within the plane, plus the capability of communicating with any cell in adjacent planes at corresponding points

above or below. Although it is apparent that the memory is not necessarily cubical, the name is used because of its suggestiveness.

2. A Computation Place (CP) which is composed of a rectangular array of computing cells, each cell being located at the crosspoints of a grid having the same dimensions as the MP grid. The computation cells do not have any interconnections between them, (they do have connections to the supervisory machine and some special processors which facilitate the interrogation of the CP condition). They receive inputs from, and pass their outputs to, the cell at corresponding grid points in the MC or the I/O plane. A primitive set of arithmetic and logic capabilities for each cell will be defined subsequently.

3. An Input/Output Plane (IOP) which interfaces the MC and the CP to transducers and mass data storage. It is also arranged as a grid of cells such that when transfer to or from MC or CP is to be made, data in corresponding grid point cells can be transferred. Several IOP designs will be considered which will require ascending orders of hardware complexity commensurate with the speed and versatility of I/O operation desired.

4. The Connection Register Read-Mostly-Store (CR), which

consists of one or more planes of special storage registers in which are stored control words that configure register interconnections in the MC, and set operation codes (OP codes) and cancellation numbers (CN) (to be described later). The dimensionality of cells in the CR is identical to that of the MP, the CP, and the IOP. The registers of the CR plane are identical to the MRC registers of the MC to be described in Section 2.3.1.

5. A control and supervisory section composed of (1) control logic associated with the preceding four system elements and (2) a supervisory machine which compiles and stores the program, executes non-array instructions, causes appropriate signals to be applied to the control logic when array instructions are called for, and handles priority interrupts. The supervisory machine stores and executes several diagnostic programs with different degrees of complexity to check for faults of different complexity. These diagnostic routines perform tasks ranging from diagnosing a faulty subsystem, a faulty plane of the subsystem, a faulty cell, to the task of finding a faulty register or even a faulty bit. A program to simulate 3-D machine operation can be stored in secondary storage and executed at a very low performance rate on the supervisory machine when the 3-D machine is down.

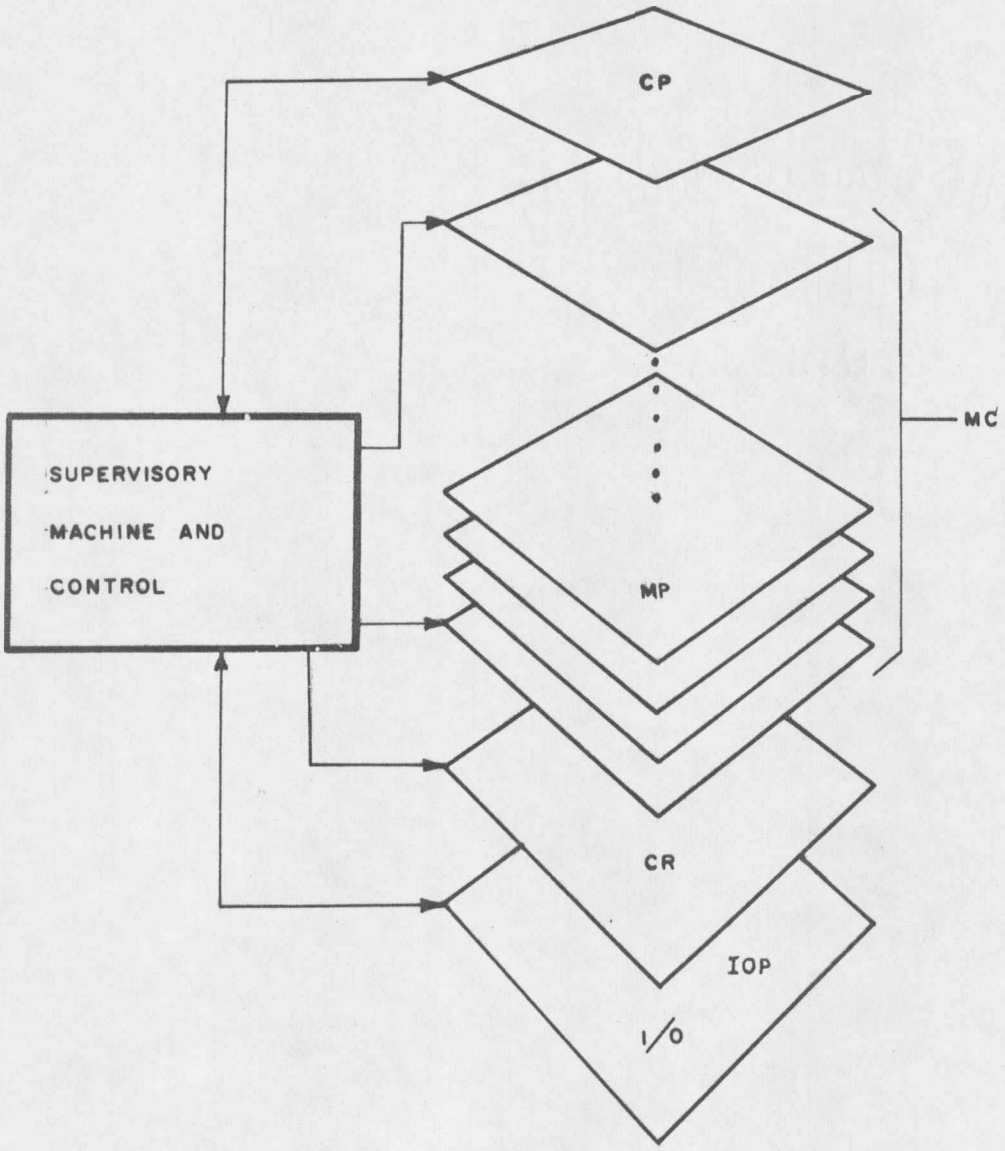


Figure 2.1 Physical Relation of Major 3-D Machine Subsystems.

2.2 Technological Assumptions

2.2.1 Integrated Circuits Assumptions

A primary problem in research dealing with cellular logic and arrays is that of properly understanding the relevant integrated circuit (IC) developments, and even more important, of correctly anticipating future developments in IC technology. It is a fact in the computer design field as well as in many others, that specialization often keeps the computer architect from full knowledge of the current and anticipated advances in IC technology; conversely, IC designers frequently do not have a complete understanding of the current developments and needs in computer architecture. With this disclaimer in mind, some assumptions will be made concerning the physical embodiment of the 3-D machine.

It seems reasonable to assume that 1000 to 1500 gates per chip of large scale integration (LSI) logic will be available in the near future (55). This assumption will be necessary for all cells whether they be in the MC, the CP, the IOP, or the CR. Therefore, although not essential to the development of 3-D machine hardware, we assume that a one IC chip per memory cell correspondence holds.

2.2.2 Non-Conductive Interplane Transfer

The 3-D machine can benefit significantly from a non-conductive data transfer between planes. Cost considerations require the assumption that interplane transfers will be accomplished serial-by-bit whereas intraplane transfers will be parallel by word in the case of non-conductive transfer. The problem of undesired interaction between separate memory planes can be completely eliminated by proper non-conductive transfer schemes. Also, the interplane connection pins required by conductive transfer can be eliminated, a step in the right direction when the total pin-limitation problem is considered. A short survey of non-conductive technology appears as Appendix H. Several technological problems of non-conductive transfer which might preclude its use in the 3-D machine are treated there. Henceforth, interplane transfer will be spoken of as though it were non-conductive, fully realizing that appropriate technology may not be forthcoming and a conductive method may be necessary. This does not detract from functional aspects of the machine; it only changes structural aspects.

2.3.1 The Memory Cube

Memory cube structure allows functional capabilities which are more extensive than those normally associated with a conventional memory store. Traditionally, memory has served a single function

in serial machines, that of storage and retrieval of data. A new role of memory began with the Holland (24) and SOLOMON (18) machines and extends through the Cannon (9) and Mulder (32) machines and the ILLIAC IV (4). All of these machines have computing elements, each with arithmetic capability and some memory, which are interconnected via a programmable, regular interconnection pattern, usually in some chain form. They are not strictly memory-to-processor interconnections but rather are associated with the manner in which data can be manipulated within the array prior to execution of arithmetic or logic operations. It is possible in any of these machines to perform memory-to-memory transfers through the processing elements and thereby obtain desired data patterns.

Except for ILLIAC IV, which is of a size and complexity greatly exceeding the proposed 3-D machine, the disadvantage here is that when such machines are used to perform data manipulations, this manipulation precludes the use of the arithmetic unit to perform simultaneous arithmetic operations. Consequently, much of the power of such machines is lost to the conflicting demands on resources. In the 3-D machine the function of more explicit data manipulation has been separated from the function of arithmetic or logic operation or input-output, and has been concentrated in the memory cube. Thus, the memory cube is a repository of information and additionally, (and in fact at least as importantly) is the data manipulation

structure for the machine, providing geometrically useful data transfer paths that can be highly effective in prearranging data for certain types of arithmetic and logical operations. Data transfer in the 3-D machine is innovative, not only in the fact that the data transfer mechanism is a structure independent of the computing mechanism but also in the fact that in the 3-D machine transfers may be made in three dimensions. As an additional benefit, the 3-D machine has no edge connections for data transfer, all transfers being made either internally via conductive hardwired paths or from plane to plane via non-conductive transfer. In the sense of being a multiplane structure with interplane transfer between corresponding cells, it resembles the Gonzales (17) machine, but there the similarity ends. The 3-D machine does not have its function spread over structure to the extent exhibited by the Gonzales machines.

Structurally, the MC is composed of a set of memory planes, each plane having within it a set of memory cells whose dimensionality matches that of the CP, i.e. for a CP composed of i rows and j columns of cells, the MP's are also composed of i rows and j columns of cells. Two data registers are available in each cell and are referred to as MR1 and MR2. Each data register is a single length register of assumed 32 bit word length. Each register has seven lines which can be used as input or output (exclusive 'OR' sense, i.e. a line will carry either an input or an output but never both simultaneously),

providing transfer capabilities within the plane in the vertical, horizontal and minor diagonal directions, plus provision for transfer between MR1 and MR2 of the same cell. Figure 2.2 shows all the possible directions of intraplane transfer for MR1 and MR2. A gate-level design of the gates controlling the conductive transfer is presented in Section 2.3.4.1.

Figure 2.3 shows a 2x2 array of memory cells, excluding storage registers. Two additional registers are shown and are labeled MRC and MRT. MRC is a control register and stores interconnection patterns, op codes and cancellation codes (CC) described briefly herein and to be described in more detail in Sections 2.3-4.1. MRT is a word-wide register which is directly connected to all registers in the cell and serves as a temporary storage for interplane transfers. MRC is a double length register (64 bits), with 42 bits reserved for cell control gate settings which are a decoded form of the interconnection pattern or code. The interconnection code (ICC) is a 6 decimal digit code, 3 to describe the connections for MR1 and 3 to describe the same for MR2.

Similarly the 42 bits in MRC which represents a decoded ICC are divided into 21 bits for MR1's connections and 21 bits for MR2's connections. MRC contains 8 bits of ALOC, 4 bits for the CN of the cell, 4 bits for interplane connections, and the remaining 6 bits are used for code parity checking as explained in Chapter 5.

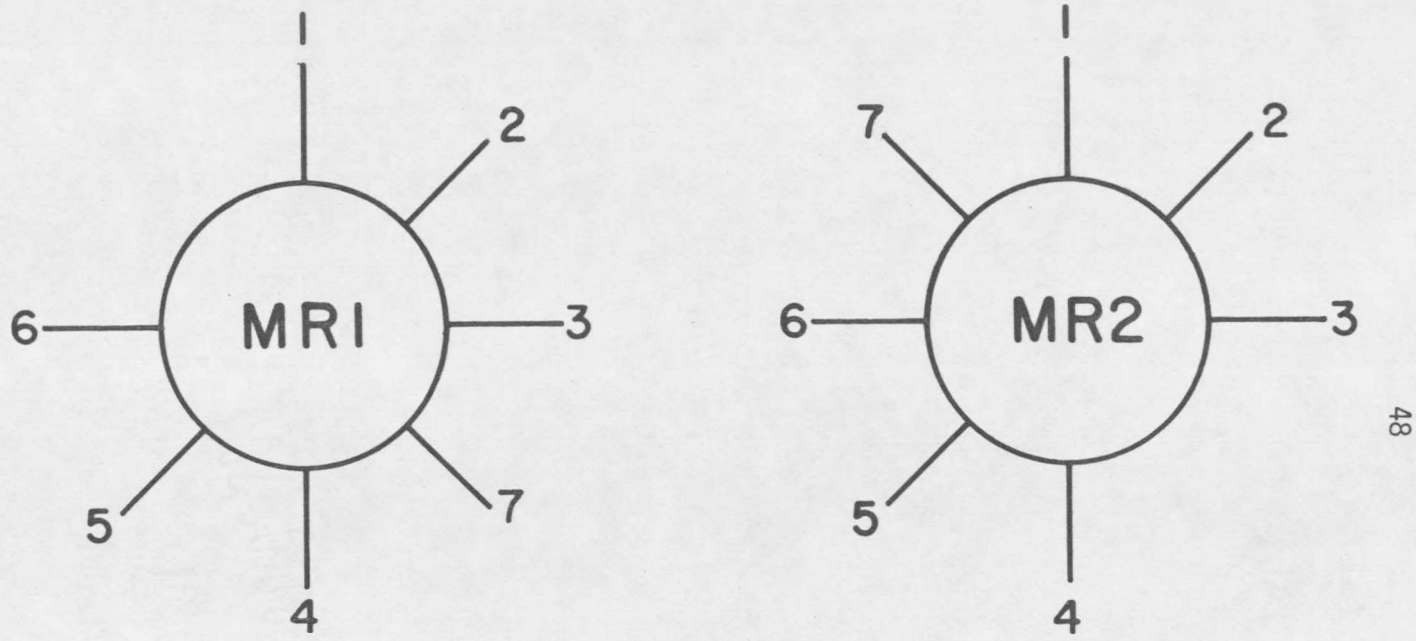


Figure 2.2 Interaplane transfer possibilities for MR1 and MR2.

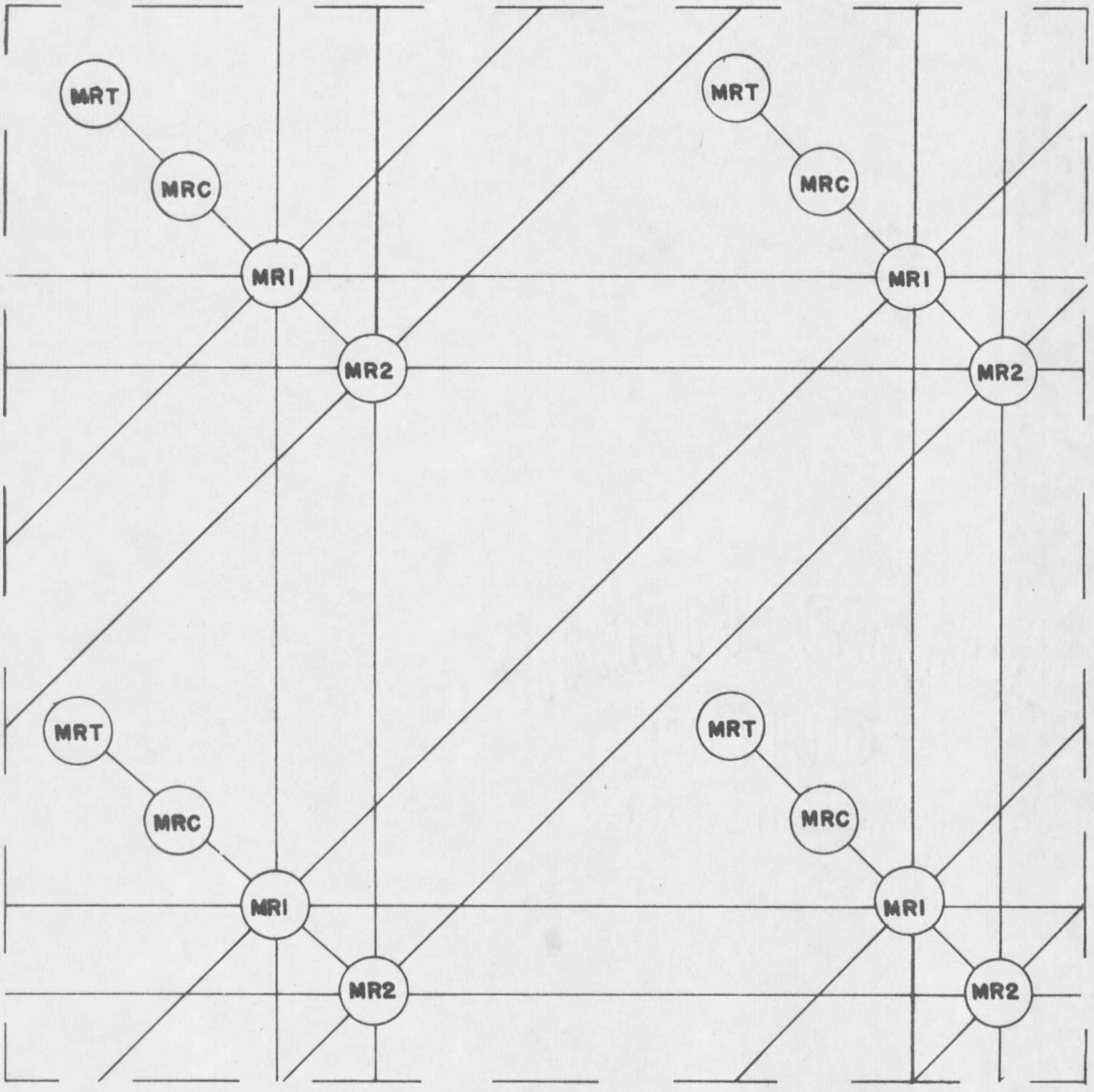


Figure 2.3 A 2x2 Memory Plane.

2.3.2 The Computation Plane (CP)

The computation plane is a rectangular array of individual computing cells, each cell of which has identical instruction execution capabilities. A preliminary introduction set including floating point and integer arithmetic and various logical operations has been defined and presented in table 2.1. The operation code field allows for future expansion to 256 operation codes. Data can be transferred to and from the CP by the non-conductive mechanism to or from the MC or the IOP only. There is no provision for intercell data transfer in the CP nor is there any provision for data connection of the CP to the outside world. This does not exclude the connection of data in the CP to some special processors such as the SORTING processor to be described later. In addition program status data relating to the condition of the data is connected directly (hardwired) to the supervisory machine through a 32-word block which is a part of a high-speed scratch pad memory in the supervisory control system. These 32 words are organized as 256 condition code status words. Each status word corresponds to the status of one of the 256 CP cells. The status word for one cell is a 4-bit word which allows any one of 16 states of data in the CP to be recognized and worked on. These states (as in most CPU's of conventional machines) indicate overflow, underflow, positive, negative, zero, etc. contents of the CPA register of the cell.

Table 2.1 Arithmetic Logic Operation Codes.

OP Code	Meaning	Hex Equivalent
NOP	No operation	00
ADD	Integer addition	01
SUB	Integer subtraction	02
MPY	Integer multiplication	03
DIV	Integer division	04
FAD	Floating point addition	05
FSB	Floating point subtraction	06
FMY	Floating point multiplication	07
FDV	Floating point division	08
SLI	Shift left one bit	09
SRI	Shift right one bit	0A
AND	Logical and	0B
IOR	Inclusive or	0C
XOR	Exclusive or	0D
NOT	Complement (1's complement)	0E
CPM	2's complement	0F

The CP cells are not restricted to execution of the same operation simultaneously, but are allowed partially autonomous operation. Exactly how much autonomy should be given to the CP is a matter for future decision, to be determined after the 3-D machine simulator is used sufficiently to indicate the cost-performance tradeoffs. Apparently the most costly situation is the one in which each cell is capable of performing a completely autonomous operation relative to all the other cells. Zoned operations are recommended as an interim solution in which case the CP plane is subdivided into sub-planes with each sub-plane doing a different operation. A Completely autonomous CP is a special case of a zoned CP in which each zone consists of a single cell. This autonomy is generated by sending the OP code to each CP cell (or zone of cells) decoder from the corresponding MRC.

Generally a complete transfer to the CP implies a transfer of two operands and an OP code for each cell. Thus some cells can perform addition, some can perform multiplication and others can perform division, simultaneously. Although operands can be transferred from either the MC or the IOP to the CP, no operation can be initiated until an OP code is received from one of the MRC registers. This requires that a transfer from the IOP be followed by a transfer from the MC in which operand transfer in inhibited (by appropriate cancellation codes to be discussed in Chapter 3)

and only the operation code is transferred.

A programming option allows ganged operation of the CP so that all computation cells will execute a single operation simultaneously, with the possibility that specified cells may be inhibited. When such an option is initiated the single operation code does not come from MRC but instead comes from the supervisory machine, since the single operation code is all that is necessary.

In addition to the richness allowed by the autonomous operation, the computation plane has all the capabilities of ganged operations to be found in such organizations as the ILLIAC IV or the PEPE machine.

Structurally, arithmetic organization of each cell calls for a 32-bit parallel arithmetic unit controlled by a microprogrammed arithmetic/logic processor. The arithmetic/logic processor consists mainly of an operation decoder, a processor controller, and a microprogram storage read-mostly memory. Figure 2.4 shows the general organization of the arithmetic unit of one zoned cells of the CP, one notes that it is not necessary to have a microprogrammed arithmetic/logic processor for each cell of the CP. The arithmetic unit of one cell of the CP requires three word-length registers (a word is of 32 bit length), CPA, CPQ, and CPB. An eight word temporary storage structure is also associated with the CPA register. This is for intermediate results that are most efficiently kept immediately

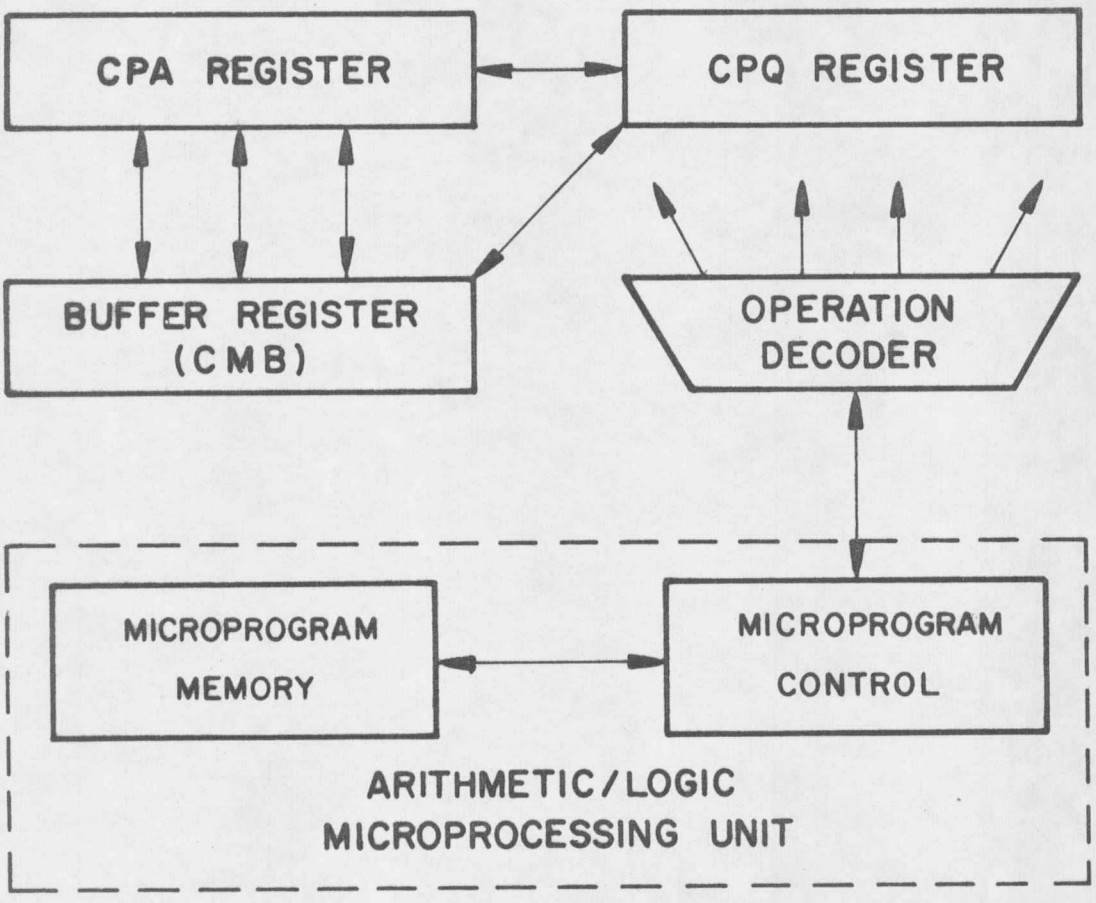


Figure 2.4 Arithmetic Unit of One Zone/Cell of CP.

accessible to the arithmetic registers rather than requiring transfer back to the MC. It has 3 identifier bits appended to the data bits which can be used in the multiprogramming mode to identify the job with which the word is associated; this actually means that the CPA register is a 35 bit register instead of being 32 bit like the other registers of the CP. Thus the CPA memory store is actually a list, but appears as a push-up pull-down stack insofar as a single job is concerned.

CPB is equivalent to MRT (memory interplane transfer register) and is equipped to transfer data non-conductively. By being directly connected to CPA and CPQ, CPB can transfer data to CPA and CPQ or receive data from CPA and CPQ. CPQ is connected to CPA to form a double length register.

To add two numbers, the augend is loaded through CPB into CPA, then the addend is loaded into CPB. Parallel add circuits (not shown) replace the augend with the sum, which can be stored in the list or transferred through CPB into MC. Subtraction is similar. For multiplication and division repeated addition and subtraction directed by the microprogrammed arithmetic/logic processor is used together with a two-word-wide shift in the combined CPA and CPQ register. The details of the CP microprocessor is beyond the scopes of this thesis. A good example of a microprogrammed processor which could serve the purpose for the 3-D machine is the Hewlett Packard

2100 microprocessor (57).

2.3.3 Input-Output-Plane (IOP)

In general, I/O operations are relatively time consuming in any machine. In a parallel machine they constitute a special bottle neck for the designer who must interface such a machine to serially oriented peripherals.

In the 3-D machine, the CP and MC are divorced from the IOP, which improves the throughput inherently in a manner analogous to the inclusion of direct memory access in a general purpose machine. However, greatly increased throughput will be possible only if the IOP is extremely fast. An I/O operation for a 16 x 16 array of words requires 2.56 μs, based on 100 MHz clock rate. This means a throughput gain of 2:1 is achieved over an equivalent data transfer in the Mulder machine, by divorcing the IOP from the CP as in the 3-D machine. The comparison does not stop here since we can speed the I/O operation by using other elaborate designs to be discussed in the next section. For example, one design having an IOP 16 times faster than the one to be discussed below achieves a throughput improvement of 17:1 over the Mulder machine.

The IOP structure to be discussed is a pipelined structure, which is very desirable. Once started, the I/O operation will continue until all cells specified have received their input.

The particular pipelined action of the IOP used is achieved through two 100 MHZ, 16-bit shift registers: the vertical-control-shift-register (VCSR) and the horizontal-control-shift-register (HCSR), as shown in Figure 2.5. The rate of the VCSR is made equal to the rate of the HCSR divided by the number of columns to receive input. This is achieved by passing the clock signal for this shift register through a divide-by-the-number-of-columns circuit. VCSR and HCSR contain control bits, never data. Figure 2.6 shows the clock signals of VCSR. In Figure 2.6 Ki represents the clock signal for processing i columns, and i = 1, 2, 4, 8, 16.

The IOP is an array of cells of the same dimension as the MP in which each cell has one storage register and a transfer register. Each row is connected to a single bidirectional bus through gates set by the HCSR. A single vertical bidirectional bus is connected to the horizontal bus through gates set by the VCSR. The vertical bus is connected to two hardware stacks, the input-buffer-stack (IBS), and the output-buffer-stack (OBS), through an input-output-buffer-register (IOBR).

An input operation is as follows:

1. The VCSR receives two binary numbers; one is the index of the first row of the array to receive input, and the other is the index of the last row to receive input.
2. The HCSR receives another two binary coded number; one is

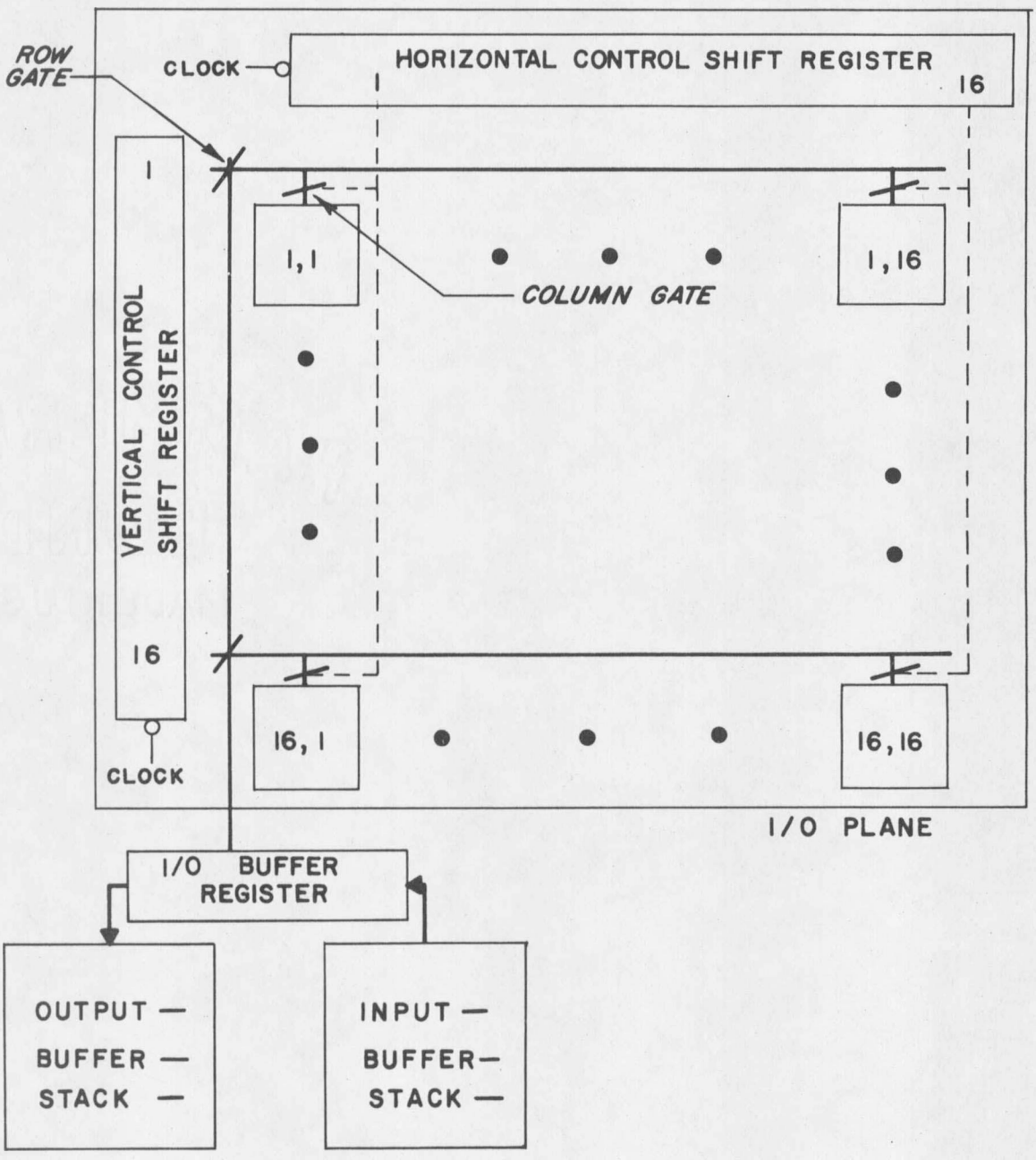


Figure 2.5 Input/Output Processor

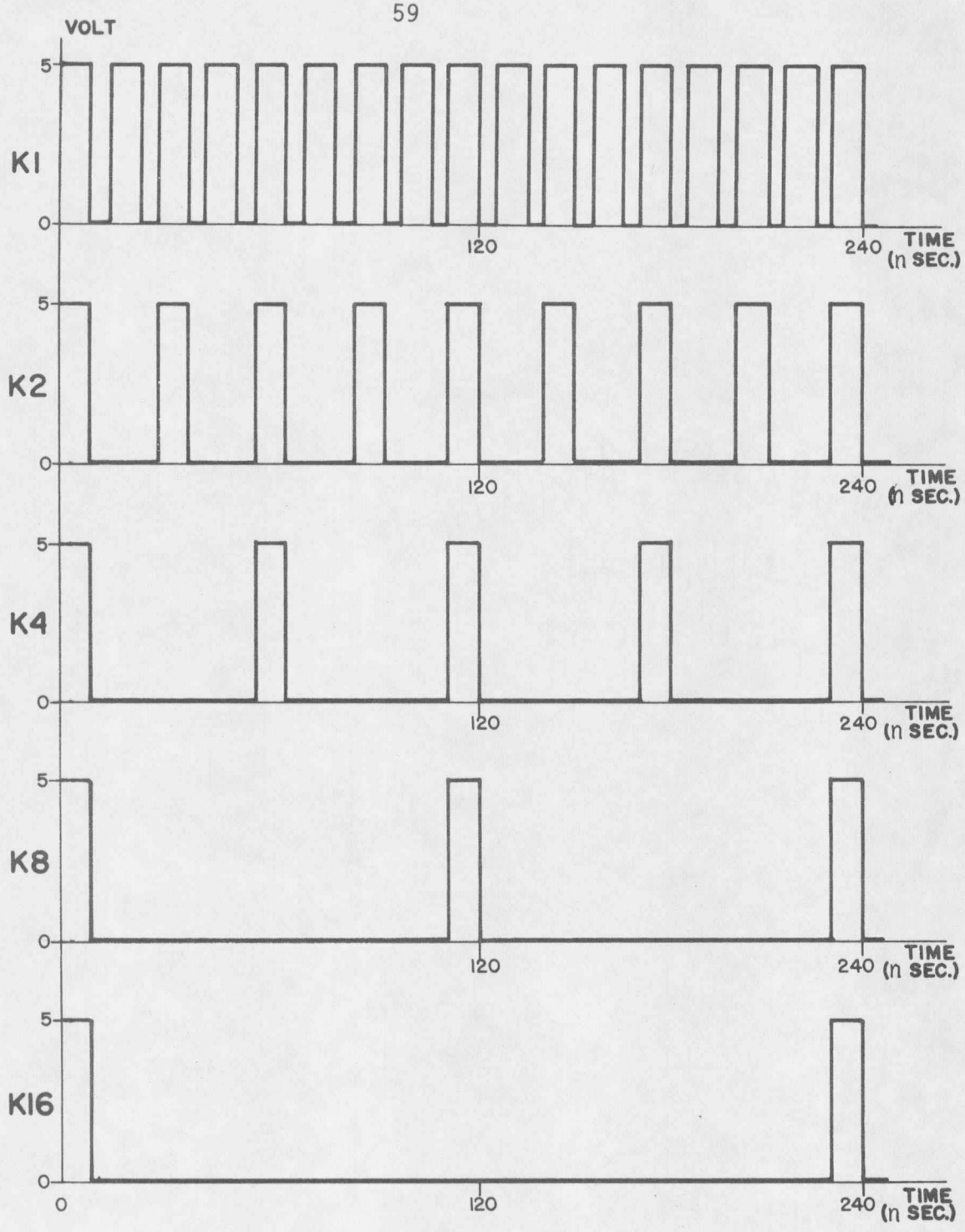


Figure 2.6 VCSR Clock Sequence Signals for 1, 2, 4, 8 and 16 column I/O operation.

the index of the first column of the array to receive input, and the other is the index of the last column to receive input. All intermediate columns between the first and last one are loaded.

3. The VCSR and HCSR set gates such that the IOBR is sequentially connected to the proper registers and words are loaded in bit parallel form into the IOP registers. It is to be noticed that VCSR closes the gate for the first row to receive input while the HCSR closes the gates for all columns specified, starting from the first column to receive input. A signal to close the gate for the next row is issued as soon as the last column of the previous row has received input. The timing of the clock on VCSR is variable according to the number of columns to receive input as indicated in Figure 2.6. Some rows/columns may be inhibited from receiving input due to faulty cells in these rows/columns. This function is transparent to the user as described in Chapter 5 on Fault-Tolerant Computing.
4. When filled, the IOP does a non-conductive transfer to the MC, the CR, or the CP as directed.

An output operation is the inverse of an input operation and is summarized as follows:

1. A data transfer takes place from the desired plane to be

outputted to the IOP through execution of a non-conductive transfer instruction.

2. The VCSR receives two binary numbers; the index of the first row of the array to be outputted, and the index of last row to be outputted.
3. The HCSR receives another two binary numbers; the index of the first column to be outputted, and the index of the last column to be outputted.
4. The VCSR and HCSR are sequenced as explained for an input operation such that data is pumped out of the IOP registers to the OBS by row through the IOBR. As in the input operations, the output from some rows/columns may be inhibited due to faulty cells.
5. When done the IOP signals back to the supervisory machine to start executing a new instruction.
6. External device initiated data transfer interrupts are assumed to be sent directly to the supervisory machine for priority execution queing.

2.3.4 Control Unit and Supervisory Machine

Control in the 3-D machine falls into two general categories:

1. Control within cells (local control).
2. Control by supervisory machine and attached external hard-

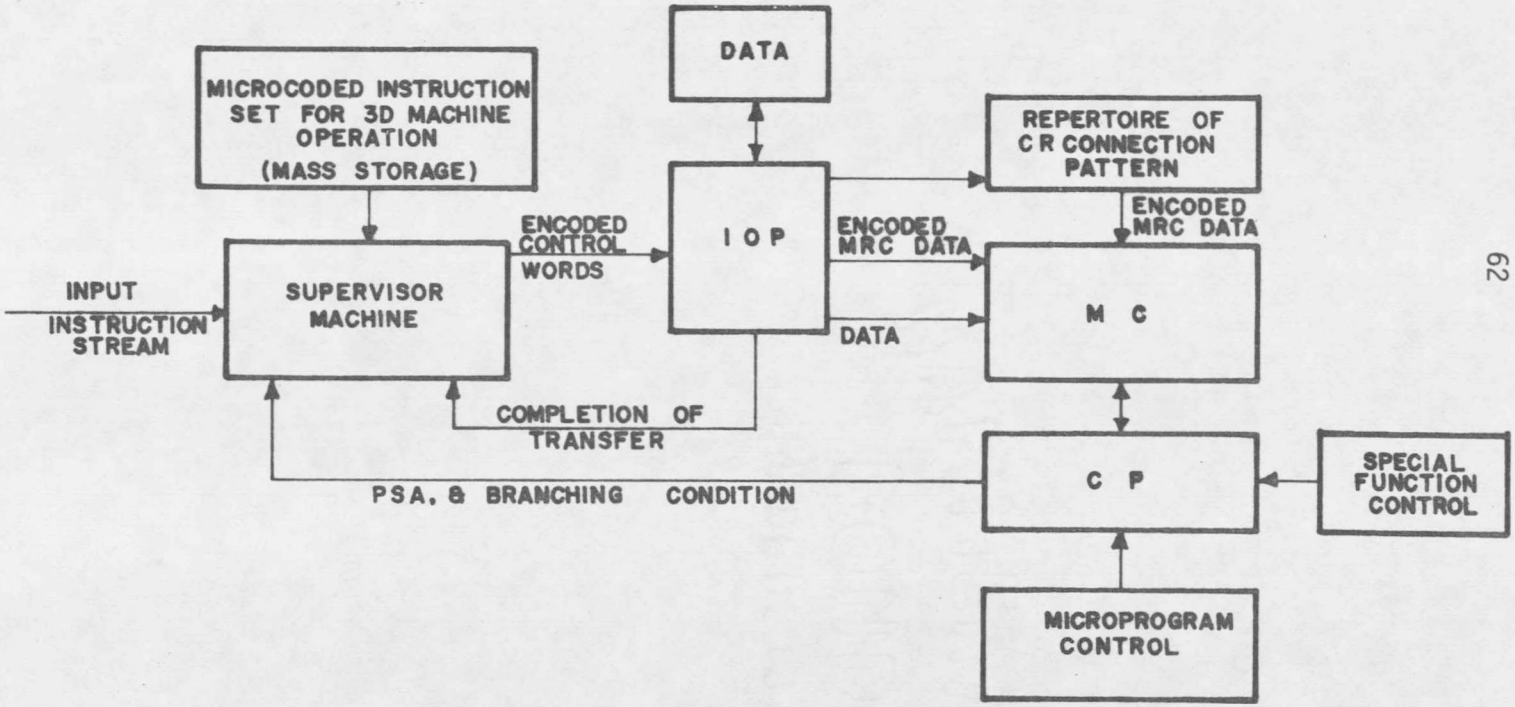


Figure 2.7 Control and Data Flow Diagram of the 3-D Machine.

ware (general or global control). Figure 2.7 is a flow diagram of various control actions and modules of the 3-D machines.

As can be seen from Figure 2.7, control is widely spread throughout the structure of the 3-D machine. Connection pattern encoding during the compilation phase can be considered as part of the control. These encoded control words are loaded in the MRC registers of the MC. They go through an activation process either in the MC to set the connection gates, and/or in the CP to choose the appropriate ALOC's.

Another control feature of the machine is the X, Y, and Z decoders which form a complete address mapping of the 3-D machine; this is fully described in Section 2.3.4.2. In a multiprogramming environment, several decoders of each type are envisioned. This replication facilitates intermixing the instruction streams of several problems without the need of reloading a single set of X, Y, and Z decoders when switching control from one program to another. This feature permits one part of a plane to work on one program while non-overlapping parts of the plane are doing something else. Also, a MC plane can be executing a conductive transfer while another plane is performing a direct-in-cell transfer, etc.

Computation plane control within the CP unit is more involved than the control for the rest of the machine with the possible exception of the IOP. This control not only assures a proper sequence of

CP execution but also changes the instruction stream order, (e.g. when a branch condition is encountered by the control, the CP control examines the condition codes of all cells inside the specified boundaries and signals back to the supervisory machine to either execute the branch instruction or fall through to the next instruction in sequence). The CP control also initializes and terminates the execution of instruction streams to be executed in the CP associated processors which include the microprogrammed arithmetic/logic processor, the search for maximum or minimum processor (whose function is described in Section 4.5), and any other special task processor to be appended to the CP according to the desired type of application algorithm to be optimized on the machine.

The IOP has its own control with a complexity which can be varied to achieve a wide range of I/O speed. A single IBS, single OBS represents the simplest I.O structure with an I/O speed equal to that of an equivalent serial machine while an IOP with multipled input stacks, and multiple output stacks can achieve a higher I/O rate.

Figure 2.8 shows the control and supervisory structure of the 3-D machine. It is simplified to give a general idea of the machine's operation.

2.3.4.1 Memory Cell Control

Memory cell control is contained in the cell control word (MRC).

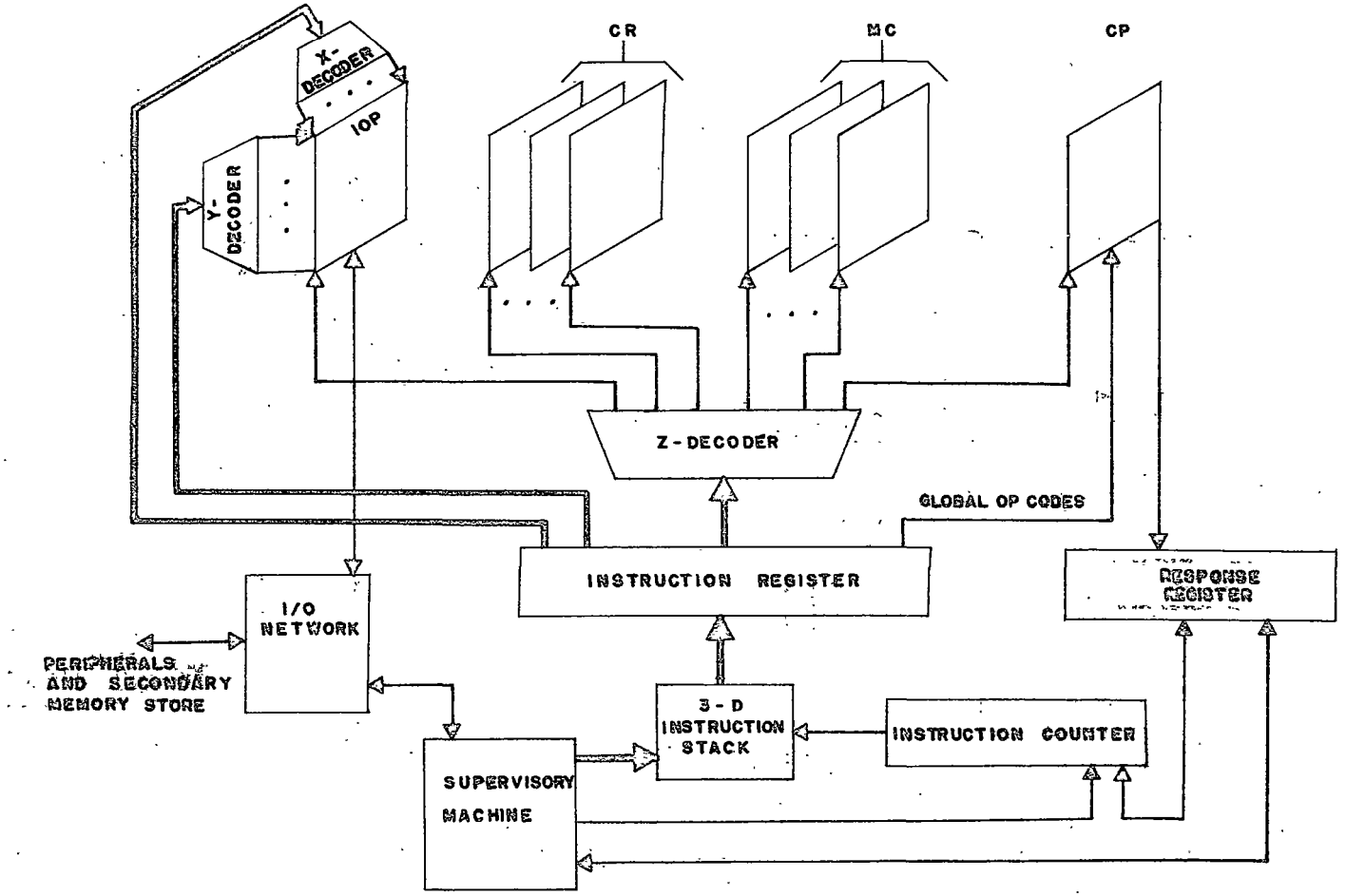


Figure 2.8 Control and Supervisory Structure

MRC is a double-word register (64 bits) which contains coded information about:

1. The gate settings for intraplane transfer connections (42 bits).
2. The ALOC (8 bits).
3. Interplane transfer address (4 bits).
4. The cancellation number of the cell (4 bits).

The cancellation number will be described in Section 3.2.2. A total of 58 bits of the MRC register are used, the remaining 6 bits are used for parity checking or for future control expansion as continued development may dictate.

The intraplane connections of the cells are designed in such a way as to permit data to be stored in the register and/or bussed through non-destructively. Figure 2.9 shows the horizontal, vertical, minor-diagonal, and inter-cell connections of register MR1. Intraplane connections for MR2 are similar.

Gates G_1, G_4, and G_7 are bidirectional and function to permit data to be bussed through without entering the cell. The other gates in Figure 2.9 are unidirectional; there is a G_{is} and G_{id} where G_{is} is closed when the cell is acting as a source of data in direction i, while G_{id} is closed when the cell is acting as a destination for data coming from direction i. These two functions are mutually exclusive for a given i.

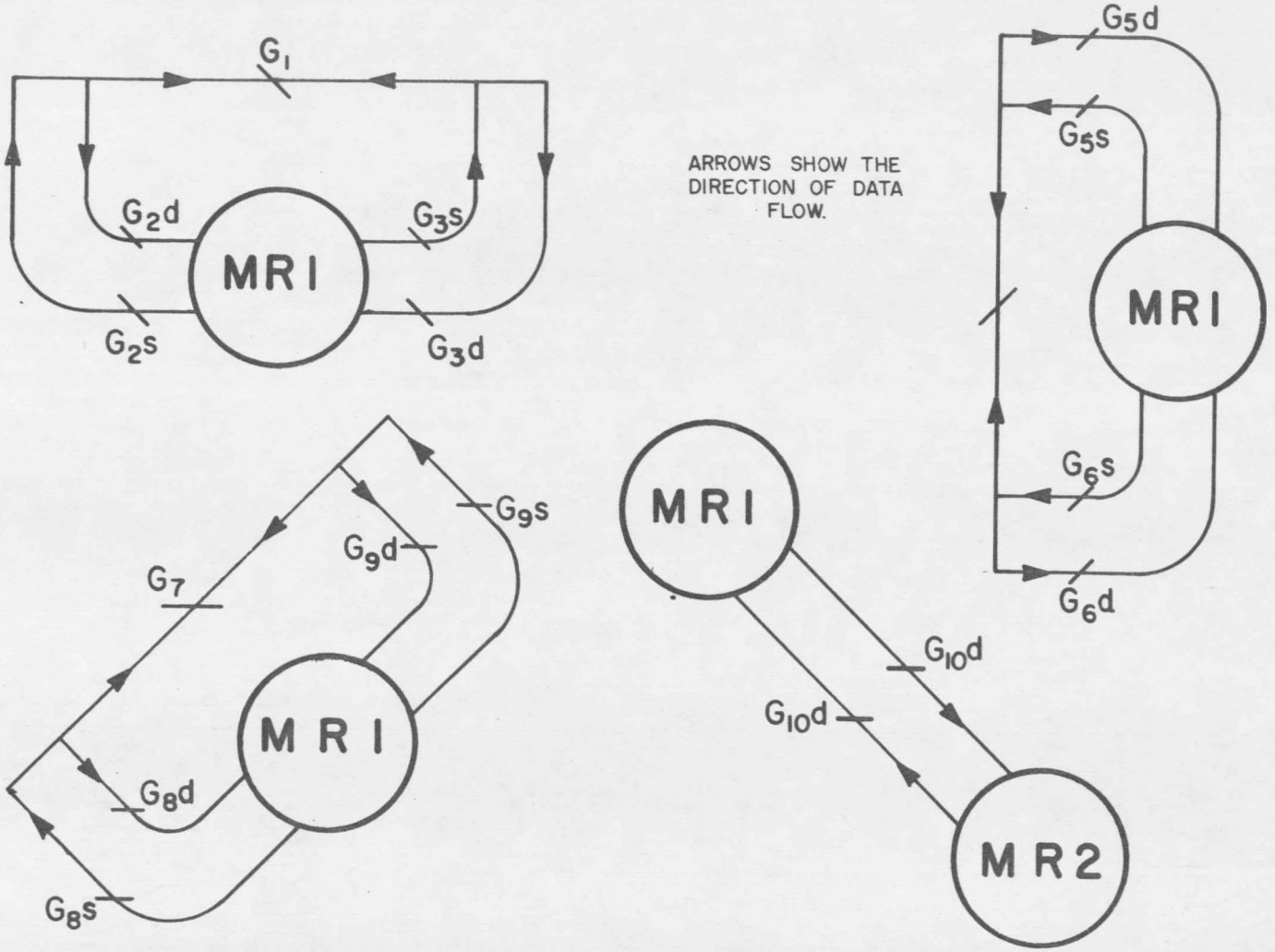


Figure 2.9 The horizontal, vertical, minor-diagonal, and inter-cell connections of MR1.

Since information is allowed to change direction during transfer (e.g., from a horizontal bus to a vertical bus), and because transfer in any given plane of the 3-D machine is a one-shot operation (there is no temporary storage during a data transfer), connections between the different busses are essential. Four bus-to-bus connections are provided which add four additional bidirectional gates G_{11}, G_{12}, G_{13}, and G_{14}. Figure 2.10 shows the bus-to-bus connections of the same cell.

A complete description of valid connection patterns without conflicts in data paths is presented in Section 3.2.1. Appendix A tabulates the connection patterns and the corresponding gates to be closed. The results of applying the Quine-McCluskly algorithm (30) on the combinational functions of the gates are listed in Appendix B.

A special logic unit called the Cancellation Logic is included as part of each memory cell control. Figure 2.11 shows the Cancellation Logic for one cell. This unit either enables or inhibits operation in any given cell in accord with a cancellation algorithm. A complete exposition of the algorithm appears in Section 3.2.2.

The cancellation number (CN) of a cell is stored in the MRC register of that cell as a 4 bit binary coded number representing the set [1,2,4,9]. At execution time a comparison network (the Cancellation Logic) compares the CN with an externally supplied number CCi = 1,2,3 which is in the range 0 through 15. If CCi covers CN (i.e., CN is one of the binary prime sum constituents of

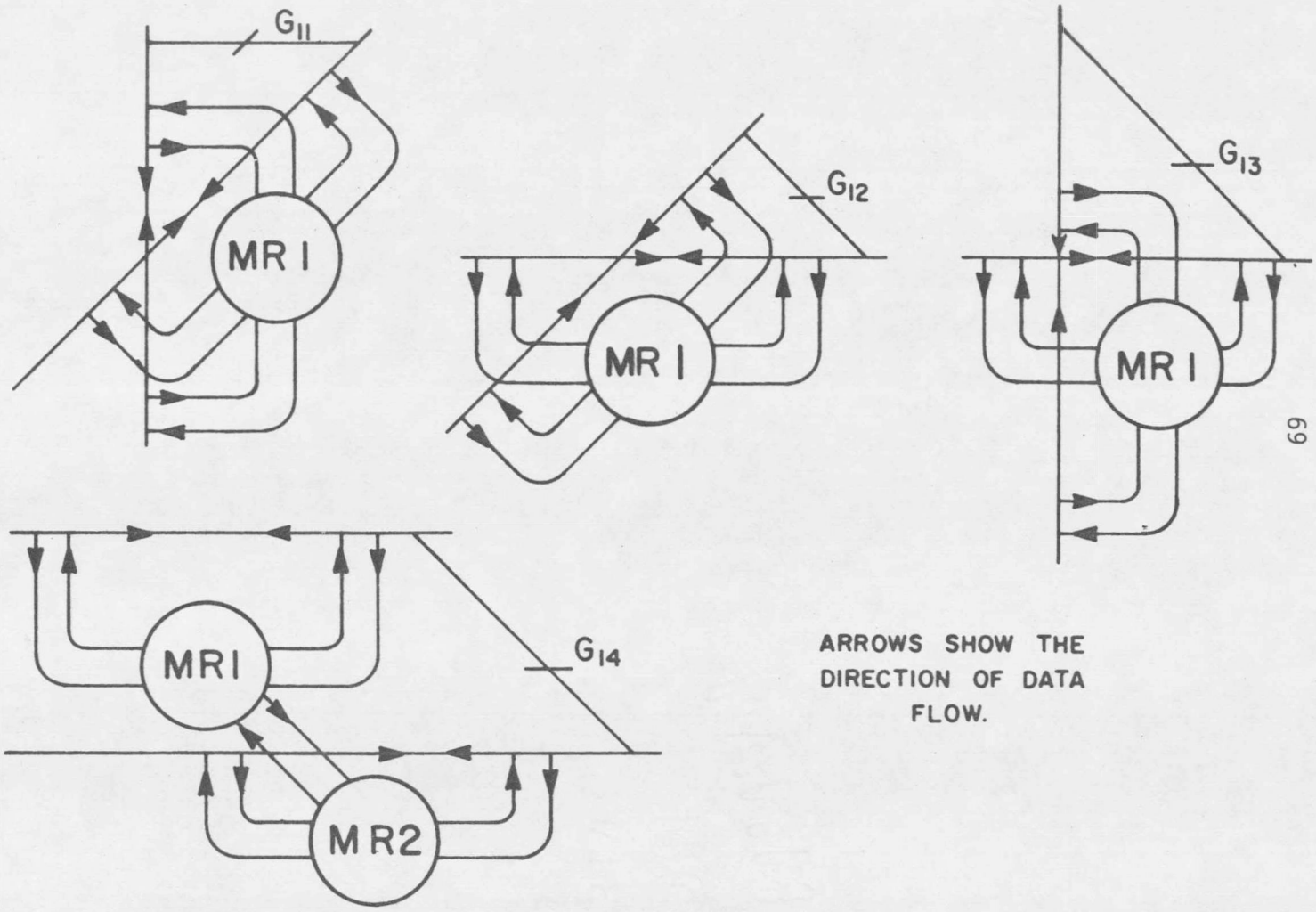


Figure 2.10 Bus to Bus Connections

CCi), an inhibit signal is sent to the register for which this matching occurs. Figure 2.11 shows the Cancellation Logic for one cell.

2.3.4.2 The X, Y, and Z Decoders Control

As mentioned previously an address mapping is performed by the X, Y, and Z decoders every time a 3-D instruction is executed.

Consider a typical 3-D instruction, e.g.

(02:05;03:07) TRAN(00,00) MR (03) MR (05)

The X-decoder will decode 02 as IR (the index of the first row of the array), and 05 as JR (the index of the last row of the array). The Y-decoder will decode 03 as IC (the index of the first column of the array), and 07 as JC (the index of the last column of the array). The Z-decoder will decode 03 as II (the source plane), and 05 as JJ (the destination plane).

The X and Y decoders are wire (or non-conductively) connected to all planes of the 3-D machine; their function is to partially enable all cells inside the array boundaries in all planes of the 3-D machine. A complete enabling of these cells is achieved by the Z-decoder and the rest of the 3-D instruction, (the TRAN instruction in this example).

In the previous example all cells of rows 2 to 5 and columns 3 to 7 are partially enabled to receive or send information. The

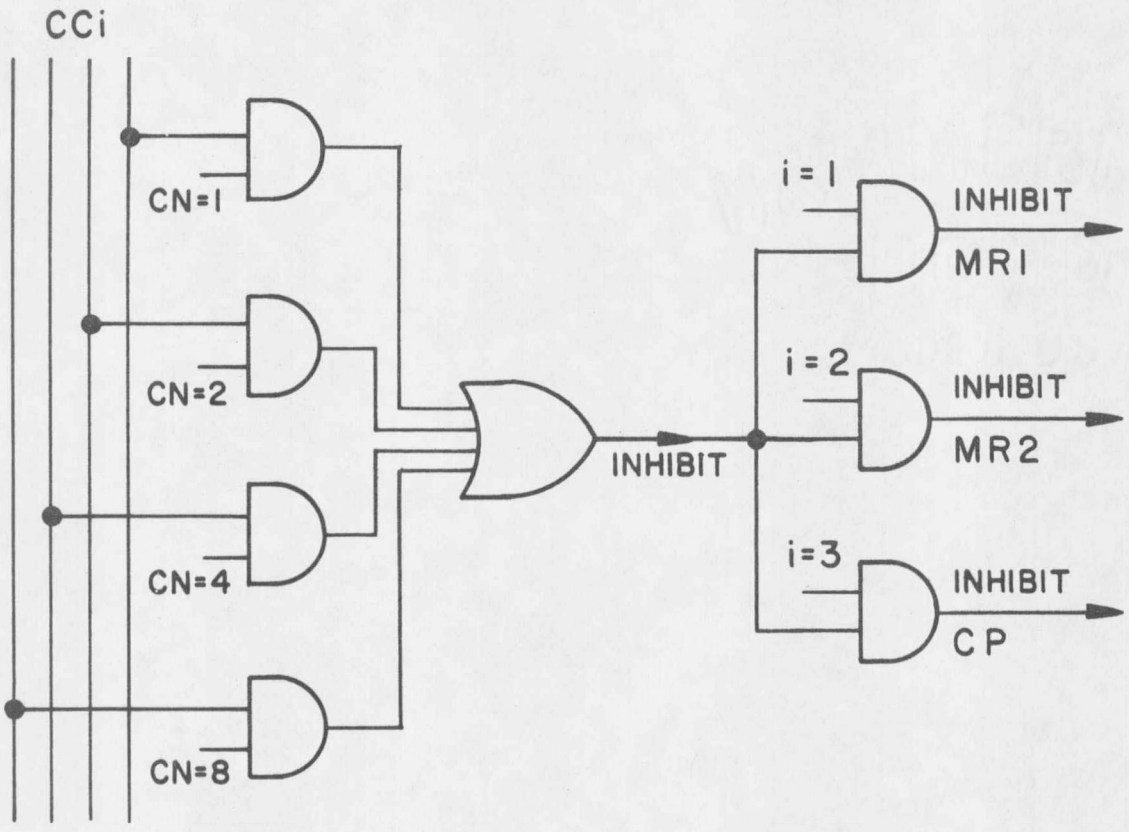


Figure 2.11 Cancellation Logic.

Z-decoder together with the 3-D instruction TRAN causes non-conductive transfer from memory plane 3 to memory plane 5 which completes the instruction execution.

In a multiprogramming environment more than one of each decoder type may be operating simultaneously. A program identification address is appended to the set of X, Y, and Z decoders to distinguish different programs in the multiprogramming environment.

In a fault-tolerant operating environment such as is discussed in Chapter 5, the X, Y, and Z decoders must be able to store the addresses of faulty rows/columns/planes of the MC. This information is used to isolate the faulty rows/columns/planes as described in Section 2.3.4.4 on IOP control. The users program resource demands can thus be met without any previous knowledge of faulty cells. The machine is reconfigured such as to avoid using cells in error, but the user does not have to know that the actual array assigned to this problem is a scattered set of cells physically.

2.3.4.3 Miscellaneous Counters and Registers

As shown in Figure 2.9, for proper operation of the machine an Instruction Register, a Response Register, and an Instruction Counter are required to run a single stream of instructions. The Instruction Register and Instruction Counter are typical of those found in any conventional machine. As for other control registers,

it may prove necessary to have a set of each to handle the multi-programming situation. The Response Register or set of Registers is needed because of the asynchronous nature of the machine. The Response Register receives the end of instruction execution signal from the 3-D machine and increments the instruction counter accordingly.

2.3.4.4 IOP Control

The main control features of the IOP are the HCSR and the VCSR; these shift registers do not carry data. When an I/O operation is taking place, the X-decoder is connected to the VCSR and supplies it with IR and JR (the index of the first and last row to perform I/O operations), and the addresses of all nonfunctional rows due to malfunctioning cells. The Y-decoder is connected to HCSR and supplies it with IC and JC (the index of the first and last column to perform I/O operations), and the addresses of all nonfunctional columns due to malfunctioning cells. The VCSR-clock-decoder drives VCSR with a clock signal which is equal to the HCSR clock divided by the number of columns to receive input.

Figure 2.12 shows the sequence of VCSR signals for an array of rows IR to JR. The bus intersections without a dash are indicative of the fact that gate VCSR(I) of that intersection is open. The gates (VCSR(IR), VCSR(IR+1), ..., VCSR(JR)) are closed in sequence with a

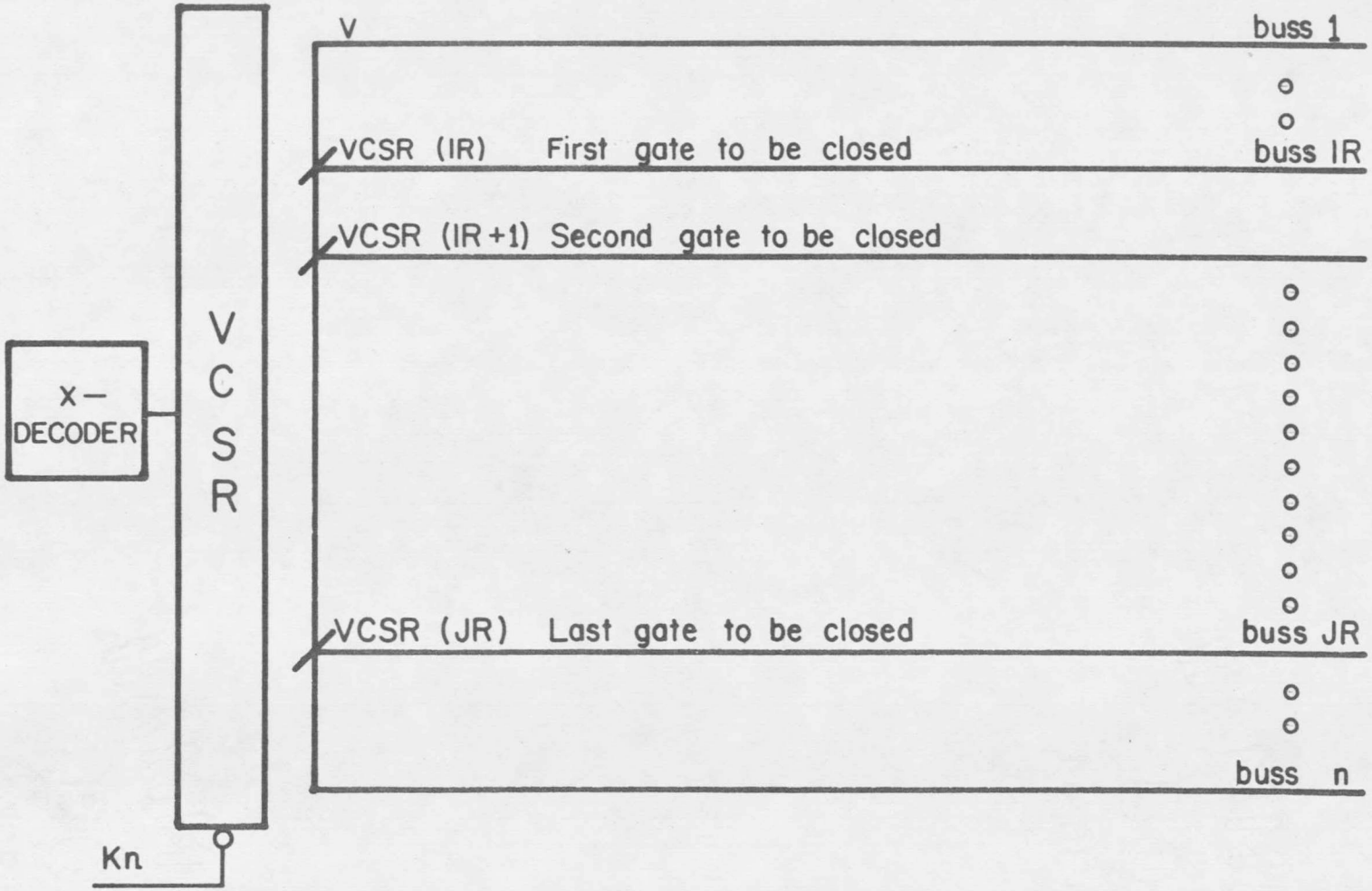


Figure 2.12 Sequence of VCSR Signals.

clock signal clockn (KN), where KN has a period n times the period of the HCSR clock signal, and n is the number of columns to receive input.

The action of the HCSR is described in a similar fashion, see Figure 2.13.

The VCSR is a regular serial-input, parallel-output shift register which has been modified to receive the address of a faulty row and inhibit the signals which activate the gate to the horizontal bus leading to that row. These extra inputs are used to achieve a fault-tolerant input/output operation. Two inputs are connected to VCSR through a selection matrix. IR is connected to the row selected as the first row to receive input. This single input appears every clock cycle on a successive output IR, IR+1, ... etc. JR' which is equal to JR+ (the number of faulty rows to be excluded), is connected through another selection matrix to the row next to the last row to receive input. JR' is connected to inhibit the output of all rows of the VCSR next to the last row selected. This way we insure that the gates leading to the busses feeding these rows will be permanently open. Figure 2.14 is a schematic of a typical VCSR/HCSR shift register connections.

The HCSR is similar to VCSR except for the clock signal as described before.

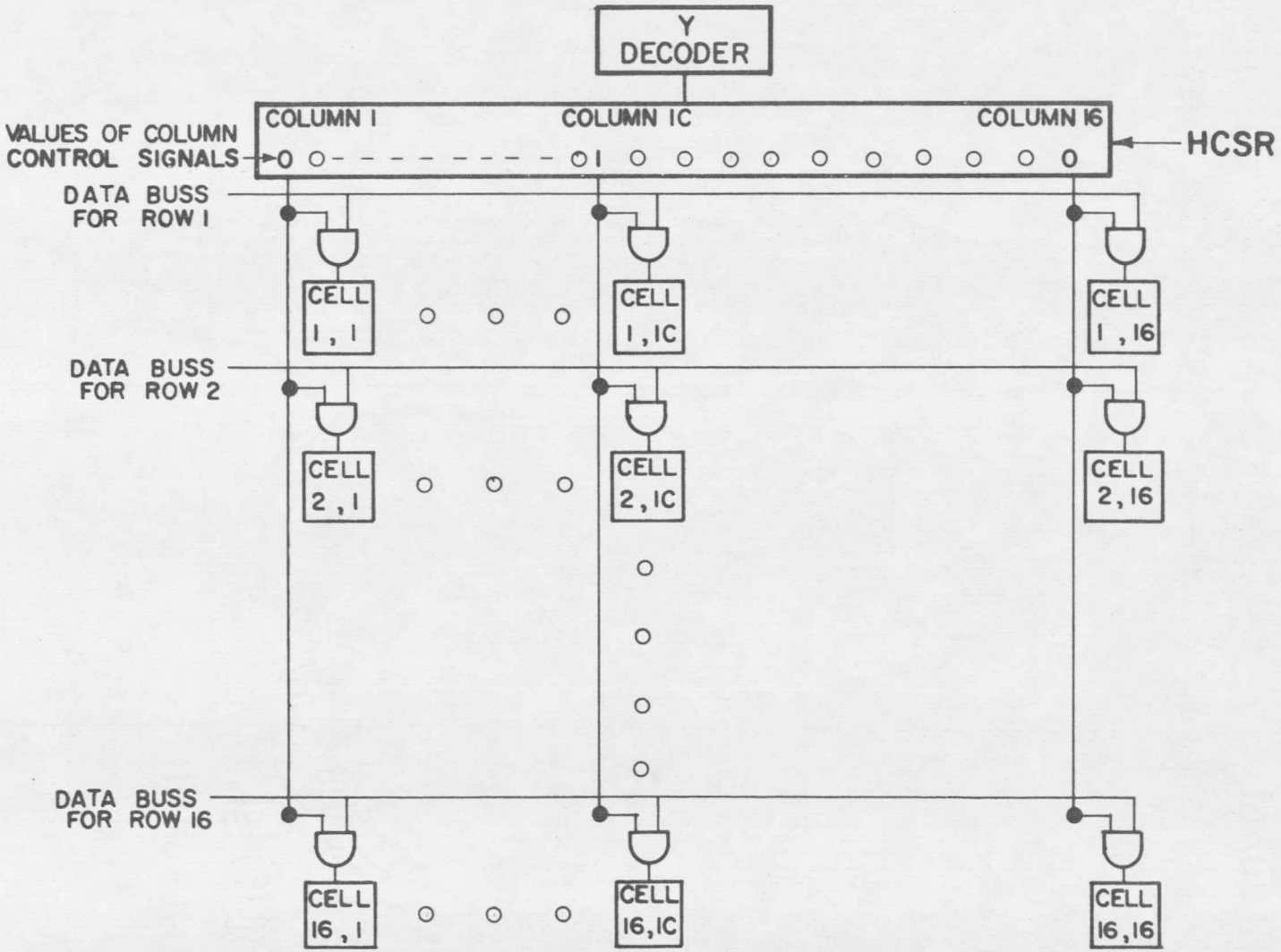


Figure 2.13 Sequence of HCSR Signals, showing column IC only having a value of 1.

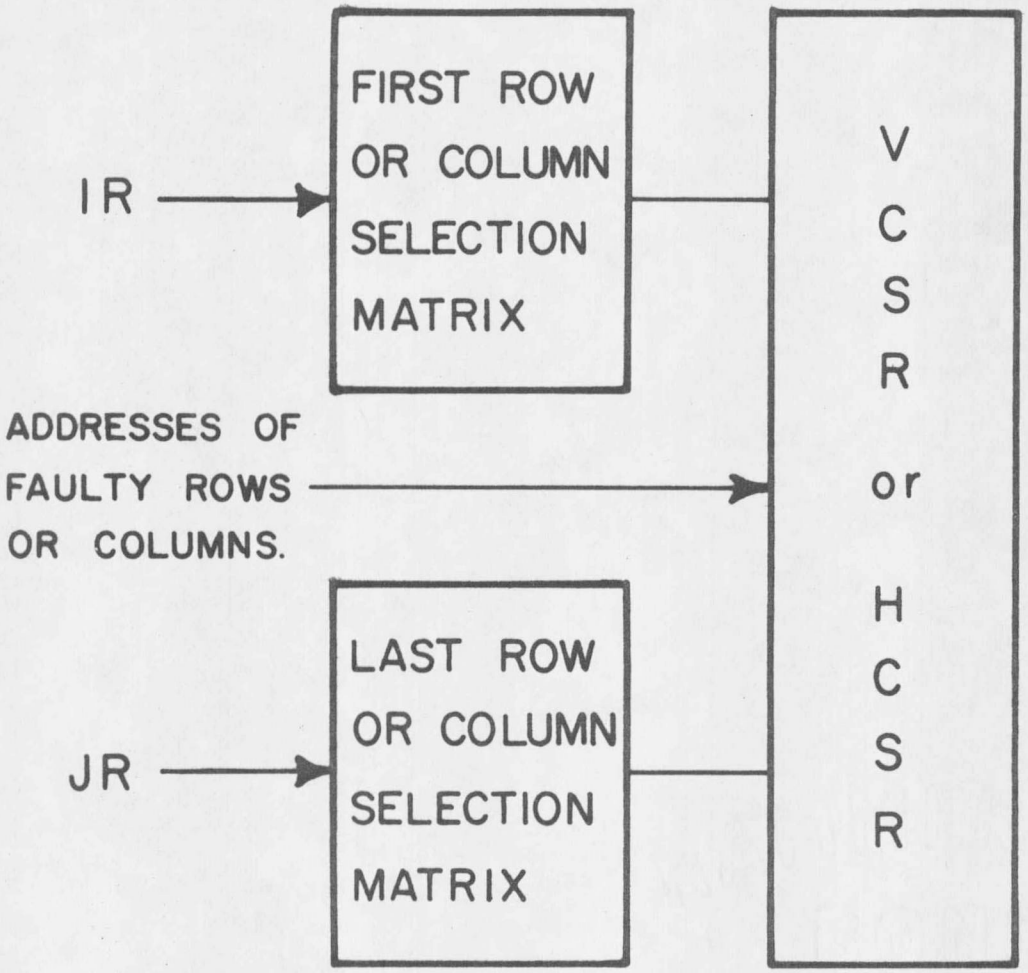


Figure 2.14 Schematic Diagram of a Typical VCSR or HCSR Connection.

A logic gate diagram of a typical VCSR/HCSR is shown in Figure 2.15. If a single row/column is to be inhibited, the address for this row/column is sent to the fault bus and a permanent inhibit signal is issued to isolate this one row/column. For the multi-fault situation, the address of faulty rows/columns are sent successively to the fault bus. An isolation accomplished signal is sent back to the X/Y decoder to acknowledge receipt of the fault address and isolation of the faulty row/column. The next fault address is sent subsequently to the fault bus.

A schematic diagram of the control signals of the simplest IOP design is shown in Figure 2.16.

2.3.4.5 Methods to Increase the I/O Rate

The 3-D machine performs I/O through the supervisory machine, or the secondary storage system, or transducers connected to the machine to accomplish some real time tasks. I/O speed can be enhanced through different IOP designs which implements one or more of the following:

1. A transducer/cell of IOP presents the fastest and at the same time the most expensive technique. See Figure 2.17.
2. For an IOP of n columns, an I/O rate n times that of the original design can be achieved if one uses an IBS, an OBS, and a VCSR per column as shown in Figure 2.18.

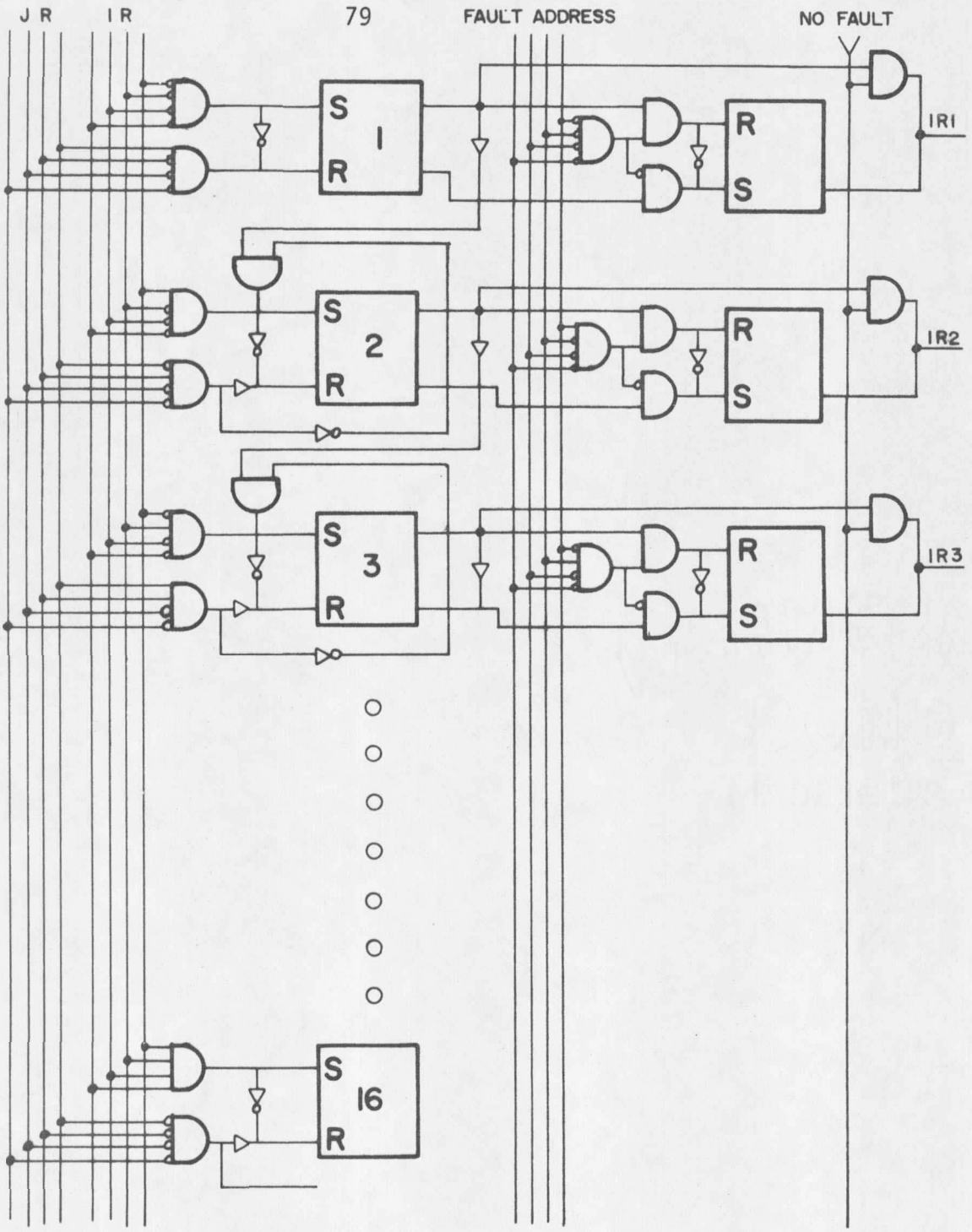


Figure 2.15 A Typical Logic Diagram for VCSR or HCSR.

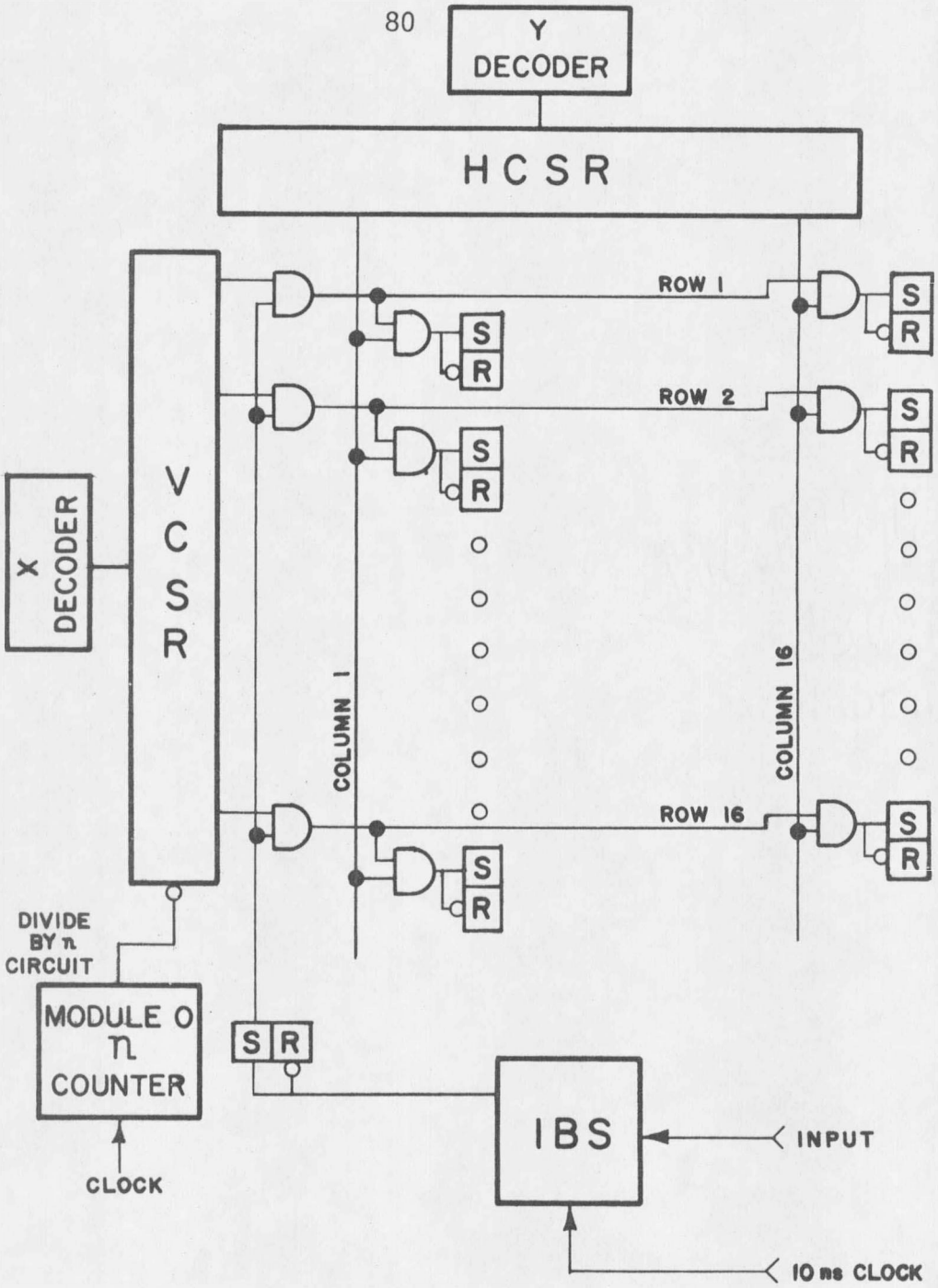


Figure 2.16 The Logic Diagram for the Control Signals of IOP.

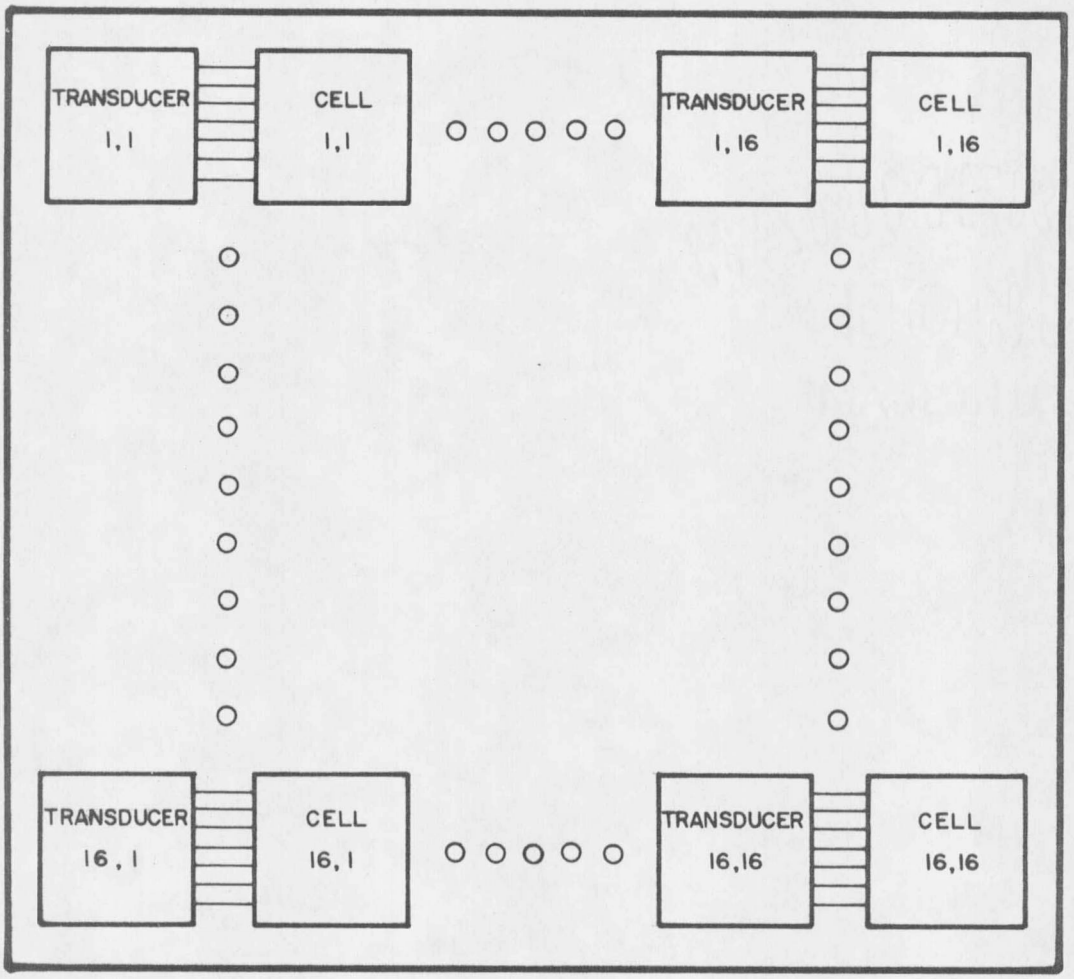


Figure 2.17 A transducer/cell IOP.

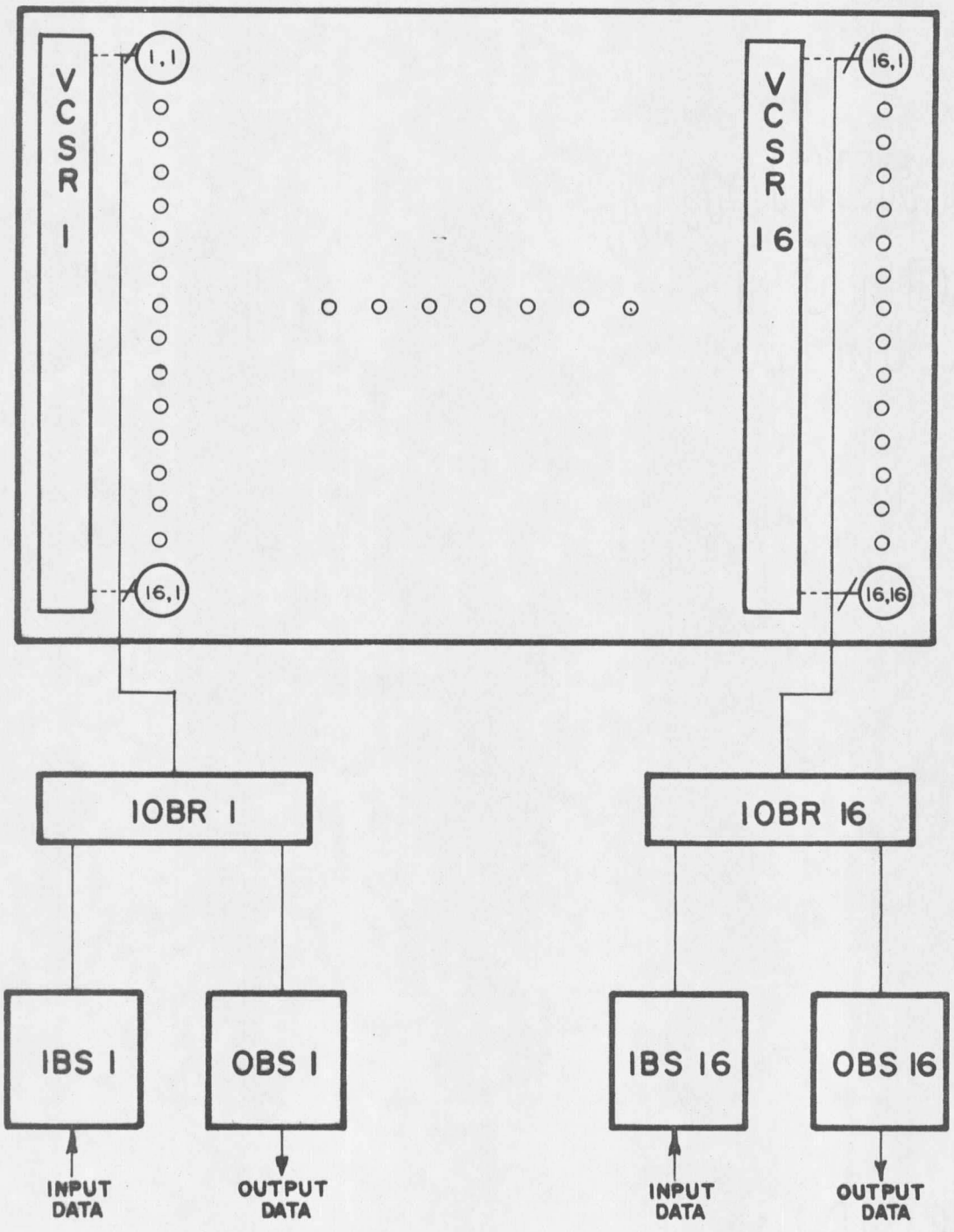


Figure 2.18 An n Times Faster IOP than shown in Figure 2.5.

3. A time-multiplexed IOP using n VCSR's, n IBS's, and n OBS's is capable of I/O rate n times that of the original IOP. I/O conflicts are very likely and have to be guarded against through extra hardware and clever programming. An example of a time-multiplexed IOP with n=4 is shown in Figure 2.20. Referring to Figure 2.19:

a_{ij} = VCSR(k,j).HCSR(k,j) signifies the control signal for an I/O operation of the cell in row i and column j through stack a.

b_{ij} = VCSR(k,j)).HCSR(k,j) signifies the control signal for an I/O operation of the cell in row i and column j through stack b.

c_{ij} and d_{ij} are defined similarly for stack c and stack d respectively.

4. Zoned IOP, in which case the IOP is divided into several subplanes (zones). Each zone is treated as an IOP similar to the original design IOP. For an IOP of m zones an I/O rate m times that of the original IOP is achieved. The cost function for an m zone IOP is proved to increase as m, based upon the gate count of shift registers required to accomplish zone control. Figure 2.20 is a block diagram of a zoned IOP.

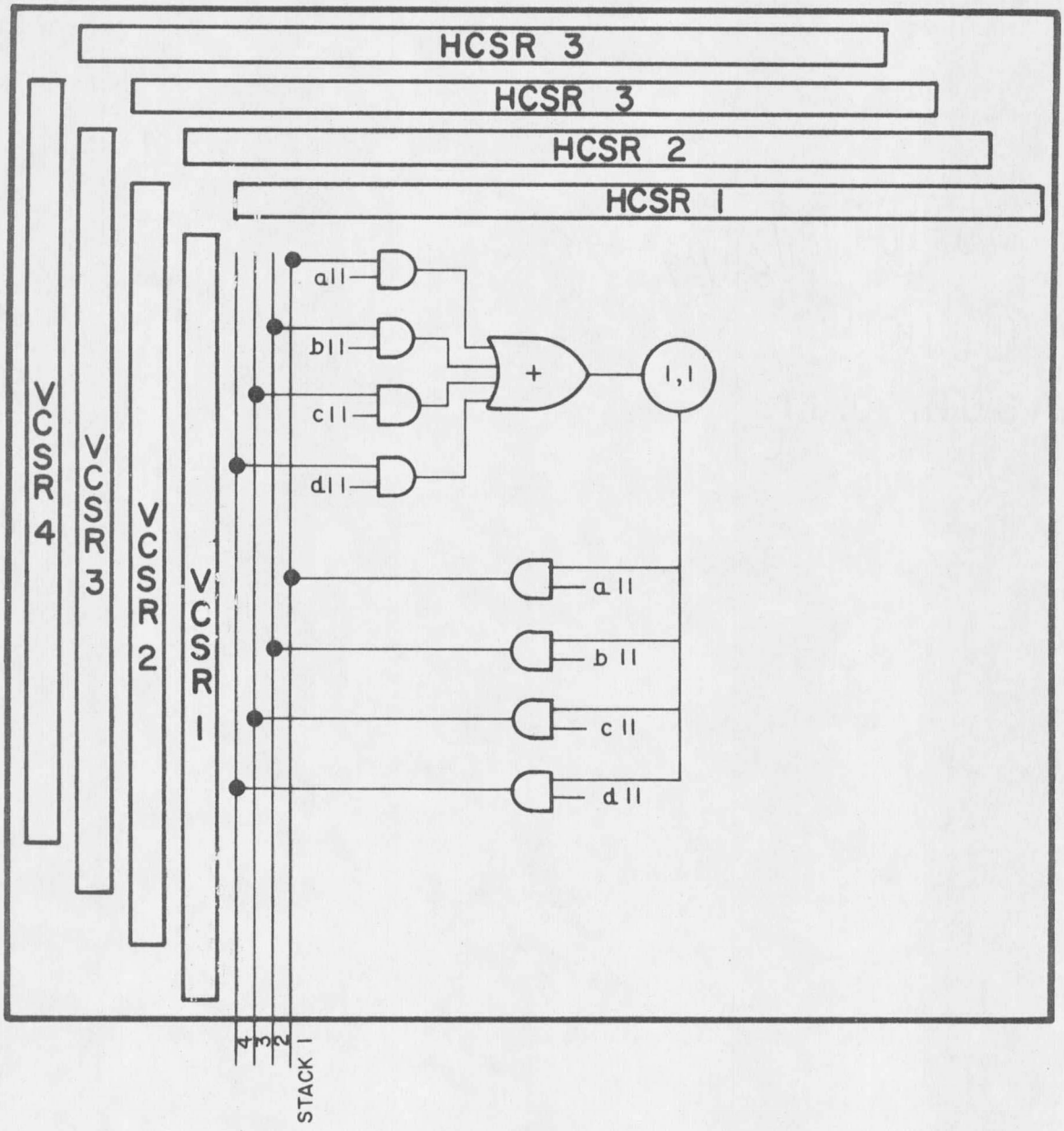


Figure 2.19 Time-Multiplexed IOP organization.

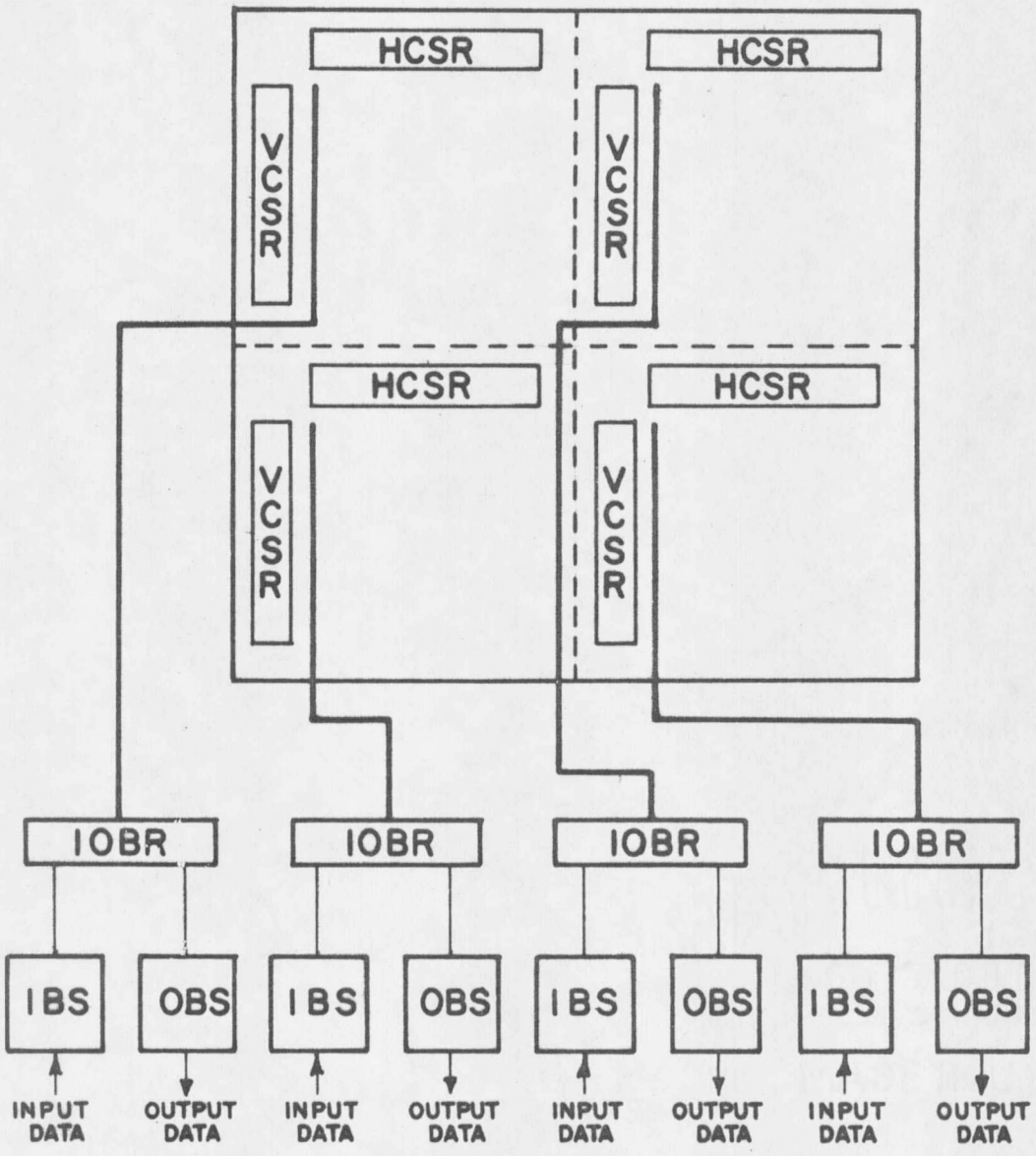


Figure 2.20 Schematic Diagram of a Zoned IOP.

CHAPTER 3

MACHINE LANGUAGE FOR THE 3-D COMPUTER

3.1 Assembly Language Structure

The assembly language (SIM3D) to be described is highly correlated with the hardware features of the 3-D computer. In fact the assembly language of the machine is the most primitive language of the machine (there is no software assembler for the 3-D machine). SIM3D is a register-transfer-level language, and since it is tightly bound to the hardware, it is appropriate to mention the five general functional parts of the machine. These have been described in detail in Chapter 2. These functional parts are: the Memory Cube (MC), the Computation Plane (CP), the Input-Output Plane (IOP), the Connection Register Read Mostly Store (CR), and the Control and Supervisory Section.

There are two distinct divisions of SIM3D programming having to do with (1) memory cube configuration instructions, and (2) operational instructions. In a conventional serial machine there are no comparable machine configuration instructions since the configuration is fixed upon manufacture. Although the operational instructions of the 3-D machine have a quite different format from the instructions of most other machines, they are relatively conventional in that they cause data transfer, arithmetic and logic operations to be carried out.

An additional innovative feature related to the configuration instructions is the introduction of cancellation numbers (CN) and

cancellation codes (CC), having to do with the execution or inhibition (cancellation) of transfer operations in the MC and/or I/O operations in IOP and/or arithmetic/logic operations (ALO) in the CP. An execute-inhibit mask matrix is generated through interaction between the CC of the 3-D instruction and the CN stored in the cell. Cancellation codes and cancellation numbers are covered in detail in Section 3.2.2. A mask matrix is an array of 0's and 1's which is generated in the cells before execution of an instruction in the cell. A cell cannot execute its instruction unless it has a 1 for the mask matrix element corresponding to that cell. The mask matrix elements are the complements of the output from the cancellation logic described in Section 2.3.4.1.

The 3-D operational instructions are a simple but powerful set. Each instruciton is a macro operation which operates on planes or parts of planes of the MC, the CP, and/or the IOP. A macro operation might be one of the following:

a. an input/output macro operation

b. an intraplane (conductive) transfer

c. an interplane (non-conductive) transfer

d. a branching operation

Although this categorization is not all inclusive, it is very descriptive of hardware features of the 3-D machine.

Arithmetic and logic operations are performed as sub-operations

of interplane transfer when the transfer is to the CP.

3.2 Memory Cube Configuration Instruction

An interconnection schematic of a 2x2 memory plane (excluding MRT) is shown in Figure 3.1. Each memory register is shown as a circle and possible data paths are shown as lines.

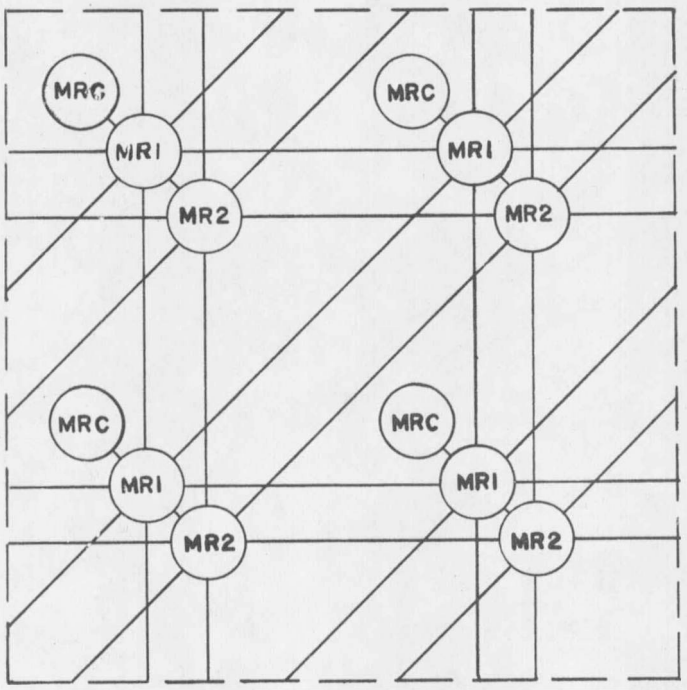


Figure 3.1 A 2x2 Memory Plane

It shows that with the single exception of the inter-register path between MR1 and MR2 of each cell, MR1 registers connect only to other MR1 registers and MR2 registers connect only to other MR2 registers. It is also significant to note that there are no edge connections in the array.

The third register in each cell is the control (MRC) register and is a double word length register containing the bits necessary to establish a particular interconnection structure for its pair of memory registers. MRC also has bits that specify arithmetic and logic operations and an execute-inhibit mode as will be described later.

It is observed that each MR1 register is adjacent to seven lines as is each MR2 register. A numbering scheme has been assigned for MR1 register lines as in Figure 3.2 (a) and a similar scheme for MR2 register lines in Figure 3.2 (b).

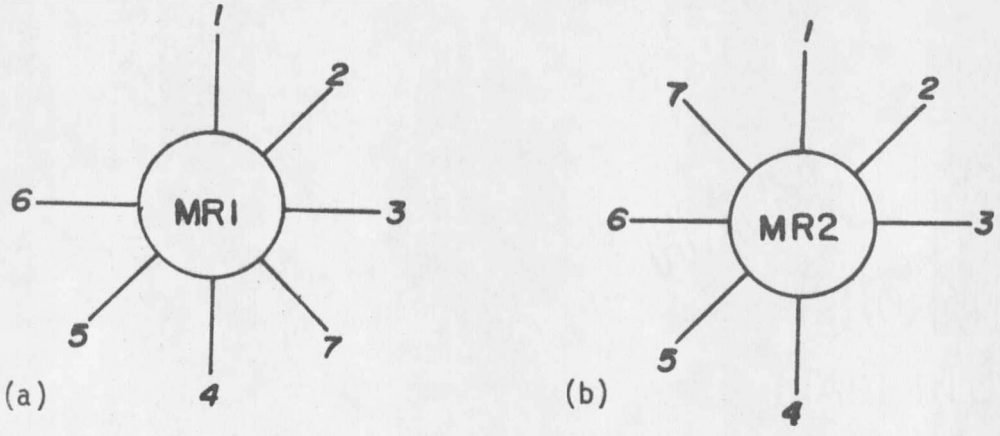


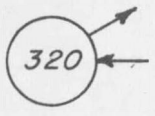
Figure 3.2

For each register, a three-digit code is used to specify (1) data input line, (2) data output line, and (3) any of several special bussing features to be explained. Input and output connection codes and their corresponding connections are shown in Figure 3.3 for MR1 registers. Only the first two digits, indicating input and output, are shown. The third digit (for special bussing) is indicated by a dash. For MR2 registers, the '7' line runs opposite to that shown, i.e. the connection between MR1 and MR2 is designated by a '7' in both cells.

3.2.1 Transfer Connections

In the register connection code, the first digit defines the input line, the second digit defines the output line and the third digit defines any connections in addition to those for load and transfer operations.

Specifically, if the third digit is a '0', no additional connections are to be made. Hence a 320 connection means input from the right and output to the upper right.



If the third digit is a '1', an additional horizontal bus connection is to be made. The first two digits still specify the input and output directions. Thus, a 511 connection has input from

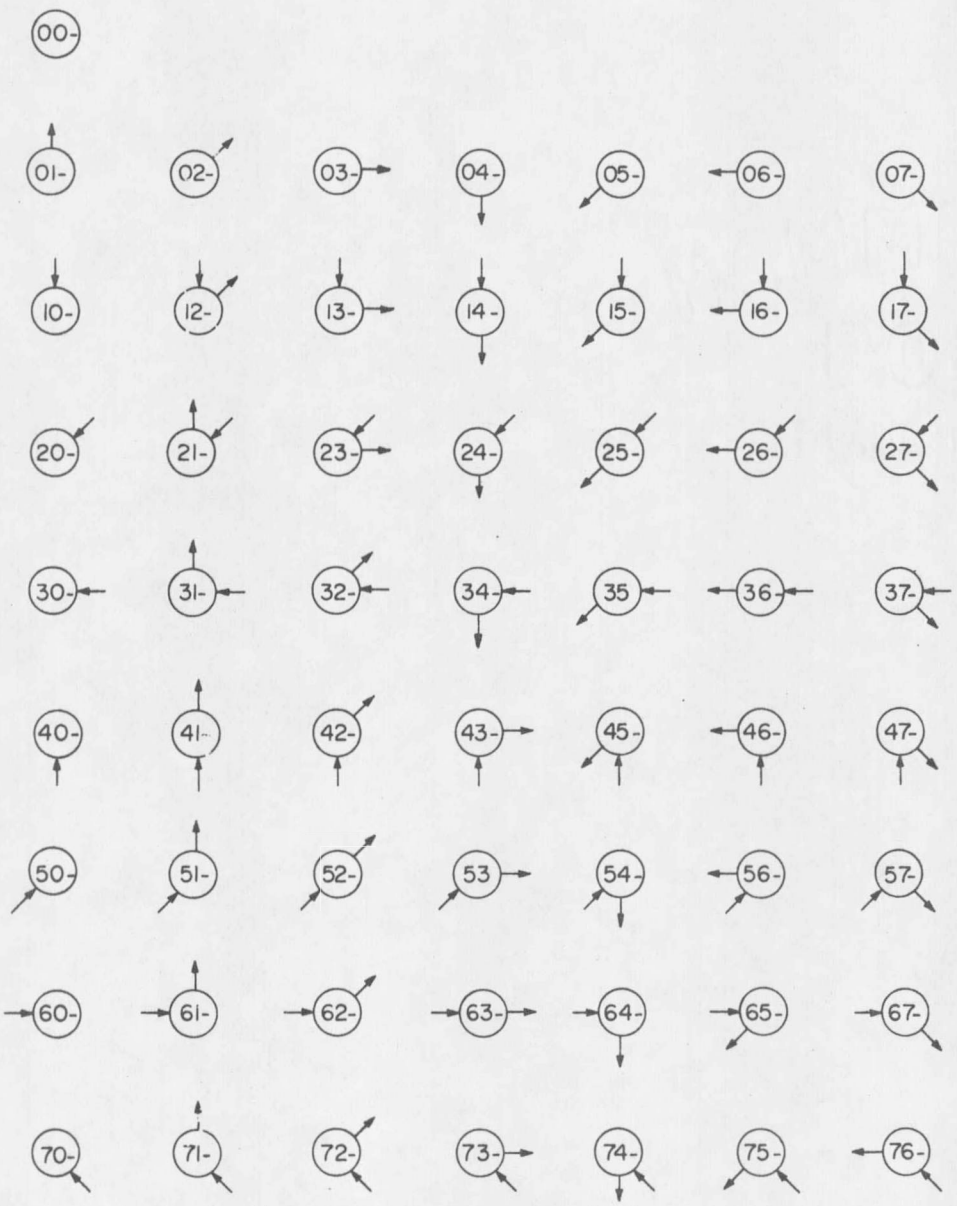
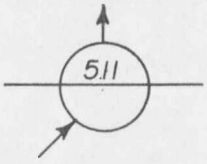


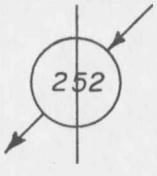
Figure 3.3 Register Connection Codes

the lower left, output to above, and a horizontal bus.

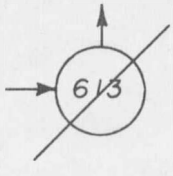


Note that a 621 connection is illegal since it creates a conflict of the input and bus line designations. Any such conflicting designation is deemed invalid.

A third digit of '2' provides a vertical bus in addition to the already specified input and output. A 252 connection is shown.



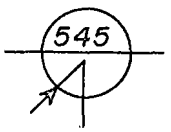
A '3' as the third digit provides a diagonal bus as shown.



If the third digit is a '4', a load and bus operation is performed. The input appears on the specified line, is loaded into the register and also is bussed through to the output line. The original contents of the register are lost, being destroyed by the load operation. A 124 connection is shown.

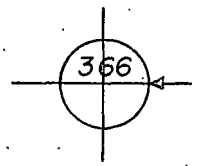


If the third digit is a '5', a horizontal bus is added to the load and bus connection. A 545 connection is shown.

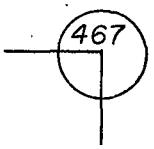


Because of conflicts, such connections as 465 are invalid.

A '6' as third digit adds a vertical connection to the load and bus. An example of a 366 is shown.

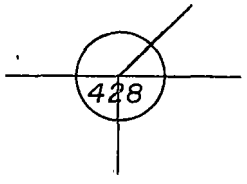


A logical completion of this progression would be to define a --7 connection as one with a diagonal bus in addition to the bus and load connections. However, we have only 10 digits to use in this spot and 12 connections if the progression is followed (another class of connections follows). The two bus patterns dropped are the diagonal ones in each remaining class since what limited experience we have indicates that they are likely to be the least used. So the 7 as a third digit is used to indicate complete bussing only without loading or transfer. An example is 467 as below.



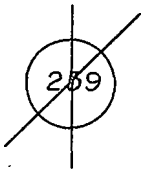
Since bussing is bidirectional, the first two digits (indicating input and output under normal conditions) can be interchanged and a 647 connection is the same as a 467.

The third digit of '8' indicates the addition of a horizontal bus to a bus defined by the first two digits. An example is 428 (or 248).



Again, such conflicting connections as 538 (or 358) are invalid.

Finally, if the third digit is a '9', a vertical bus is added to the bus defined by the first two digits. An example is 259 (or 529).



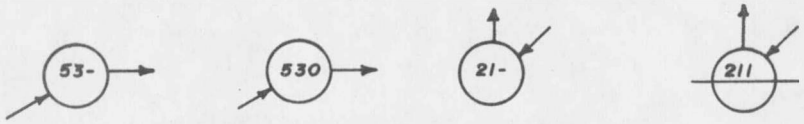
It is recognized that not all combinations are included here, but there is an undeniable richness nevertheless. Should there be some problem or class of problems that is insistent upon a change, one will surely be made. As it stands, the simulator is set up to handle the register connections shown.

A complete input-output and special bussing code for each memory cell is then two three-digit numbers in which the first three-

digit number specifies the input line for MR1, the output line for MR1 and special bussing for MR1, respectively. The array of three-digit number pairs for a given plane is called a principal pattern. A complete specification for a memory plane is a principal pattern plus an arithmetic-logic operation code (ALOC) for each memory cell. The ALOC may vary from cell to cell, providing a computation mode called the autonomous mode. More will be said of this in the section on arithmetic-logic operation codes.

A principal pattern for a simple 4x4 matrix transpose is shown in Figure 3.4, in which interconnection codes are written directly inside each MR1 and MR2. MRC is not shown. One data transfer cycle is required to effect the complete transpose compared to twelve for serial operation.

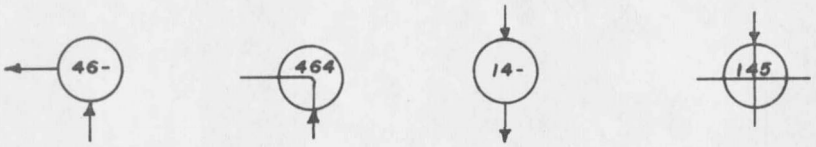
A summary of the bus patterns is shown below. With each of the new patterns is shown the prototype connection from which it is derived.



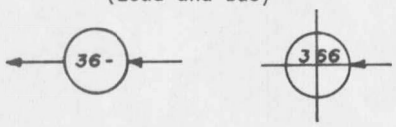
(No Bus) (Horizontal Bus)



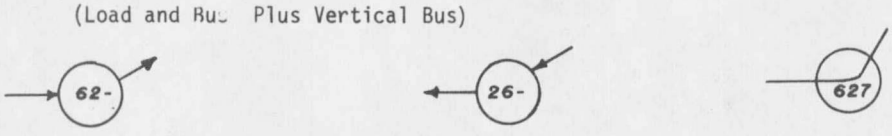
(Vertical Bus) (Diagonal Bus)



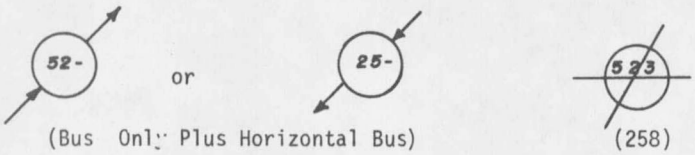
(Load and Bus) (Load and Bus plus Horizontal Bus)



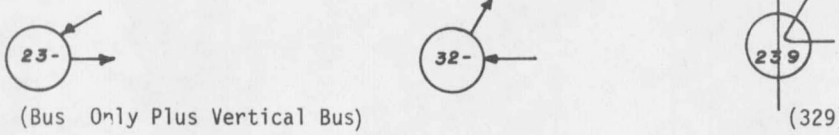
(Load and Bus Plus Vertical Bus)



(Bus Only) (267)



(Bus Only Plus Horizontal Bus) (258)



(Bus Only Plus Vertical Bus) (329)

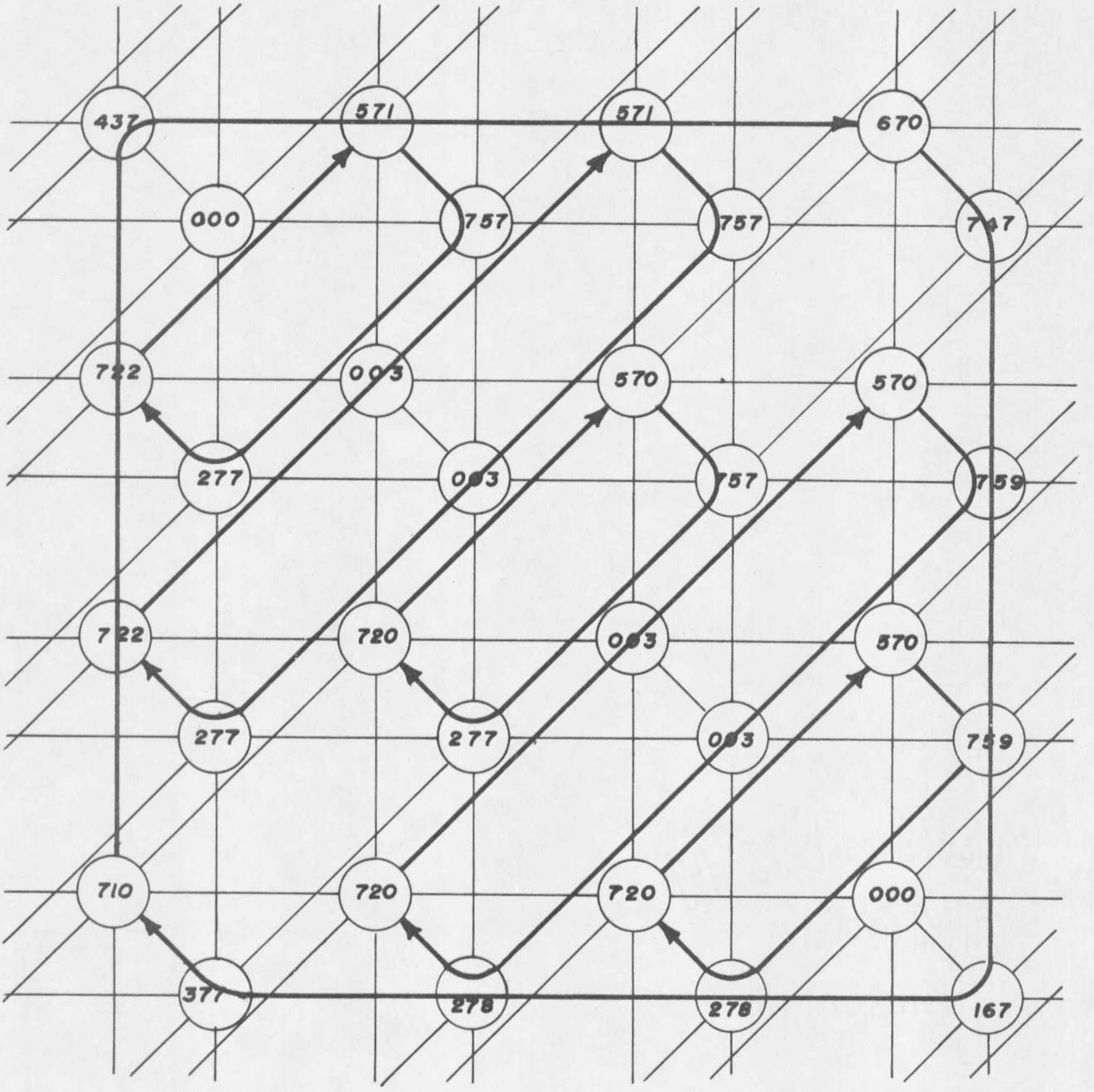


Figure 3.4 A 4x4 Matrix Transpose Pattern.

3.2.2 INSTRUCTION INHIBITION, CANCELLATION NUMBERS AND CANCELLATION CODES

It is a common practice in array machines to have some mechanism whereby an instruction to the array (each module in the array receiving the same instruction in most machines) can be either executed or ignored by individual modules depending on the setting of bits in an execute-inhibit mode register in each module. These mode registers have values that can be set by programming and/or data value, thereby providing a dynamic aspect in executing patterns. ILLIAC IV (4) and the SOLOMON computer (18) are examples of array machines having this general feature. The effect is to provide a partially configurable computing pattern that is useful in array arithmetic.

The 3-D machine also has a mechanism for dynamically specified instruction execution and inhibition. As in some other schemes, it involves a condition that is set in each module and a global standard against which each module's condition is measured. Those modules meeting the global standard execute; those not meeting it are inhibited. The module condition is represented by an integer called the cancellation number (CN) out of the integer set [1,2,4,8] (powers of 2). A memory plane of the 3-D machine will have a set of CN's associated with it (one CN per cell). CN's are generally not the same in all cells because it is the variation in CN's that gives the scheme its power.

The CN for a given cell is assigned automatically on the basis

of four possible combinations of MR1 and MR2 with respect to their having inputs or not having inputs. Note that the CN assigned is entirely geometric in nature, being unrelated to the data in the MR1 and MR2 register. The effectiveness of creating an execute-inhibit mask based on interconnection geometry exists because very often there is a correspondence between data transfer patterns and subsequent data processing operations. The CN assigned to a given module is as follows:

a. Neither MR1 nor MR2 have inputs: CN=1.

b. MR2 only has an input: CN=2.

c. MR1 only has an input: CN=4.

d. Both MR1 and MR2 have inputs: CN=8.

The global standard against which CN are measured is called the cancellation code (CC) and is transmitted to the array as part of the 3-D statement (its specific location in the statement will be shown later but is unessential at this point). The CC is an integer, ranging between 0 and 15. As shown in Table 3.1, each CC has a one-to-one correspondence with one of the sixteen possible functions of two logical variables. The logical variables in this instance are called MR1 and MR2, meaning input to MR1 and input to MR2, respectively. Thus a choice of CC is actually a choice of a logical function against which MR1 and MR2 inputs of individual memory cells are compared. If, in a given cell, the input connection causes the logical function to

be TRUE, execution is inhibited; if the input connection causes the logical function to be FALSE, execution proceeds.

The functions in Table 3.1 are arranged so that as one proceeds down the table (toward higher CC's), those functions whose CC's are not powers of two are the result of one or two logical OR operators applied to appropriate preceding functions whose CC's are powers of two. Such appropriate preceding functions are those whose CC's will sum to the CC in question. As a mnemonic device, the sum of the matching CN's is equal to the CC in each case that execution is inhibited.

Thus, any CC which is a power of two stands alone in that only one CN matches it (causes its associated logical function to be TRUE) while other CC's are matched by two or three CN's, those CN's being the unique set of CN's that sums to the given CC. As examples, CC = 5 is matched by CN = 1V4, and CC = 14 is matched by CN = 2V4V8, whereas CC = 8 is matched only by CN = 8. In every case of a match, execution in that cell will be inhibited. The resultant execute-inhibit pattern in the array is called a cancellation mask.

In the operational instruction set, a broader use of cancellation codes will be made in that for a given macroinstruction, more than one CC may appear, each associated with a particular segment of the overall macroinstruction.

Based upon limited experience, it can be said that use of

Table 3.1. Cancellation Codes

CC	Logic Symbol	Meaning	Matching CN
0	---	No cancellation	---
1	(∿MR1)Λ∿ MR2	Neither MR1 or MR2 have inputs	1
2	(∿MR1)ΛMR2	Only MR2 has an input	2
3	∿MR1	No input for MR1	1V2
4	MR1Λ∿MR2	Only MR1 has an input	4
5	∿MR2	No input for MR2	1V4
6	MR1 ≠ MR2	One of MR1, MR2 has an input	2V4
7	(∿MR1)V∿MR2	No input for MR1 or for MR2	1V2V4
8	MR1ΛMR2	Both MR1 and MR2 have inputs	8
9	MR1 - MR2	0 or 2 inputs for MR1, MR2	1V8
10	MR2	MR2 has an input	2V8
11	(∿MR1)VMR2	Either MR2 has input or MR1 doesn't, or both	1V2V8
12	MR1	MR1 has input	4V8
13	MR1V∿MR2	Either MR1 has input or MR2 doesn't, or both	1V4V8
14	MR1VMR2	Either MR1 or MR2 or both has an input	2V4V8
15	1	Cancel everything	1V2V4V8

appropriate combinations of transfer patterns and cancellation codes is an art form or Gestalt operation. Its ultimate value is yet unknown.

3.2.3 OPERATIONAL INSTRUCTIONS

3.2.3.1 Instructional Format

Operational instructions are of fixed format, consisting of the five fields in sixteen columns each as shown in Figure 3.5.

Field	1	2	3	4	5a	5b
Column	1 16	17 32	33 48	49 64	65 70	71 80
Desc.	Bound.	3-D Instr.	Source	Dest.	ALOC	Label

a. Operational Instruction Format

(01:03;01:04) TRAN(00,00,16) MR(04) CP FAD 555

b. A Typical Operational Instruction

Figure 3.5

Details of each field will be covered in the following sections. As a preliminary remark, it is seen that operational instructions have embedded within them a 3-D instruction, which defines only the

specific operation to be performed within the 3-D machine but does not specify any boundaries, source destination, etc. This distinction between operational instructions and 3-D Instructions will be carefully maintained in subsequent sections. 3-D instructions will be covered in a separate section.

It is worthwhile to note that any XDS Extended FORTRAN IV statements are also operational instructions, provided they have an 'F' in Column 1. Familiarity with FORTRAN is assumed and no further syntactical discussion will be made. References to the utility of FORTRAN in SIM3D programming will be made at appropriate points.

3.2.3.2 Field Descriptions

Field 1

Field 1 is a boundary specification field for the rows and columns on which the 3-D operation will take place (subject to further inhibition by CC and CN). An 'F' in the first column of Field 1 signifies a FORTRAN type statement (non-array) which is executed exclusively by the supervisory machine. This allows a mixed stream of 3-D and FORTRAN statements providing capabilities for looping, branching, indexing, counting, etc. A 'C' in the first column signifies a comment in the usual FORTRAN sense.

Example

```
        col 1
        ↓
(01:05;02:07)
```

This specifies a 5 x 6 array in which the instruction is effective on rows 1 to 5 inclusive and columns 2 to 7 inclusive.

Field 2

The 3-D instruction fits into Field 2. The instruction is appended by two or three cancellation codes. The first cancellation code (CC1) constructs a cancellation mask which applies to operation on the contents of the MR1 register. The second cancellation code (CC2) does the same regarding the MR2 register and the third cancellation code (CC3) will not only allow or inhibit the ALOC but also differentiates between a local ALOC for each cell and a global ALOC for all cells inside the specified boundaries. CC's are used only in the interplane transfer instruction (TRAN) when CP is the destination. Refer to the following section on 3-D instructions for complete details on CC3.

Example

```
  col 17
  ↓
INPT(00,15)
```

This is an input instruction which inputs data into the MR1 registers of the cells specified since CC1 = 0. This instruction inhibits input to the MR2 registers since CC2 = 15.

Example

```
col 17
↓
INPT
```

An input instruction is valid without a cancellation code when the input is to a control register (MRC of CR registers).

Field 3

This field specifies either a source of information or a destination for the information depending on the type of 3-D instruction. In input operations it signifies the destination in the 3-D machine for information coming from the input buffer stack. In output operations it signifies the location in the 3-D machine from which output will be transferred to the output-buffer stack. In transfer operations it signifies the data source.

Example

```
col 1          col 17         col 33
↓              ↓              ↓
(01:02;02:04)  TRAN(00,15)    MR(04)
```

This is a transfer instruction from the MR1 registers of memory plane 4. MR2 registers are unaffected since CC2 = 15.

Field 4

This field specifies the destination for data transfer.

Example

```
col 1          col 17         col 33         col 49
↓              ↓              ↓              ↓
(01:02;02:04)  TRAN(00,15)    MR(04)         MR(16)
```

This is a complete 3-D instruction for a transfer of the contents of the MR1 registers of plane 4 to the MR1 registers of plane 16 for rows 1 and 2 and columns 2 through 4 of a 2 x 3 array. MR2 registers are again unaffected.

Field 5

Field 5 is divided into two subfields:

A. Field-5a

This is reserved for the ALOC when the CP is operated in a ganged mode. This field occupies columns 65 through 70.

Example

```
↓ col 1          ↓ col 17          ↓col 33   col 49   ↓ col 65
(01:02;02:15)    TRANS(00,00,16)   MR(04)    CP       FAD
```

This transfers the contents of the MR1 registers and MR2 registers of memory plane 4 (within the specified boundaries) to the corresponding CPA and CPQ registers, respectively, of CP. A floating point addition is performed by all non-inhibited CP cells, and the results are stored in the CPA's.

B. Field 5b

This is reserved for labels. A label is a numeric figure of up to 10 decimal digits which can be stored anywhere in columns 71 to 80 of the card.

Example

```
↓col 1        ↓col 17          ↓col 33 ↓col 49   ↓col 65 ↓col 71
(01:02;02:04) TRANS(00,00,16) MR(04)   CP        FAD     555
```

The instruction is identical to the previous example except for having 555 appended as a label.

3.2.4 3-D INSTRUCTION SET

There are three basic types of instructions to fit the architecture of the 3-D machine:

a. Input/Output (I/O) instructions

b. Transfer instructions

c. Branching instructions

3.2.4.1 Input/Output (I/O Instructions)

INPT

INPT is an input instruction which causes two sequential operations:

a. Transfer of data by rows from the input-buffer-stack to the I/O plane (within the specified boundaries).

b. Transfer of the contents of the I/O plane to a memory plane (specified in field 3) within the same boundaries as above.

OUTP

OUTP is an output instruction which does exactly the reverse of INPT, i.e.:

a. Transfer of the contents within specified boundaries of the memory plane specified in field 3 to the I/O plane.

b. Transfer by rows of the contents of the I/O plane within the specified boundaries to the output-buffer-stack.

3.2.4.2 Transfer Instructions

CTRN

CTRN is an intraplane transfer. This instruction moves information within the specified boundaries of a given plane. Data transfer into a register can be inhibited (cancelled) by the cancellation mask. Output from a cell cannot be inhibited.

DTRN

DTRN is a direct (intracell) transfer. This instruction transfers the contents of the MR1 register of a cell to the MR2 register of the same cell or vice versa. Transfer direction depends on field 3 (source field) and field 4 (destination field). This transfer is also subject to a cancellation mask.

TRAN

TRAN is an interplane transfer instruction having four possible transfer modes: (1) from memory plane to memory plane, (2) from

connection register plane to memory plane control register, (3) from computation plane to memory plane, and (4) from memory plane to computation plane.

MP to MP transfer requires two cancellation codes, CC1 and CC2, referring to MR1 transfers and MR2 transfer, respectively.

When TRAN is executed as an interplane transfer instruction, MR1 registers transfer only to MR1 registers, and MR2 registers transfer only to MR2 registers.

CR to MRC transfers are made to establish new principal patterns in memory planes and are not subject to a cancellation code based on previously existing patterns. No CC's need be appended to TRAN upon such a transfer.

Transfers from the CP to a MP are performed in a similar manner with CPA (the accumulator) being transferred to MR1 and CPQ (the Q register) being transferred to MR2. The cancellation mask constructed from CC1 (for MR1) and CC2 (for MR2) is based on the principal pattern of the destination plane in the MC.

When a transfer to the CP occurs, a third cancellation code, CC3, is used and an arithmetic-logic operation code is transferred concurrently with data. This ALOC is the one appended to the memory plane principal patterns and, under proper conditions of CC3, governs the arithmetic-logic operations of individual cells in the CP.

Table 2.1 lists the set of currently operational ALOC's.

The last card in any program (not including data cards) is the STOP instruction which begins in column 17. The STOP instruction serves two purposes: it marks the division between program and data during the translation and file creation segment, and it signals the usual termination of execution during the execution segment.

Input data is of two types: numerical data and interconnection pattern configuration data. Numerical data has an input format of 8E10.3 and needs no explanation. Interconnection pattern data is read through a format of 4(I3,1X,I3,1X,A4,8X) and takes a little arranging. The sample field below shows a typical input.

```
   col 1       col 12
    ↓           ↓
760 667 FSB
```

This specifies that registers MR1 and MR2 are to have connections corresponding to 760 and 667, respectively, and that an operation of floating subtraction is to be performed when any transfer is made from this memory plane to the computation plane, provided that the transfer is made with cancellation code less than 16, i.e., the autonomous mode of computation plane operation is in effect. The next fields begin in columns 21, 41, and 61, respectively. Thus a complete 80 column record of interconnection patterns has four sets of specification of principal patterns and ALOC's.

Termination of the data set is indicated by a record ¢EOD. This differs from the standard !EOD only in the omission of the !

symbol (which would be incorrectly interpreted as a monitor control card), i.e., column one is left blank.

As before, CC1 and CC2 refer to transfer of MR1 and MR2, and their destinations are CPA and CPQ, respectively. Since each cell in memory has an individual ALOC, various cells in the CP may be instructed to perform different operations upon a MC to the CP transfer. Such operations in the CP occur when CC3 is in the range from 16 to 31, this indicates that the ALOC of the MP is to be ignored by the CP and that the ALOC in field 52 is to be used in ganged mode execution by the CP. Again execution is subject to a cancellation mask based on the MP source, in addition to the boundary specifications of field 1. The effective CC3 insofar as cancellation mask construction is concerned is computed as $CC3_{eff}$ = CC3 modulo 16. As examples, CC3 = 23 indicates a global CP operation under $CC3_{eff}$ = 7; CC3 = 16 indicates a global CP operation $CC3_{eff}$ = 0 (no cancellation).

3.2.4.3 Branching Instructions

BRNU

This instruction causes an unconditional branch to the operational instruction whose label appears in field 3 of the BRNU instruction. Note that this instruction makes a new use of field 3.

BRNZ

BRNZ causes a branch if the contents of all MR1 and MR2

registers of the last memory plane referenced are zero. By selecting appropriate values of CC1 and CC2 desired MR1 and MR2 registers, respectively, can be masked out of the comparison process.

BRNF

BRNF causes a branch if the contents of all MR1 and MR2 registers of the last memory plane referenced are less than a constant. This constant appears in field 4 of the operational instructions. The effect of CC1 and CC2 is the same as in BRNZ.

STOP

STOP suspends 3-D machine operation without affecting register contents and puts the control unit of the 3-D machine in the first phase of its fetch cycle. It then suspends operations of the supervisory machine.

3.3 The Simulator (SIM3DMIX)

A simulator (SIM3DMIX) for the 3-D machine is available for use. Appendix C is a listing of SIM3DMIX. This simulator was written in FORTRAN for batch mode execution on the Sigma 7 computer and can be executed locally through card input or from remote terminals. FORTRAN has been selected over the assembly language of the local computer (METASYMBOL for the Sigma 7) to make it possible to use the simulator on other computers. Specifically, the Extended XDS FORTRAN IV (52) has been used. This version of FORTRAN has some features, not available

in standard FORTRAN IV, which allowed efficient data manipulation down to the bit level. This is realized by using the in-line assembly feature of XDS FORTRAN, in which assembly language statements can be included in the FORTRAN program (52, 54). In-line assembly language statements are identified by an S in the first column of the card (record in general). This in-line assembly makes it possible to do bit and byte manipulation very effectively without the need to call lengthy FORTRAN subroutines. This capability not only save time in writing these FORTRAN subroutines, but also made it possible to have a reasonable size simulator program (25-40 K words). The 25K words option can run on most general purpose (GP) computers without overlay.

A 3-D machine program consists of a mixture of operational instructions and FORTRAN statements. The operational instructions control execution within the array while the FORTRAN instructions give program structure through DO loops, FOR and WHILE statements, arithmetic and logical IF statements, indexing, counting, etc. As an example, the set of statements

```
F       DO 15   I = 1, 7
        (Group of operational instructions)
F 15    CONTINUE
```

will cause seven executions of the operational instruction group.

FORTRAN statements in SIM3DMIX are identified by the character 'F' in column 1 and they must have this character. Failure to insert

the 'F' will result in an error message. This restricts FORTRAN labels to columns 2 through 5 but is not expected to pose any problem. Statements 1 through 9999 should be sufficient. Also, since SIM3DMIX eventually produces a program entirely in FORTRAN, labels cannot be duplicated on FORTRAN statements and operational instructions. Any label can be used only once in a program. This allows an operational instruction to be treated the same as a FORTRAN statement in that it has a distinct label and can be the object of a branch or the terminal statement of a DO loop, etc.

Comments can be inserted in the program through the usual method of placing a 'C' in column one. They will appear unchanged in the translated program.

3.4 Program Organization and System Considerations

A general 3-D machine has been described so far, with an ixj array of cells in the MP with k planes in the MC. As far as a program to implement that machine simulation is concerned, some decision on i, j, and k had to be taken.

SIM3DMIX as presented in Appendix C and as has been run on the Sigma 7 computer, is an interpreter simulator for a 3-D machine made up of:

1. A MC which is a stack of 16 MP's, each MP is a 16x16 array of cells. Each cell contains MR1 and MR2 registers; each

if a full word register (32 bits). An MRC register for each cell stores a metapattern instead of connection gates settings. A metapattern is the result of applying a special subroutine (SUBROUTINE PACK of Appendix C) to the connection codes. SUBROUTINE PACK finds the source register for the particular register under consideration if any exists. The stored information in MRC as implemented in the simulation (this is different from MRC of the 3-D machine itself due to the difference between gate level design and register level simulation), consists of 8 bytes packed as follows:

byte 1: is the source row address for MR1.

byte 2: is the source column address for MR1.

byte 3: is the source row address for MR2.

byte 4: is the source column address for MR2.

byte 5: is the CN.

byte 6: is the most significant part of the ALOC.

byte 7: is the least significant part of the ALOC.

byte 8: is unused.

An MRT register for each cell is provided as a temporary storage location.

2. An IOP capable of I/O operations according to the simplest scheme described in Chapter 2.3. This IOP has a 16x16 array of registers similar to the MRT registers in the MC.

3. A CP of a 16x16 array of computer cells. Each cell has two registers CPA and CPQ in addition to the transfer register CPB. The ALO's go according to the CP description in Chapter 2.
4. A CR section as a stack of 10 control register planes. Each CR plane is a 16x16 array of control words identical to the MRC words.

Typical runs of the program indicate the need for between 35-40 K words of core memory in addition to the requirement of writing some scratch disk files in the SIM3D-to-FORTRAN translation stage of compilation.

By reducing the size of the MC to 8 planes of 8x8 cells each and using 8 CR planes, the core memory required to run SIM3DMIX can be reduced to 22-27 K words.

SIM3DMIX does not simulate the gate level of the 3-D machine; nevertheless, it gives an indication of the feasibility of building the 3-D machine. The register transfer level which is the theme of this thesis is very well implemented in SIM3DMIX.

3.5 Use of the Simulator

The 3-D simulator operates in a two-pass sequence: (1) Mixed operational instructions and FORTRAN statements are translated to an equivalent FORTRAN program; and (2) the equivalent program is

executed. The translation is accomplished through calling a load module names SIM3D and inputting the 3-D program complete with data. This input can be in the form of cards, teletype records, or a pre-constructed file stored on disk or tape. No matter what the 3-D program source, SIM3D will create three additional files: one of the original 3-D program (complete with data), one of the translated program and one of the data section of the original 3-D program.

The original program file is created solely as a convenience to the user. After SIM3D has produced the translated program and associated data files, it is not necessary that they be immediately executed. The user will have a record of the original program and data set as this original file is the first printed output that occurs upon execution of the translated program.

The translated program is a set of FORTRAN statements of the original program. (This package of FORTRAN subroutines exists in a separate file called SUBP whose proper references will appear later). In creating this set of subroutine calls, SIM3D extracts the instruction type, region of activity, source and destination, cancellation code, instruction label and ALOC from each operational instruction, and constructs one FORTRAN call per operational instruction. At execution time, the subroutine will perform the actual simulation.

The data section of the original 3-D program is stripped off

by SIM3D and put unchanged into a separate file to be used as data input at execution time.

Within SIM3D, the DCB assignments are: F:110 for the original file; F:109 for the translated program file; and F:115 for the data file. In the sample set of control and program statements, these assignments will be shown.

3.6 Typical Program Arrangements

Commands for executing the program from remote terminal differ somewhat from those required for batch mode execution, and therefore two sets of records for program execution are shown in Tables 3.2 and 3.3. Each command set is in turn divided into two subsets, one required for translation and the other for execution. For terminal execution, it is assumed that a 3-D program has been created under EDIT or by similar means and that it is in a file named SIMS. It is also assumed that output is desired on the line printer. If local teletype output is desired, omit the !SET F:108 LP command in terminal execution.

Table 3.2 Execution Via Card Input

	!JOB XXX,NAME,PRIORITY.
	!LIMIT (TIME,10), (UO,200), (LO,200),(CORE,40)
	!ASSIGN F:109,(FILE, SIMLOAD), (OUT),(SAVE)
Translation Segment	!ASSIGN F:110, (FILE, SIMPROG),(OUT),(SAVE)
	!ASSIGN F:115,(FILE,DATA),(OUT),(SAVE)
	!RUN (LMN,SIM3D,667)
	(The 3-D Program)
	!ASSIGN M:SI,(FILE,SIMLOAD),(IN)
	!ASSIGN F:110,(FILE,SIMPROG),(IN)
Execution Segment	!ASSIGN F:105,(FILE,DATA),(IN)
	!FORTRAN LS,GO,S,BC,SI
	!LOAD (GO) (EF,(SUBP,667),(EXEC)

Table 3.3 Execution Via Terminal

	LOGON, XXX,NAME,PASSWORD
	!SET F:108 LP
	!SET F:109 FILE/SIMLOAD;OUT;SAVE
	!SET F:110 FILE/SIMPROG;OUT;SAVE
Translation Segment	!SET F:115 FILE/DATA;OUT;SAVE
	!SET F:105 FILE/SIMS;IN
	!SIM3D 667
	(the machine will reply with *EXIT*. This may take some time.)
	!SET M:SI FILE/SIMLOAD;IN
	!SET F:110 FILE/SIMPROG;IN
	!SET F:105 FILE/DATA;IN
Execution Segment	!FORT SIMLOAD
	OPTIONS > LS,GO,S,BC
	!RUN $, SUBP
	(again the machine will reply with *EXIT* and may take some time to do so.)
	!OFF

CHAPTER 4

APPLICATION OF THE 3-D COMPUTER TO MATRIX ORIENTED PROBLEMS

4.1 General Remarks

Although there is no accepted or standard means to measure parallel processor performance, theoretical estimates of execution time of certain matrix oriented problems are indicative of parallel processor performance. In this sense, the 3-D machine is more effective than its predecessors in executing matrix oriented problems if the different planes of the machine are operating in an overlapping fashion, even with essentially unchanged algorithms. The three fundamental matrix operations of transposition, multiplication, and inversion will be described. For these operations on nxn arrays using non-overlapping algorithms, execution time of the 3-D machine is proportional to n compared to n^3 in the case of an equivalent serial machine. A factor greater than one to one improvement in execution time over the Mulder machine results from separating the I/O, data transfer, and computation. For a problem which uses x% of the execution time doing I/O, y% doing data transfer, and z% doing computation, the 3-D machine execution time could be as low as the maximum of (x, y, z)% compared to a 100% for the Mulder machine. In the following sections some parallel algorithms are described. Listings of programs to implement them and execution summaries are in the appendices.

4.2. Matrix Transpose Algorithm (MTA)

The MTA assumes that the matrix is stored in a set of MR1

registers in a natural order, i.e. an nxm matrix is stored in a set of nxm contiguous cells. The matrix described herein is assumed to be an nxn matrix, although the MTA algorithm is generally applicable to non-square matrices. A graphical representation describes the transfers which take place during the transposition process. See Figures 4.1, 4.2, 4.3, 4.4, 4.5. In this representation an 'X' in a cell means the element in that cell is in the proper location and is to be undisturbed; a 'D' means that the element is to interchange position with the symetrically located element on the minor diagonal through that minor diagonal's bus (i.e. an i, j element interchanges position with the j, i element); and 'O' means that an i, j element is transposed through a horizontal then vertical (or vertical then horizontal) bussing where the direction change from one bus to the other is through the first 'X' encountered.

4.2.1 The Algorithm

This algorithm describes how to write intraplane connection patterns to accomplish a transposition. The number of connection patterns required to transpose an nxn matrix is equal to $\lceil (n-1)/3 \rceil$, where the symbols $\lceil\ \rceil$ signify the integer ceiling function, defined as the smallest integer greater than the floating point number enclosed.

The algorithm requires $\lceil (n-1)/3 \rceil$ cycles of intraplane trans-

fers compared to $n^2 - n$ cycles in the equivalent serial machine (this figure does not include the serial machines bookkeeping time for array indexing). The MTA proceeds as follows:

step 0: $k \leftarrow 0$

step 1: For each row, interchange the two elements to the right of the last X in that row in the D sense. The third element is interchanged in the 0 sense.

step 2: $k \leftarrow k + 1$; if $k < \lceil (n-1)/3 \rceil$ go to step 1; otherwise done.

Figures 4.1 to 4.5 show the schematic of the successive cycles of the MTA. Five CR words/element are essential to have a prestored MTA sufficient to transpose matrices up to 16x16.

Appendix D is a listing of the MTA for a 4x4 matrix. A general program for an nxn matrix is obvious.

4.3 Matrix Multiplication Algorithm (MMA)

The MMA proposed is a version of the MMA presented by Rudberg in 1968 (40). It requires a number of execution cycles proportional to n for an mxn matrix A multiplied by an nxp matrix B. This in contrast to a requirement proportional to n^3 cycles for the equivalent serial machine.

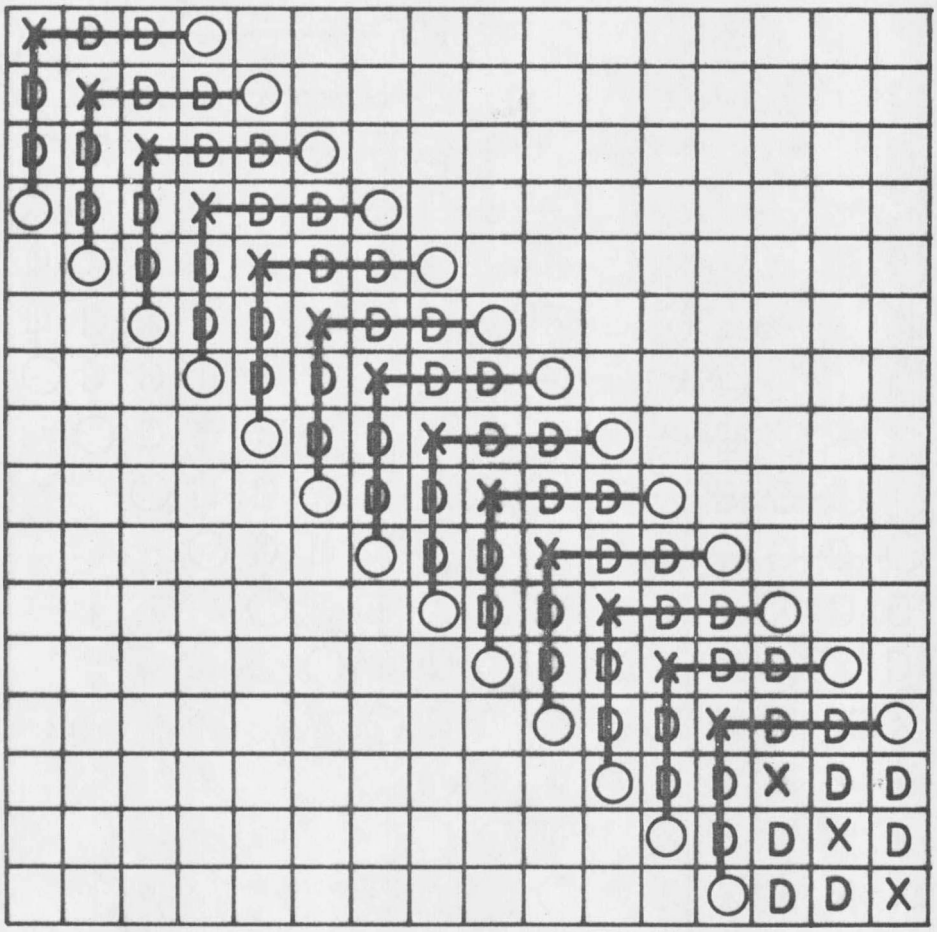


Figure 4.1 First Step of Matrix Transpose Data Flow.

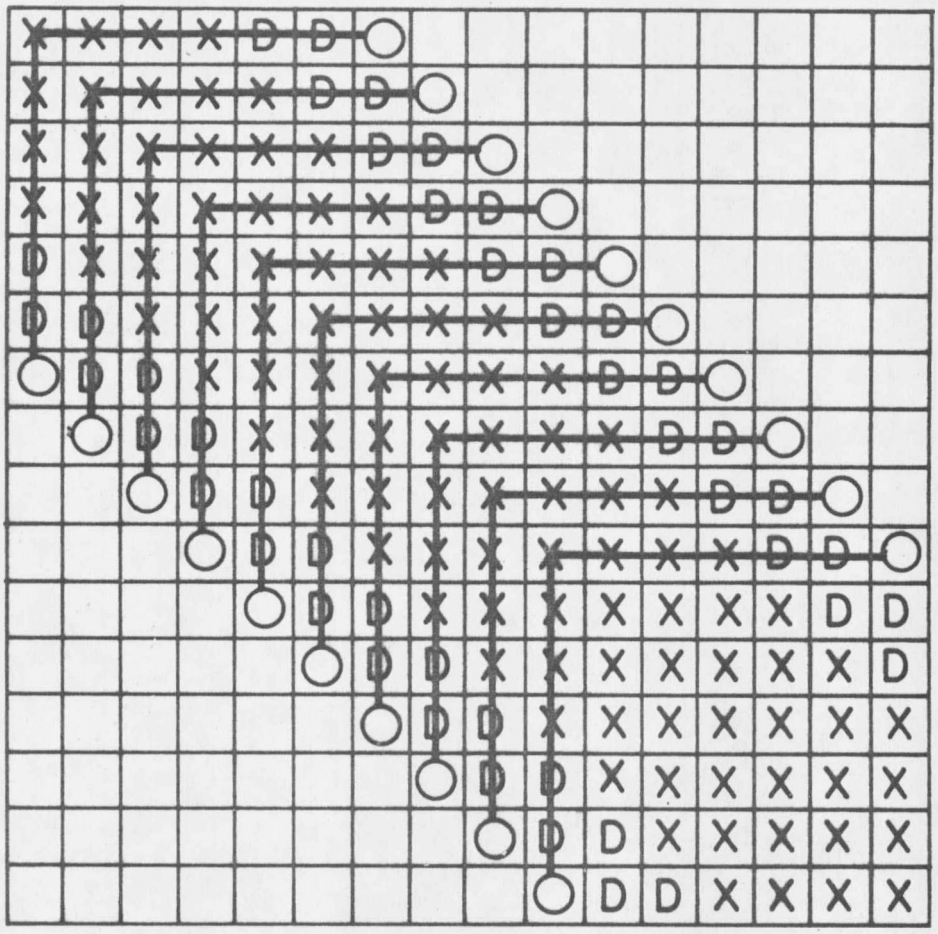


Figure 4.2 Second Step of Matrix Transpose Data Flow.

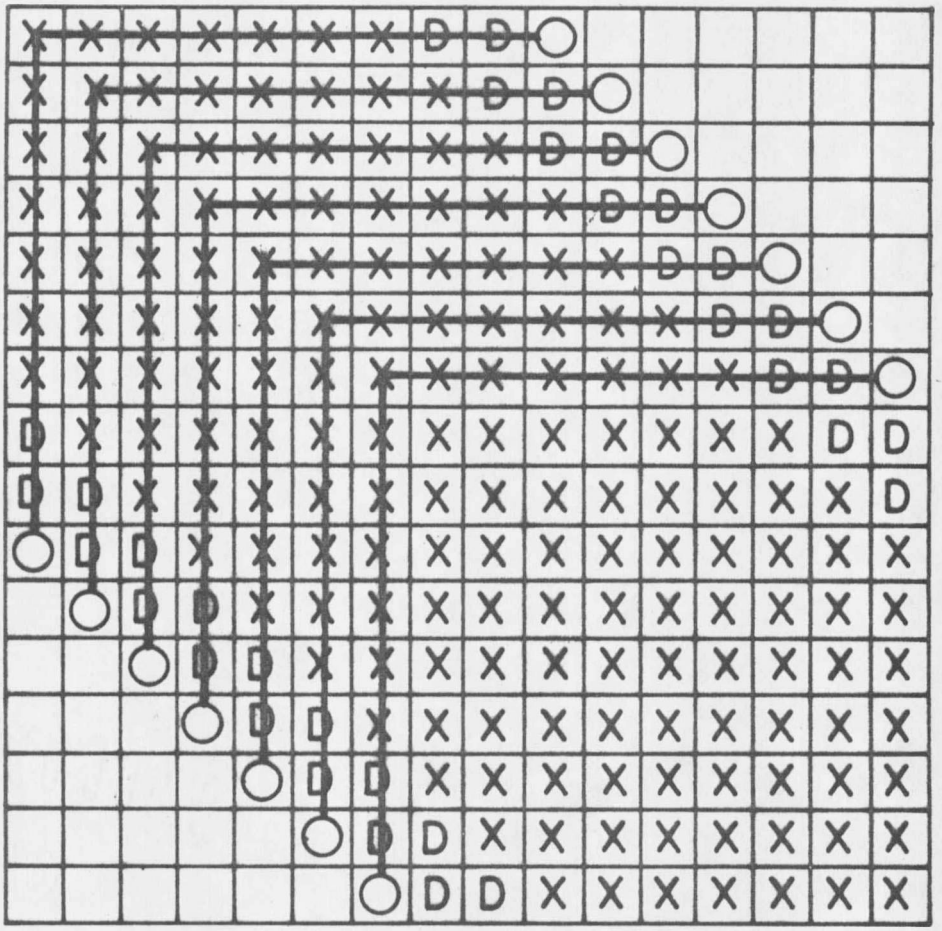


Figure 4.3 Third Step of Matrix Transpose Data Flow.

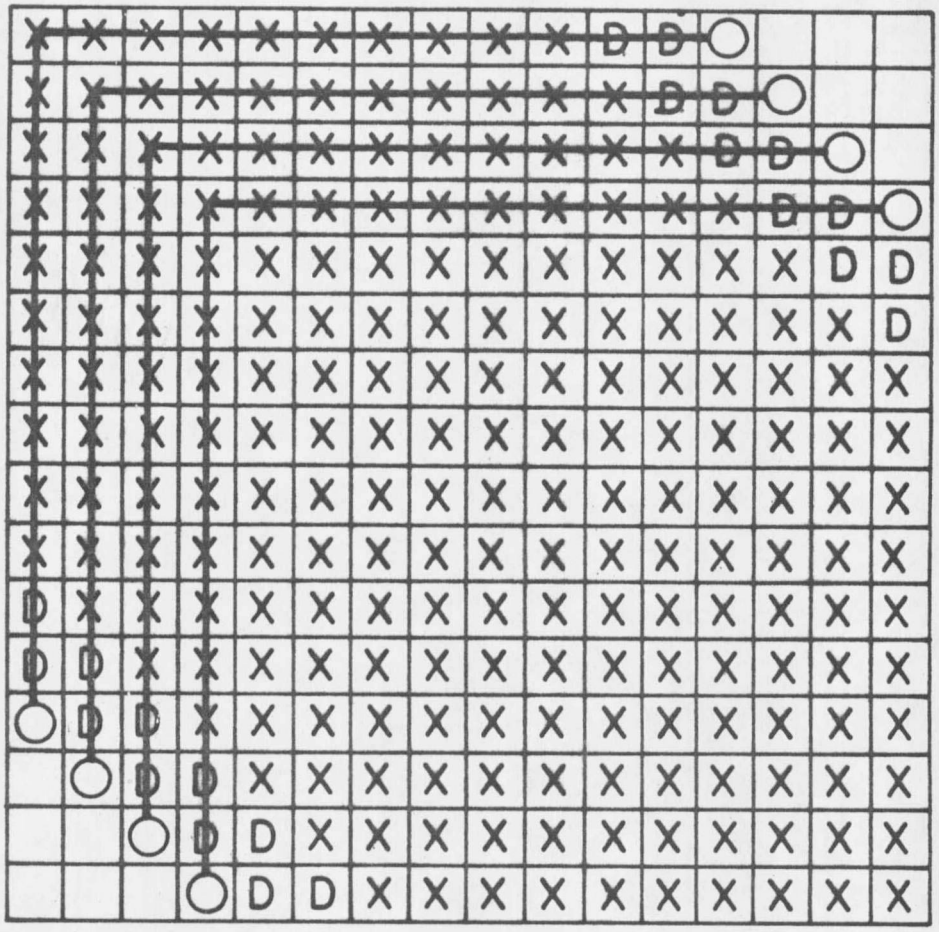


Figure 4.4 Fourth Step of Matrix Transpose Data Flow.

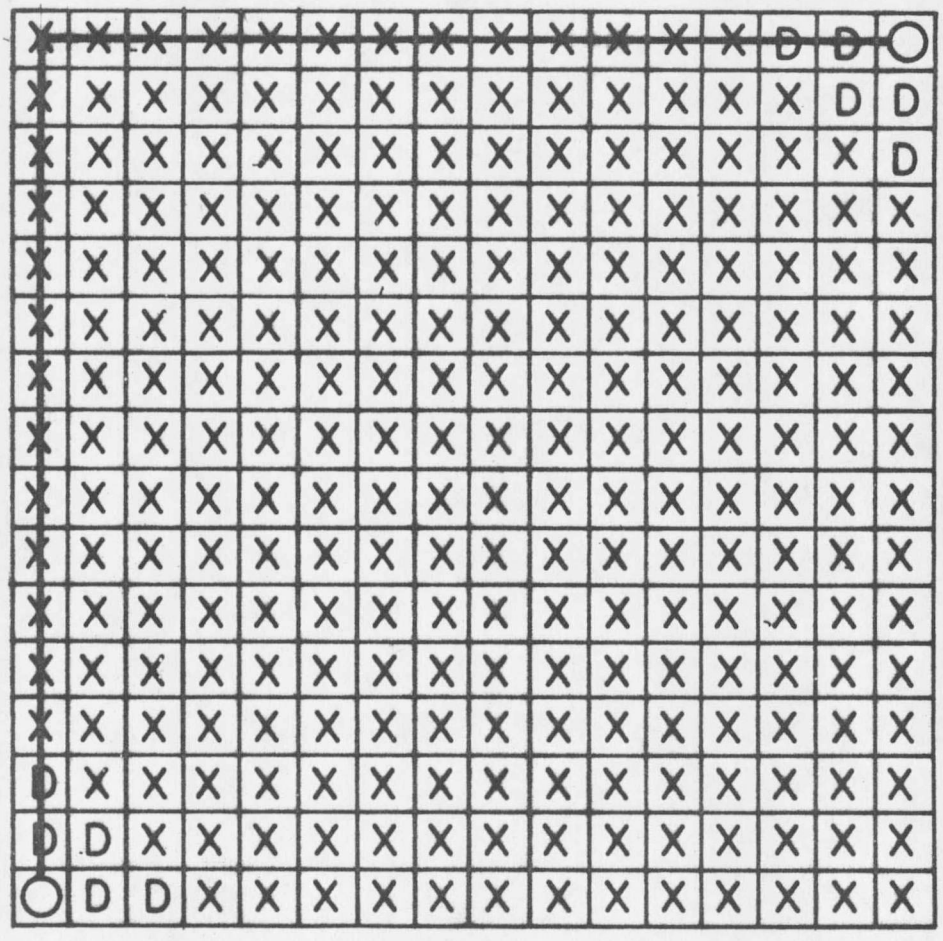


Figure 4.5 Fifth Step of Matrix Transpose Data Flow.

4.3.1 The Algorithm

The MMA assumes that matrix A was previously stored in the MR1 registers of a given MP in natural order and that matrix B is stored in the MR2 registers of the same plane in natural order. The algorithm is as follows:

step 0: k←1.

step 1: Transfer the first column of A to an unassigned MP to form a column submatrix and the first row of B to the same MP to form a row submatrix.

step 2: Transfer the column submatrix of A found in step 1 of the MP and likewise the row submatrix of B to n rows of that same MP as shown in Figure 4.6.

step 3: Transfer the MP of step 2 to the CP and perform a ganged multiplication operation. Add the result of multiplication in each cell to a register in each cell of the CP which are treated as accumulators.

step 4: Shift matrix A one column to the left and matrix B one row up as shown in Figure 4.7.

step 5: k←k+1; if k < n go to step 1; otherwise done.

The contents of the accumulators in the CP are the product C of A and B.

Appendix E is a listing of a program to implement the MMA on SIM3DMIX. A summary of results is included.

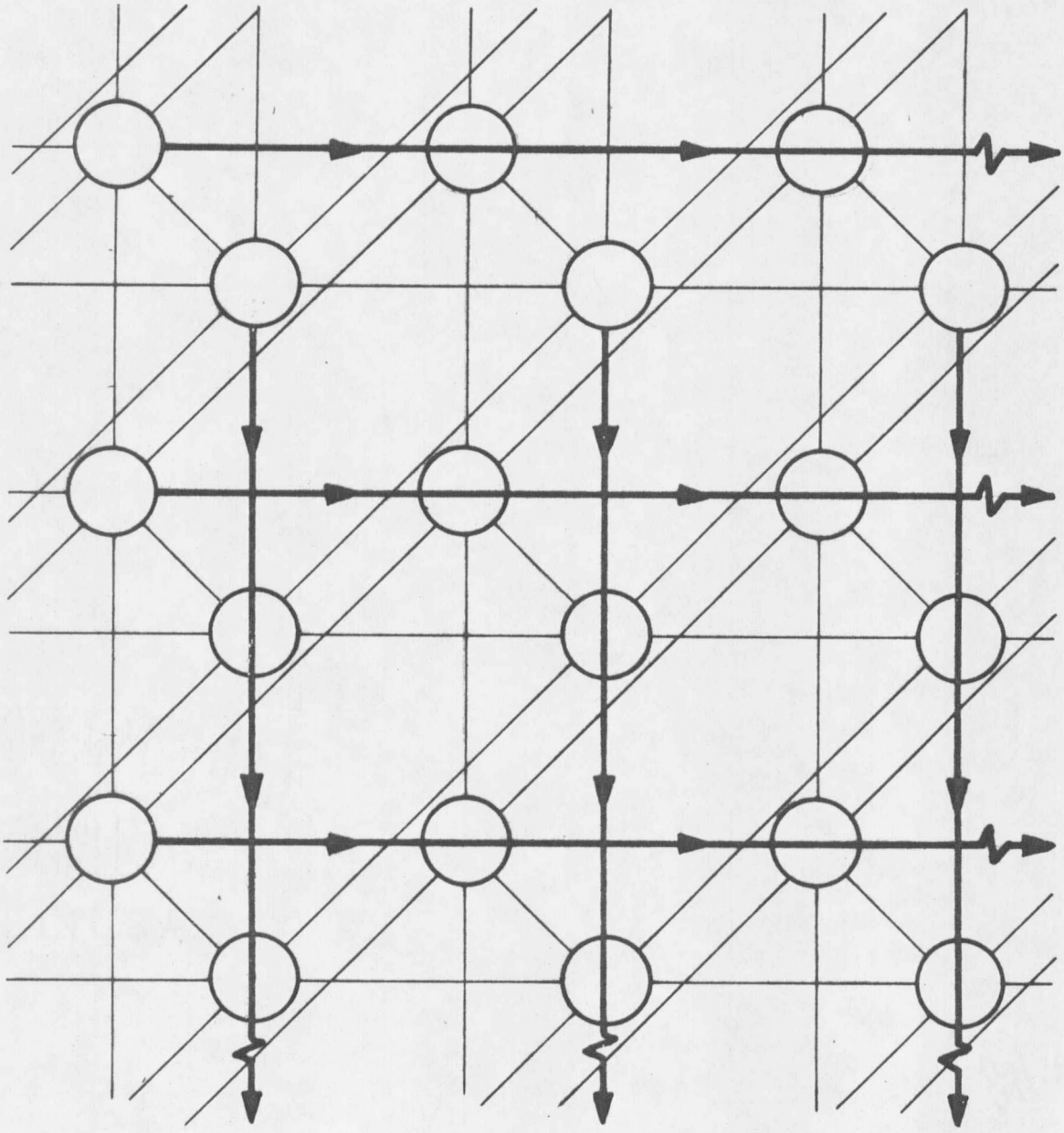


Figure 4.6 Step 3 of MMA

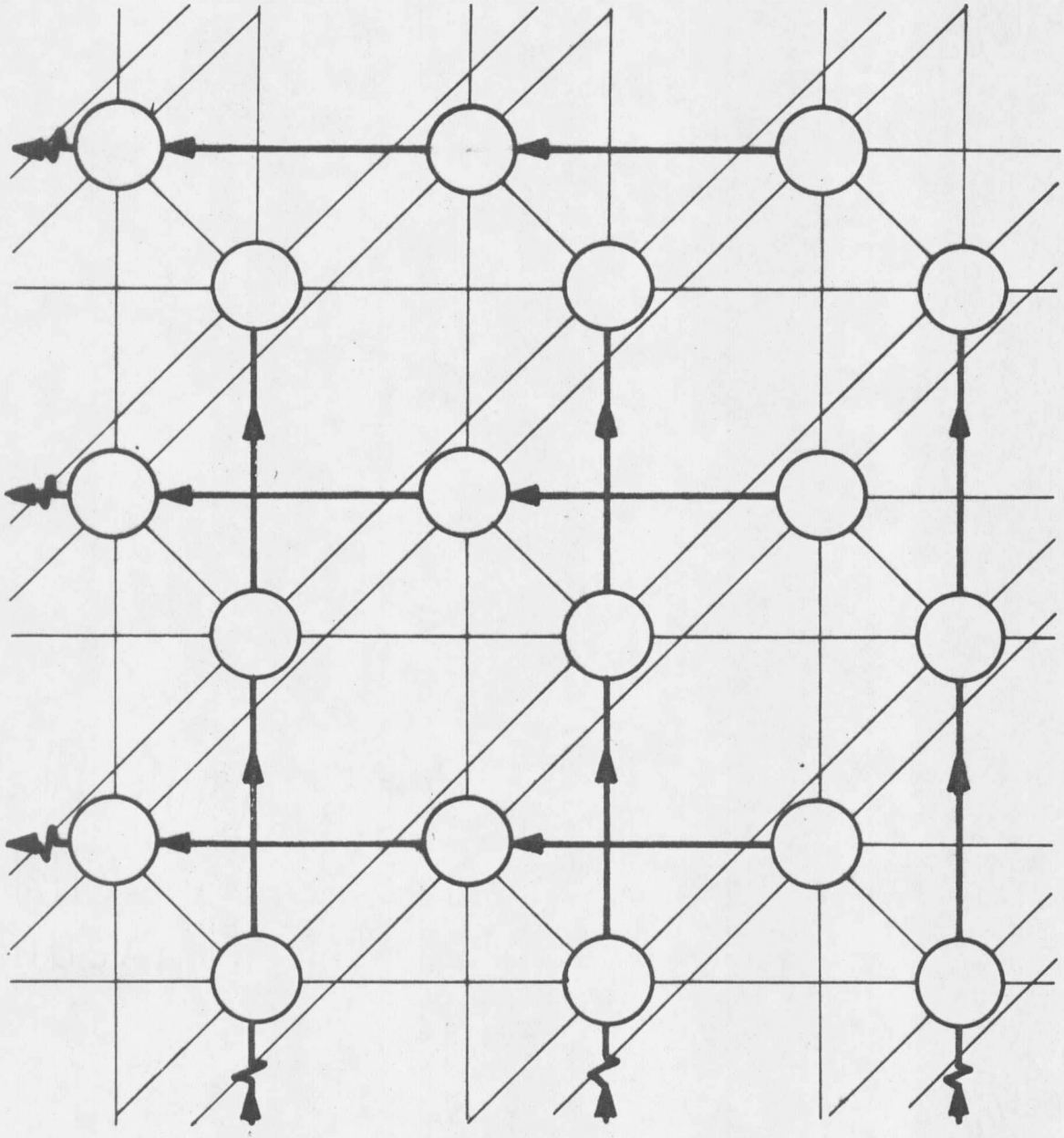


Figure 4.7 Step 4 of MMA

4.4 Matrix Inversion Algorithm (MIA)

A parallel version of the Gauss-Jordan matrix inversion algorithm (16) has been written and tested. It is recognized as being far from the most effective use of 3-D machine resources for such an operation. The number of execution cycles is proportional to n in the case of an nxn matrix, contrasting with an n^3 figure for the equivalent serial machine. The MIA can also find the solution of a set of simultaneous linear equations with no execution time penalties.

4.4.1 The Algorithm

For the solution of n simultaneous linear equations and the associated problem of finding the inverse of their coefficient matrix, assume that an nx(2n+1) array of cells of the 3-D machine is available on each of three MP's. Assume that the set of equations is stored in the first nx(n+1) portion of the array and the identity matrix I in the second nxn portion, i.e., the arrays should be organized in the MP as follows:

$$
\begin{array}{cccccccc}
a_{11} & a_{12} & \cdots\cdots\cdots & a_{1n} & x_1 & 1 & 0 & \cdots\cdots 0 \\
a_{21} & a_{22} & \cdots\cdots\cdots & a_{2n} & x_2 & 0 & 1 & \cdots\cdots 0 \\
\vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
a_{n1} & a_{n2} & & a_{nn} & x_n & 0 & 0 & \cdots\cdots 1
\end{array}
$$

These elements are initially stored in the MR1 registers of MP1. The case of a zero pivot (i.e. $a_{11} = 0$) is recognized and handled by interchanging two rows and two columns of the array as described in the algorithm. The algorithm is as follows:

step 0: $k \leftarrow 0$

step 1: Transfer MP1 to MP2.

step 2: If a_{11} of MP2 is zero interchange the first two rows and interchange column n+2 and n+3.

step 3: Transfer a_{11} of MP2 to all MR2 registers of the first row.

step 4: In MP2 transfer all MR2 registers of the first row to all MR2 registers of all rows of the array.

step 5: Divide the contents of the MR1 registers by the contents of their corresponding MR2 registers and return the result to another MP, e.g. MP3.

step 6: For MP3 propagate the contents of the MR1 registers of all rows except for the first row to the MR2 registers of the same row. Multiply the contents of all cells except for the cells of the first row.

step 7: Transfer the contents of all MR1 registers of MP3 directly to the MR2 registers of the same cells; subtract the MR2 registers of MP3 from the contents of the MR1 registers of MP2 and store the results

in MP2.

step 8: In MP2, interchange the MR1 registers of the first column with column k + 2 and interchange the MR1 registers of the first row with row k + 2.

step 9: k ← k + 1; if k < n go to step 2; otherwise done.

The MR1 registers of columns 1,2,..,n of MP2 contains the I matrix, column n+1 of MP2 contains the solutions for the set of equations, and columns n+2, n+3,....,2n+1 contains the matrix inverse.

Appendix F is a listing of a program to implement the MIA on SIM3DMIX, a summary of the results is included.

4.5 Search for a Maximum or a Minimum Element of an Array, Algorithm (MMESA)

The algorithm described here is organized in such a way as to give a search time independent of the size of the array under search. This algorithm is written to be directly implemented in hardware. The motive behind this algorithm is the rapid solution of minimization algorithms (37). A microprogrammed search processor is directly attached to the CP of the 3-D machine which can be initiated directly by the supervisory machine to find the maximum or the minimum of an array which has been already transferred to the CP. The algorithm requires that the CPA registers can function as left

circular shift registers and that a flag (an R-S flip-flop) exists for the condition of each CPA register.

Data should be represented in a 2's complement floating point form for the algorithm. The algorithm can be modified to accomodate other forms of data representation. A mask matrix which is part of the cancellation structure of the 3-D machine is assumed (previously described in Section 3.1). Any element in the array which is not to be included in the search is declared so by storing a 0 in the location in the mask matrix corresponding to that element. SHIFT (k,I) indicates that bit location k of word I is loaded in the most significant location in the shift register to be processed (on the left end). FLAG(I) is the flag corresponding to word I of the array. An nxm array is transferred to a linear one by using the index relation:

$$I = j + (i-1) \times m$$

4.5.1 The Algorithm

step 0: If for some I, SHIFT (1,I) = 0; THEN for all I, MASK (I) $\leftarrow$ ($\overline{\text{SHIFT(1,I)}}$ Ω MASK(I).

step 1: k $\leftarrow$ 2

step 2: FLAG(I) $\leftarrow$ (SHIFT(K,I) Ω MASK(I)

step 3: IF for all I, FLAG(I) = 0; THEN for all I, FLAG(I) $\leftarrow$ MASK(I), k$\leftarrow$k+1, go to step 1.

step 4: IF Σ FLAG(I) = 1 go to step 5; otherwise
all I

MASK(I) $\leftarrow$ FLAG(I) Ω MASK(I); k$\leftarrow$k+1

step 5: IF k=32 there is more than one element with value equal to the maximum, take the one with the least index I as the maximum, go to step 5; otherwise go to step 1.

step 6: Done with I the index of the maximum element.

The maximum of the array is contained in CPA (k,j) where I = j + (k-1). By initially applying the complement operation on the array, the algorithm can find the minimum element of an array. SUBROUTINE SRCH of Appendix C is a FORTRAN implementation of the MMESA.

CHAPTER 5

FAULT-TOLERANT DESIGN AND OPERATION OF THE 3-D COMPUTER

5.1 Introduction

A fault-tolerant design is a very involved problem, manifesting itself in both the hardware and software design of the 3-D machine, reaching from the logic gates through the operating system. Since the objective of this thesis is to describe the register-transfer-level design, with partial gate level design for only nonconventional parts like the memory cube, this Chapter on fault-tolerant design and operation should be considered more of a guideline for future research rather than a final decision. In what follows quantitive analysis is avoided which can be found in References 1, 3, 10, 11, 19, 21, 28, 36, and 46.

5.2 The Necessity of Redundancy

Fault-tolerant computing can be defined as the process of producing reliable results using non-reliable circuits or non-reliable data. This is in contrast to information theory which assumes the use of reliable circuits on non-reliable input information. Fault-tolerant computing is essential for the 3-D machine operation for the following reasons:

1. It is desirable if graceful degradation of performance is attained. Graceful degradation means the capability of the machine to keep executing the job at lower and lower speeds as more and more hardware fails until this failing

hardware is replaced or the problem has been solved. This graceful degradation is only possible through hardware and software redundancy. This is in contrast to the inelastic or catastrophic failure limits exhibited by other modes of fault-tolerant operation (1, 3, 27, 35).

2. The 3-D machine could be located in places that preclude manual repair when needed.
3. As a real-time control processor the 3-D machine has ample hardware complexity which makes the machine liable to fail more often than a simplistic machine. One way to increase the mean time between failures is through redundancy.
4. The penalties for fault incurred in certain machine applications can be very serious.

5.3 Remarks on Fault-Tolerant Techniques

In the early days of computers highly reliable components were emphasized to improve system reliability. It is an approach which may continue for years (36) but is not a sole alternative.

Hardware redundancy was introduced by Von Neuman (46) in 1956. Hardware redundancy simply means the use of extra parts so systems can operate correctly with some parts failing before system operation starts to degrade. Very few of the hardware redundancy techniques are such that the degradation is graceful.

Software redundancy may be included whenever system timing permits (the main concenr here is about real time applications). Software redundancy includes multiple solutions and roll-back methods (11).

5.4 Hardware Redundancy Methods

At different levels of the system, different redundancy methods are used; of these the following two methods are described:

1. Parity checking and information coding.
2. Voting and decision devices.

5.4.1 Parity Checking and Information Coding

Error correction can be accomodated by appending an appropriate number of error correcting bits (21) to the data word. As an example a 32-bit data word requires six bits for a single-error-correcting Hamming code (21). In general, the relation between the number of information bits n and the number of correction bits k for single error-correcting code is (21)

$$2^n \geq n+k+1$$

Table 5.1 presents the smallest number of correcting bits for several values of n. Multiple error detecting and/or correcting codes are discussed in the literature on coding (20, 21, 48).

Table 5.1 Number of Correcting Bits Required for Different Values of Word Size.

Information Bits n	Extra Bits k
1	2
2	3
4	3
8	4
16	5
32	6
64	7

5.4.2 Voting and Decision Devices

Voting and decision devices are devices which receive three or more inputs and produce a single output which agrees with the majority of these inputs in the case of majority voting. If any other predetermined criteria for voting is used, a weighted vote taking process takes place which produces a weighted output. The general case of vote taking produces an output of the form

$$U\ U = (w_1 \times x_1 + w_2 \times x_2 \ldots\ldots + w_n \times x_n)/n = \frac{1}{n} \sum_{i=1}^{n} W_i X_i$$

where:

U is the output.

x_i is the i^{th} input.

w_i is the weighting factor for input i.

n is the number of inputs.

Assuming that components have the same error probability, a majority vote (i.e. $w_1 = w_2 \ldots = w_n = 1$) is the best to use.

The CP is the subsystem where the vote taking is needed the most. Triple Module Redundancy (TMR) (36) can be utilized unless more reliability is desired in which case N Module Redundance (NMR) (36) can be used. Reference 36 contains the mathematical analysis which provides the probability of failure of a system in the general NMR case.

Assuming that the vote taker is a highly reliable device, a TMR operational subunit of the CP or any of its processors should be

as shown in Figure 5.1. Here A1, A2, A3 are different replica of the same subunit and B1, B2, and B3 are different replica of the output of these subunits to be processed through the next stage of the CP's unit.

A more complicated TMR subunit (which assumes that the vote taker is making errors) should be as in Figure 5.2. A NMR system is shown in Figure 5.3. A NMR system having N vote takers is shown in Figure 5.4.

A general redundant system with spare units has been described in reference 36 with all the mathematical analysis. It is called a hybrid redundant system and is denoted by HMR(N,I) where:

HMR stands for a Hybrid Module Redundancy

N is the number of active modules

I is the total number of modules including spare units.

An HMR (3,5) is shown in Figure 5.5.

5.5 Software Redundancy Methods

A technique which has been used with general purpose computers and shows potentials for being used with the 3-D machine is the Roll-back technique (11). The idea behind roll-back as it is intended to be applied to the 3-D machine is to partition any program into different parts. Each partition of the program is terminated by saving a set of parameters through an I/O process to the secondary storage system.

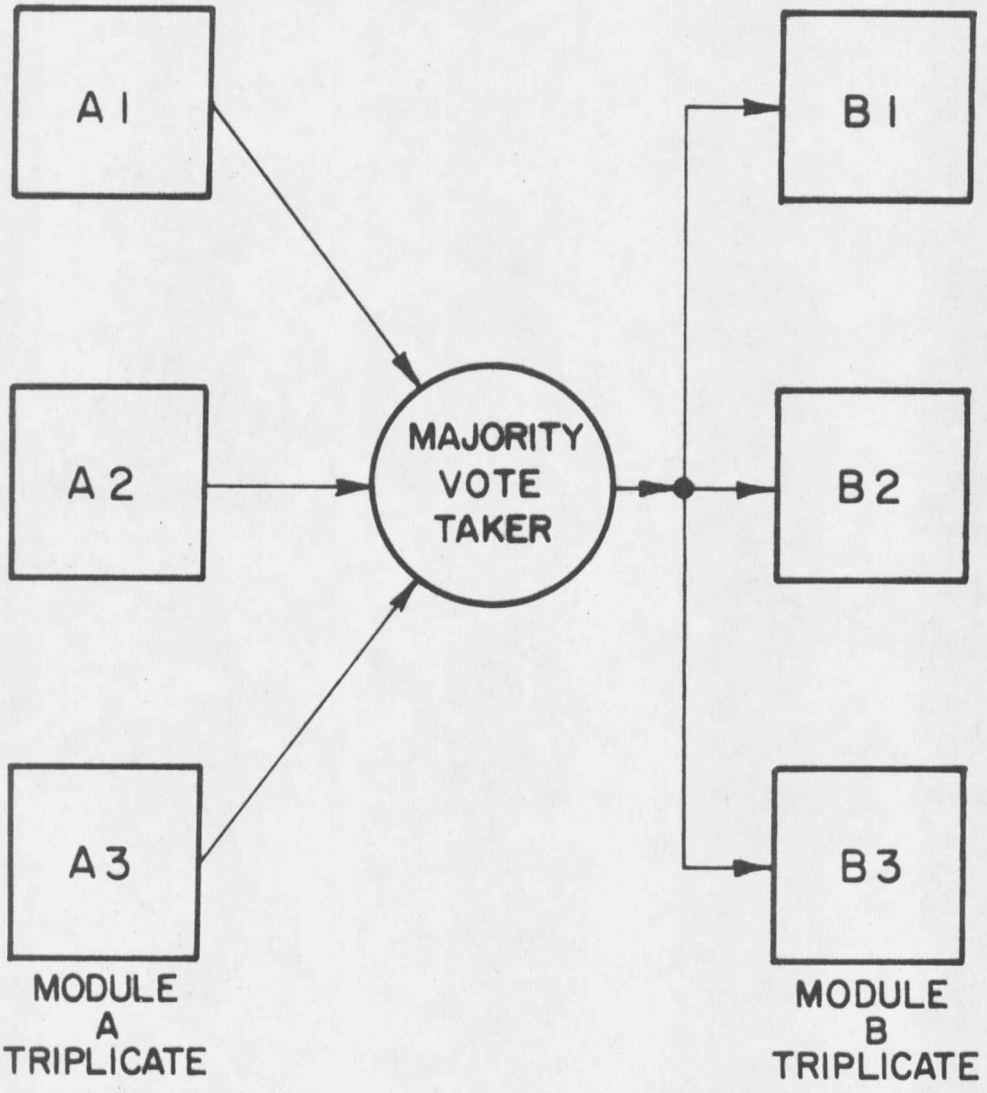


Figure 5.1 TMR Subunit

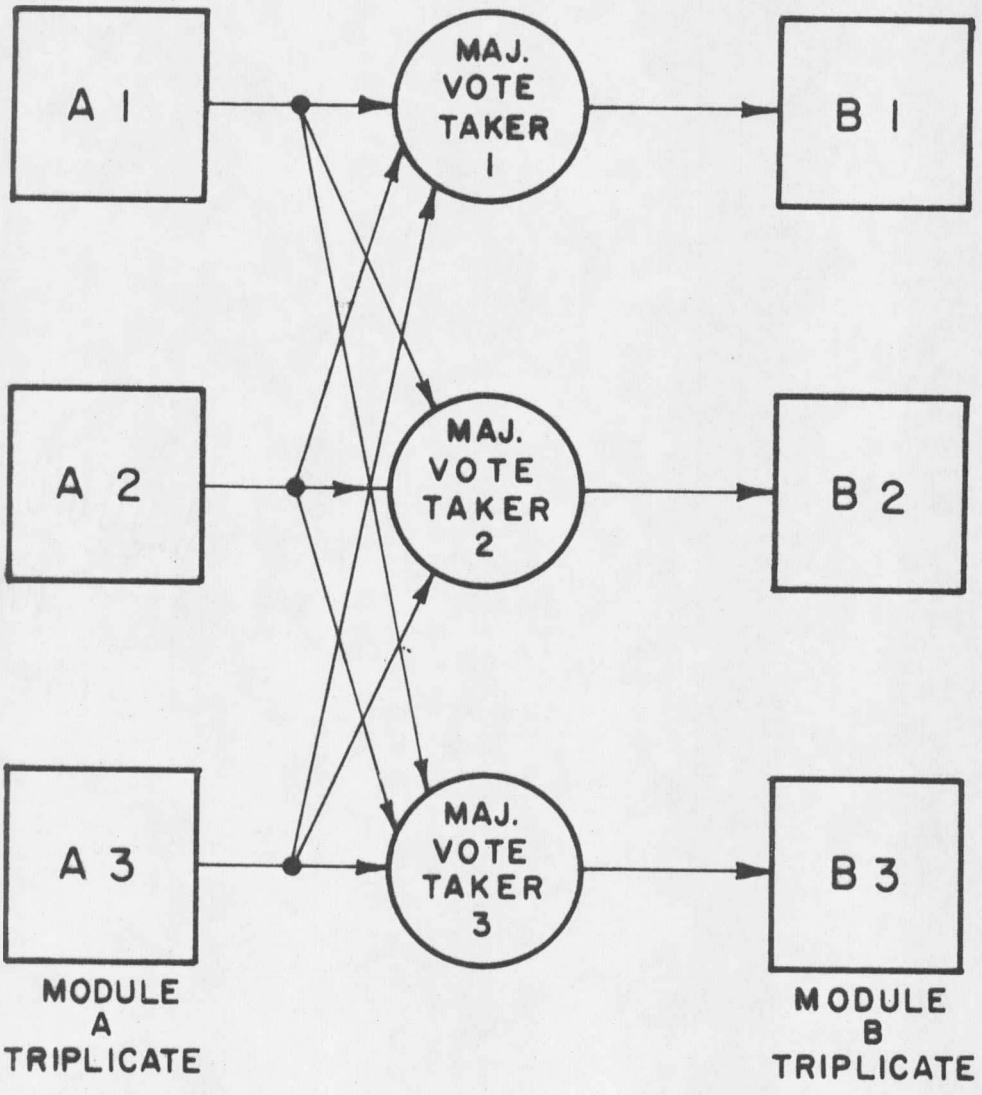


Figure 5.2 TMR Subunit with TMR Vote Takers.

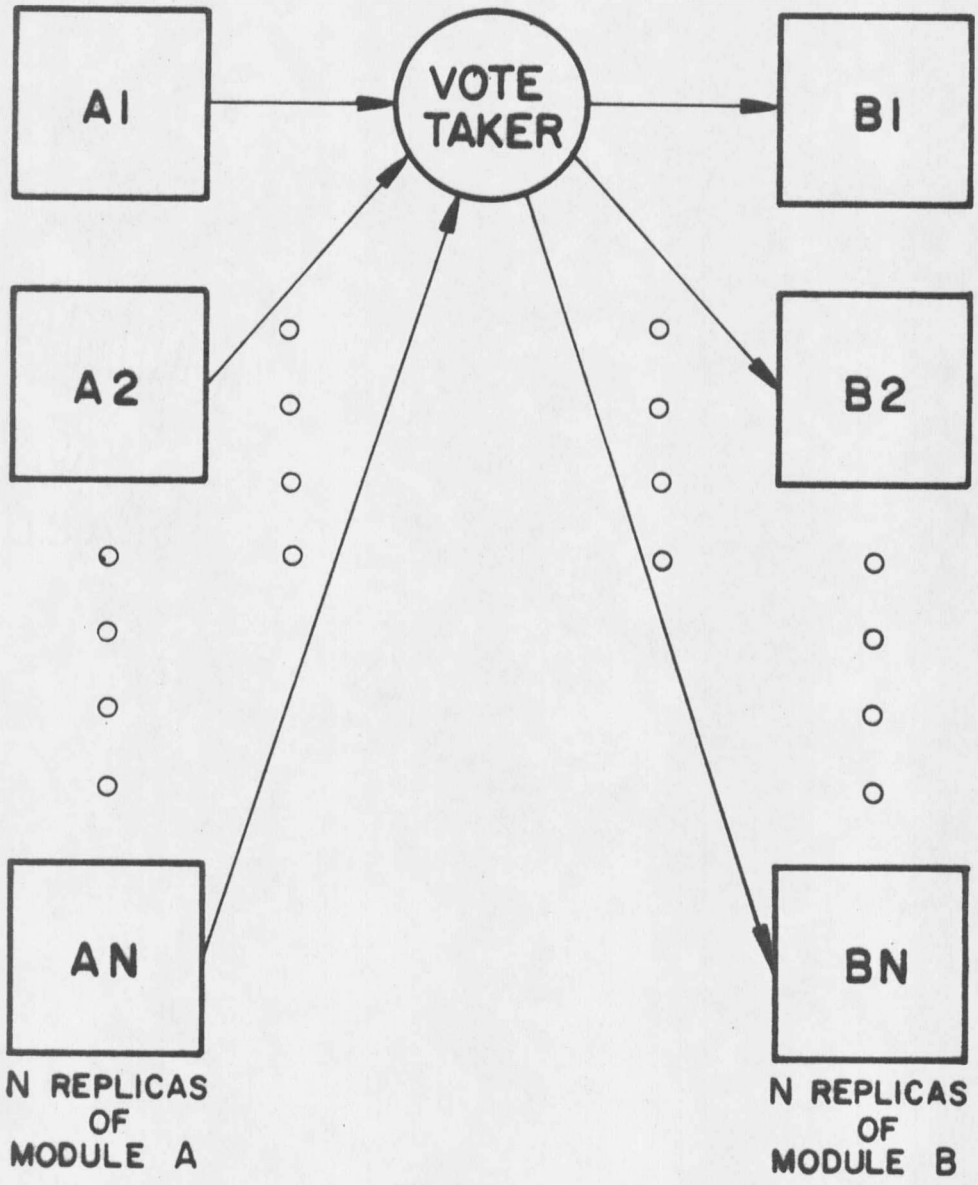


Figure 5.3 NMR Subunit.

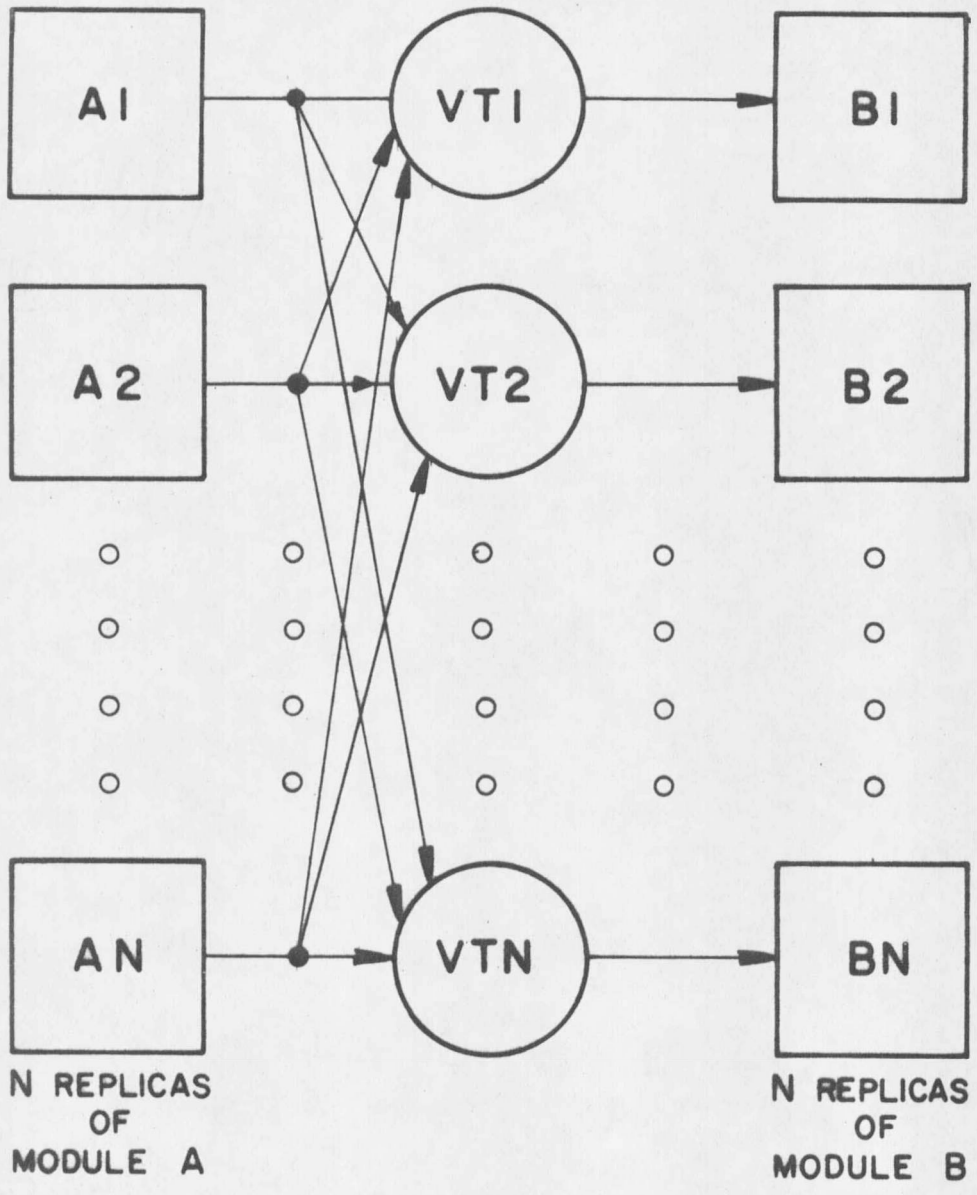


Figure 5.4 An NMR Subunit with N Vote Takers.

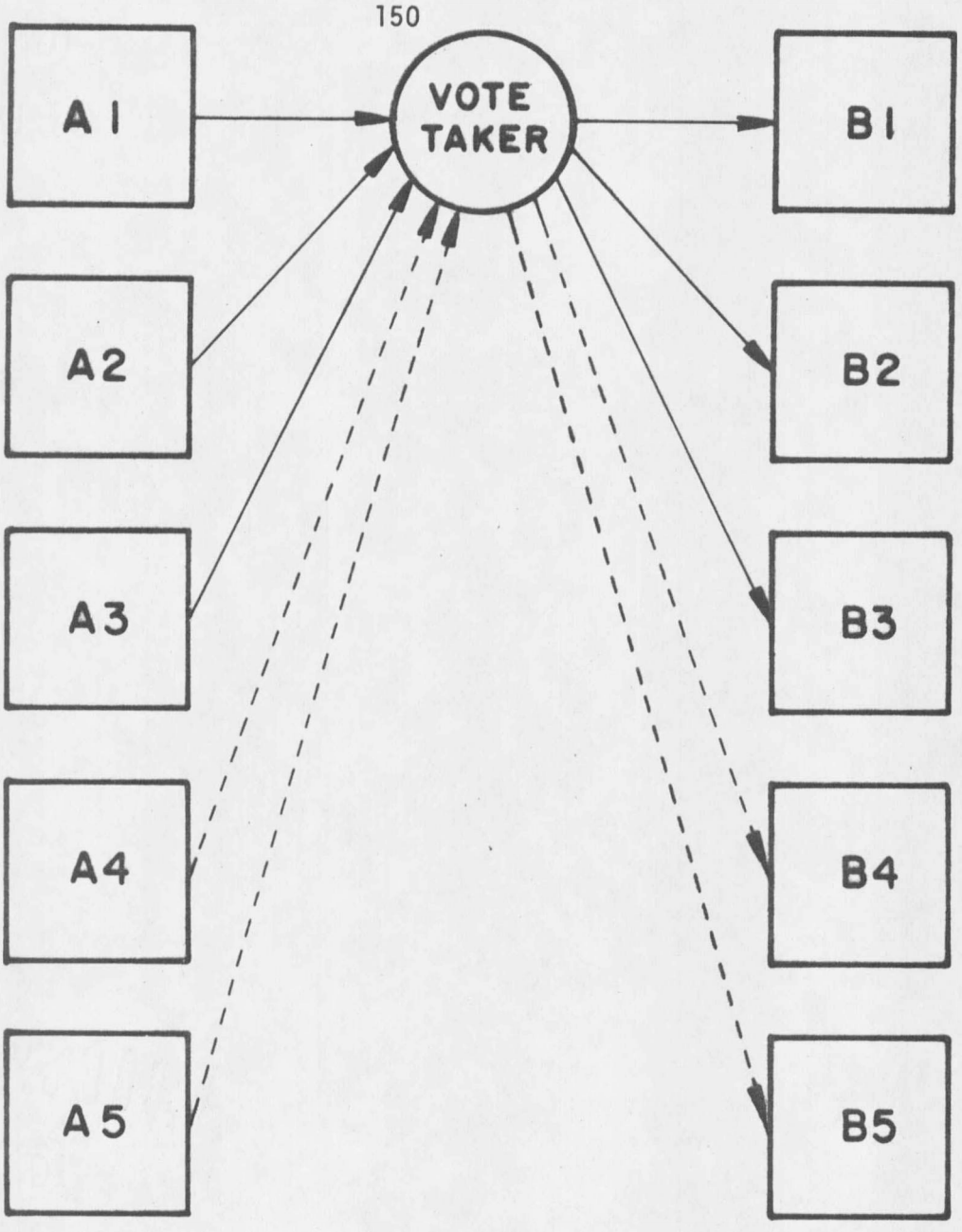


Figure 5.5 An HMR (3,5) Subunit.

In case an error is encountered in the next partition a roll-back process will take place. Rolling back requires the loading of the last saved state of the system.

The penalty paid in using a roll-back technique is manifested in: 1) the extra storage required to save the state parameters of the program, 2) the time spent in storing the state parameters after the successful completion of the partition represented by this state, 3) the time spent in bringing this information back to the 3-D machine when an error is encountered, and 4) time required to restore the 3-D machine environment. This includes such things as the time to change cancellation numbers of the cells and to decode the control word, etc.

Program Status Array of Words (PSAW) is a tool which proves to be helpful in software debugging. For GP computers, two words are sufficient to describe the status of the central processing unit (CPU) of most machines. The 3-D machine requires more than two words as described in the following section.

5.5.1 Program Status Array of Words (PSAW)

It takes about two words to describe the status of the CPU of a GP computer, e.g. the Sigma 7 computer's program status double-word (52) (PSDW). An array of words is required to describe the status of the 3-D machine. Two words of the array describe the status of the

supervisory machine and are called the Supervisor Program Status Double-Word (SPSD). Another three words describe the 3-D instruction status and include a pointer to a block of 32 words. These 32 words are copied from a collection of flip flops called the condition codes (CONC is the abbreviation to be distingquished from CC which is reserved for the cancellation code). There are 256 CONC's, one per cell of the CP. Each CONC is 4 bits long, allowing 16 different states of the contents of the CP cell to be recognized. Those currently used are positive, negative, and zero. Other conditions such as overflow and underflow can be included as appropriate. Figure 5.6 shows the PSAW for the 3-D machine. A listing of the meaning of the different fields of the PSAW follows (The SPSD is taken from the Sigma 7 reference manual) (56).

CONC Supervisor's condition code. This generalized 4 bit code indicates the nature of the result of an instruction. In some operations, only a portion of the condition code is involved; thus, the term CONC1 refers to the first bit of the condition code, CONC2 refers to the second bit, and so forth.

0 Means unused or parity bit.

FS Floating significance mode control.

FZ Floating zero mode control.

FN Floating normalize mode control.

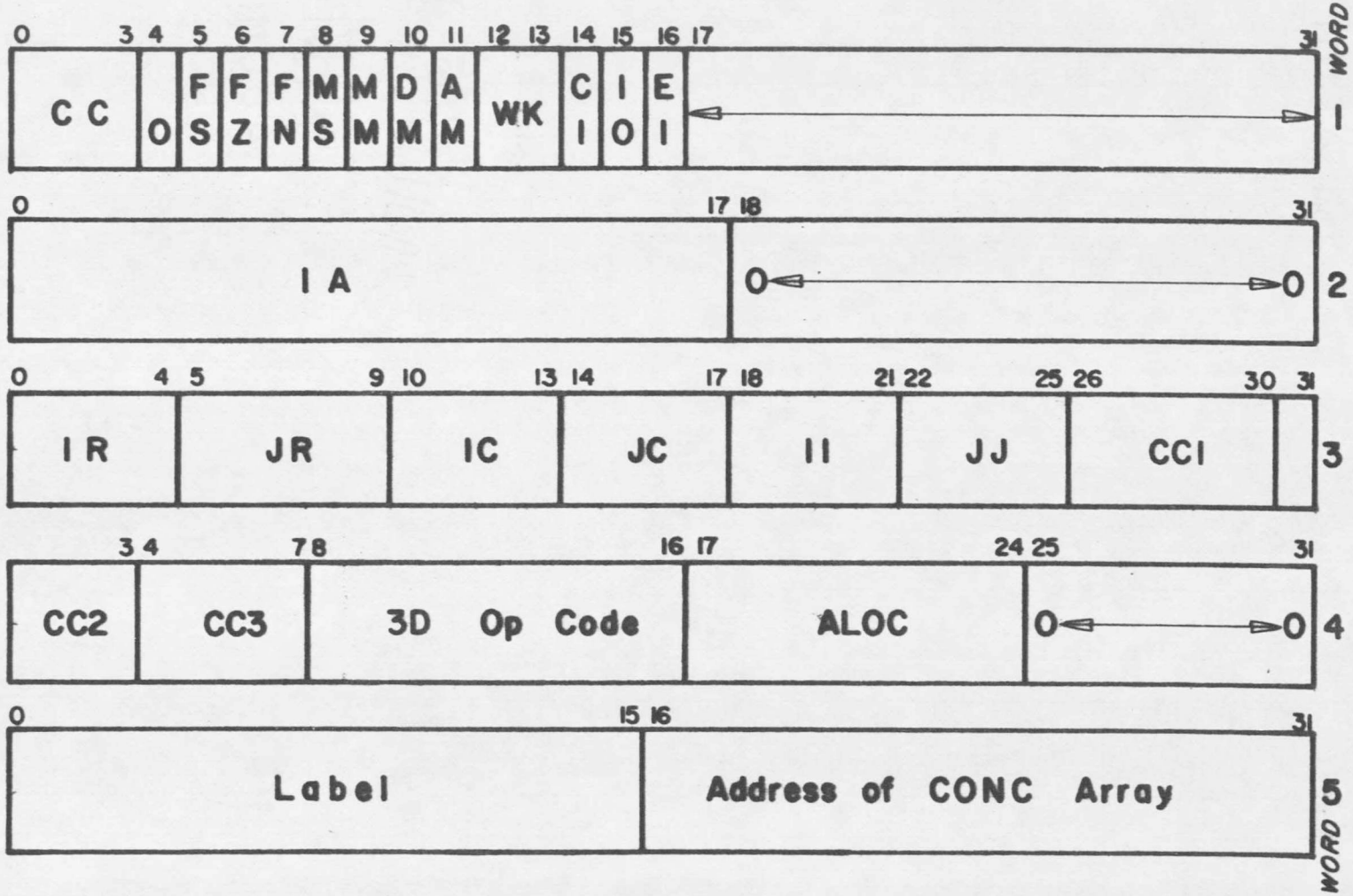


Figure 5.6 PSAW Format.

MS Master/slave mode control. The supervisor is in the master mode when this bit is 1; it is in the slave mode when this bit is 0.

MN Supervisor's memory map control. An optional memory map is in effect when this bit is a 1.

DM Supervisor's decimal mask.

AM Supervisor's arithmetic mask for fixed point arithmetic overflow.

IA Supervisor's instruction address. This 17-bit field contains the virtual address of the next instruction to be executed.

WK Supervisor's write key. This field designates the protection class of the supervisor's memory.

CI Supervisor's counter interrupt group inhibit.

IO Supervisor's I/O interrupt group inhibit.

EI Supervisor's external interrupt group inhibit.

RP Supervisor's register pointer which selects one of 32 possible blocks of general registers as the block current.

IR The index of the first row of the array in the 3-D machine on which the 3-D instruction is executed.

JR The index of the last row of the same array.

IC The index of the first column of the same array.

JC The index of the last column of the same array.

II The index of the source plane in the 3-D machine.

JJ The index of the destination plane in the 3-D machines.

CC1 Cancellation code for MR1 register set.

CC2 Cancellation code for MR2 register set.

CC3 Cancellation code for the CP.

3-D op code The operation code for the 3-D macro instruction to be executed.

ALOC The operation code for ganged operation of the CP.

label Is a binary representation (mapping) of the decimal label appended to the 3-D instruction.

Address of CP's CONC array Is the address of a 32 words block which contains the CONC of the CP.

The PSAW is a useful tool to trace the source of error when a job is aborted.

5.6 Software/Hardware (Hybrid) Fault-Tolerant Techniques Pertinent to the 3-D Machine Organization

The techniques to be described are peculiar to the 3-D machine organization. The first technique is a hybrid-by-subplane redundancy method which is similar to hardware redundancy methods. It assumes that each of the subplanes acts as a one-out-of-n similar hardware processors.

The second technique to be considered is the hybrid-by-plane

technique and is applicable to problems for which time of execution can be tripled or multiplied by n for n > 3 without great penalty. In this case several solutions of the problem are obtained from different planes of the MC. These planes should be interwoven by the program to produce intermediate and final results in the appropriate locations to be compared. A majority voting scheme is assumed. A description of both techniques follows.

5.6.1 Hybrid-by-Subplane Fault-Tolerant Computing

The modularity of the 3-D machine architecture makes it possible to duplicate a problem in the 3-D machine in the same plane if the array size of the problem is less than the size of the plane, e.g. for an n x n MP, and for an n/2 x n/2 size problem, up to 4 copies of the same problem can be handled simultaneously using a single MP. It does not take extra time to execute a 3-D instruction on the duplicates of the problem as long as all the duplicates are in the same plane. This hybrid technique will require extra time for comparing the duplicates of the problem whenever the program reaches a suitable state for comparison. This extra time is assumed to be negligible compared to the total time of execution which the problem takes. A majority vote is used to decide on the right solution at the check point in the program. This right solution should be propagated to all duplicates, and the program continues.

5.6.2 Hybrid-by-Plane Fault-Tolerant Computing

A hybrid-by-subplane redundant system is characterized by parallel execution of a parallel organized set of duplicate programs. By the same token, a hybrid-by-plane redundant system is characterized by serial execution of a parallel organized set of duplicate programs.* This definition implies a warning against using this hybrid-by-plane technique unless it is the only alternative to achieve a fault-tolerant computing device.

For problems with large size arrays it becomes necessary to utilize more than one plane to achieve the required redundancy. By having data occupying more than one plane, not more than one interplane 3-D operation can take place simultaneously, which accounts for the serial execution of the instruction in different planes. An example of interplane 3-D instructions which cannot be executed simultaneously is a set of TRAN instructions because of possible data path conflicts in the interplane transfer. This limitation does not apply to all 3-D instructions. A hybrid-by-plane redundant system, although characterized by execution of parallel organized programs, does some parallel execution of the programs whenever this parallel execution does not create any data path conflicts.

E.g. more than one plane can be performing a conductive trans-

* Each program on a single plane.

fer simultaneously. As an example of hybrid-by-plane redundant computing, assume that a problem can be confined to a single plane and due to its importance it has to be solved using an N-redundancy scheme. This means loading N planes with duplicates of the same data. Assume that the problem can be solved in M steps, and let $f_j(i)$ identify the j^{th} step of the solution on the i^{th} plane. Figure 5.7 shows a flow chart of the succession of steps which take place until the correct answer is obtained recorded in all the N planes.

5.7 Fault Detection and Diagnostic Routines

5.7.1 Fault Detection

There are several levels of error detection according to the level of fault-tolerant computing as explained before. Three of these error levels are presented next. These are:

1. Parity error detection.
2. Vote disagreement detection.
3. Unsuccessful roll-back.

5.7.1.1 Parity Error Detection

Using the Hamming code described in Section 5.4.1 a single error is detected and corrected by the parity checking hardware. Multiple errors can be signalled out to the supervisor machine. Depending upon

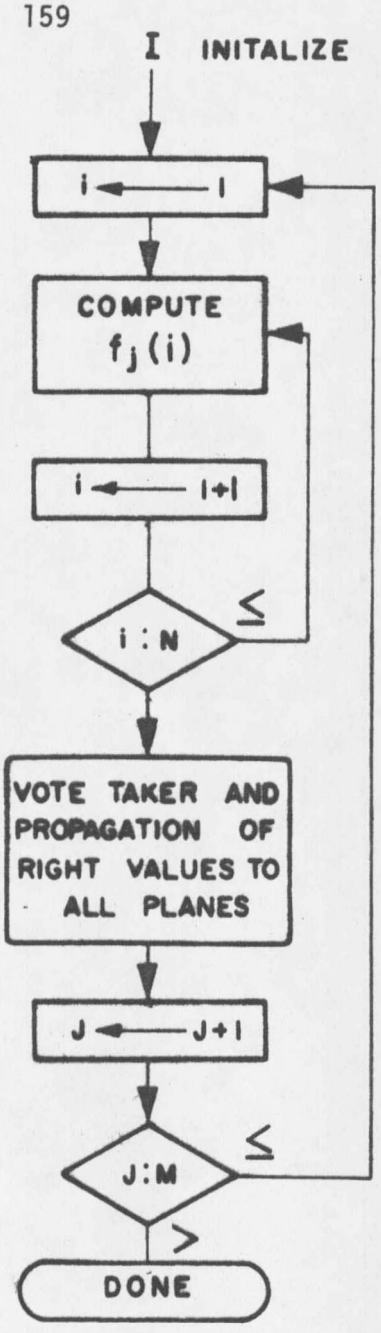


Figure 5.7 Flow chart of a Hybrid by-plane Fault-tolerant computing system.

the type of application the supervisor machine can either ignore the signal or stop the machine. The PSAW together with the parity checking signal determine the location of the faulty word.

5.7.1.2 Vote Disagreement Detection

For the 3-D machine subunits which have hardware redundancy with vote takers and decision devices, a module which disagrees with the majority vote is reported to the supervisor machine. The supervisor machine stores the location of that module in a fault observance table. Upon a second report of a fault in that module, the module is isolated and a 3-D machine reconfiguration takes place if necessary. 3-D machine reconfiguration will be considered in Section 5.8.

5.7.1.3 Unsuccessful Roll-back

Whenever roll-back is used, the programmer should specify the number of retries. If the number of retries in a given program exceeds the maximum allowed by the programmer, this situation is reported to the supervisor machine (the reporting is affected by the system's program). The supervisor machine decides on either continuing with executing the program, disregarding the fault or aborting the job. The PSAW contains the location of the offending instruction and may be able to locate any faulty module which might be causing the error.

5.7.2 Remarks on Diagnostic Routines

It is essential for the operation of the 3-D machine to run some diagnostic routines. The complexity of a diagnostic program is correlated to the seriousness of the situation. A routine diagnostic program is usually much simpler than a program designed to find a suspected erroneous cell.

A routine diagnostic program for the MC might consist of a pattern of 0's and 1's which upon input to the 3-D machine is non-conductively transferred to all memory planes. A transfer to the CP from each memory planes and a comparison of the contents of the CPA and CPQ is affected. Assuming that the same pattern of 0's and 1's was originally loaded in the MR1 and MR2 register sets. The condition codes (CONC) of the CP should indicate equality. Scanning the PSAW and recording the location of cells for which the equality condition is not valid does include the routine diagnosis program for the MC. Similar procedures can be used to construct routine diagnosis programs to examine the IOP and the CR. For the CP diagnosis, the aforementioned procedure is necessary but not sufficient since the CP is more than a storage unit. A program which executes all the basic ALO's on known operands with pre-calculated results and a comparison of the results obtained from the CP with the pre-calculated results locates the malfunctioning subunits of the CP.

5.8 3-D Machine Reconfiguration

Whenever the programmer asks for rows IR to JR and columns IC to JC to process his problem, the X and Y decoders do a memory mapping to provide the required rectangular array of cells which does not have to be at the same physical location sought by the programmer (memory mapping). In the case of a faulty cell encountered inside the problem boundaries, the X and Y decoders bypass the x and y location of this faulty cell, this in essence isolates the row and column in which the faulty cell exists. For a problem mapped to an area with a faulty cell with an array size less than the size of the MP, this isolation does not apply to the remainder of the MP outside the array boundaries of the problem.

As an example consider a problem with IR = 1, JR = 4, IC = 1, JC = 4. Assume that the X and Y decoders find the first available row to be row 4 and the first available column to be column 4; and if cell (5,6) is in error then the X and Y decoders have to ignore row 5 and column 6. The following correspondence holds:

Row:		Column:	
1	→ 4	1	→ 4
2	→ 6	2	→ 5
3	→ 7	3	→ 7
4	→ 8	4	→ 8

The array as described by the programmer and its mapping in the 3-D machine are shown in Figure 5.8.

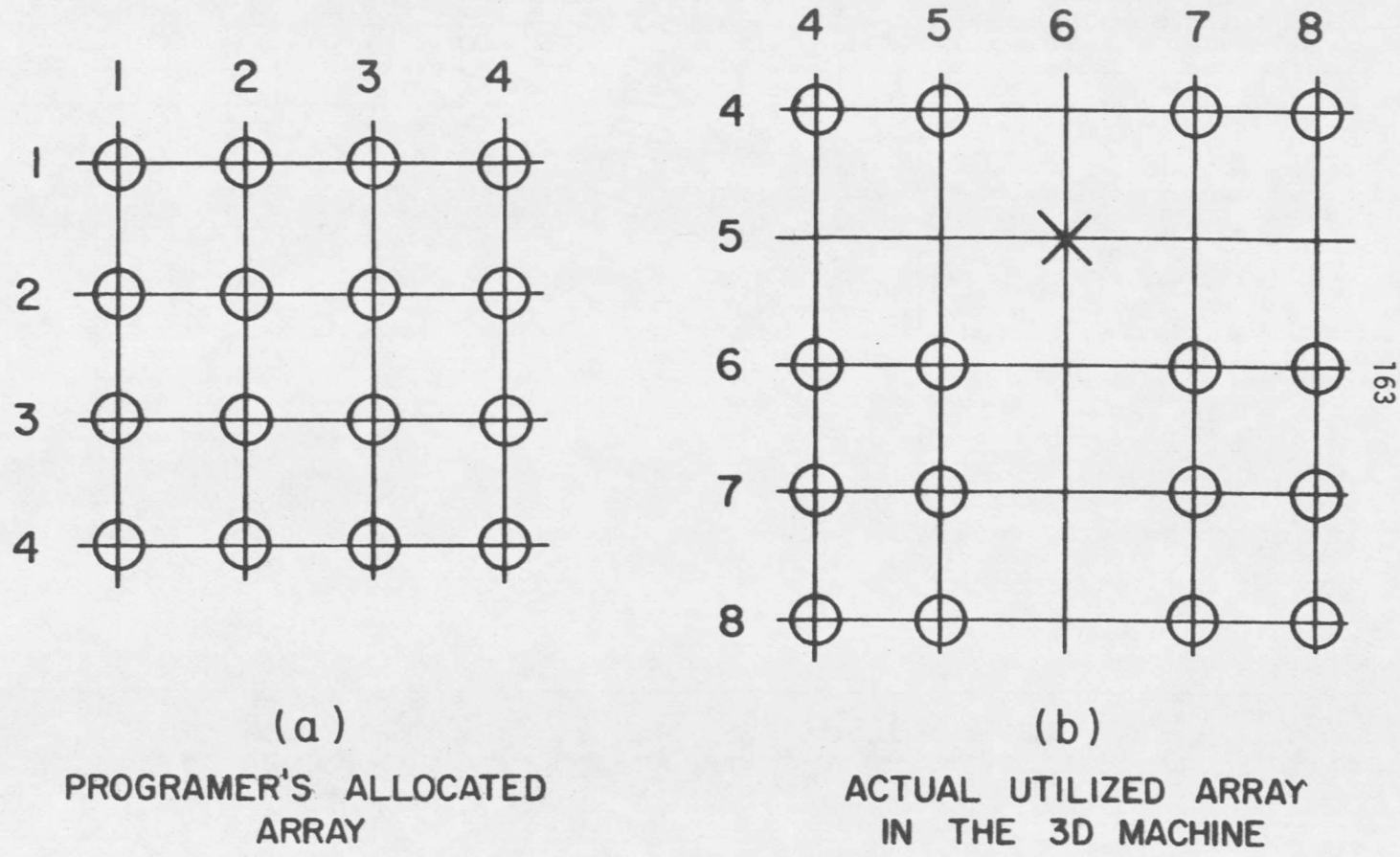


Figure 5.8 Fault Tolerant Active Subplane of the 3-D Machine.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

6.1.1 Summary

A register transfer level design of the 3-D machine has been presented in this thesis. An interpreter-simulator (SIM3D) has been written and tried successfully on several matrix-type standard operations. Fault-tolerant design and computing using the 3-D machine has been discussed in general terms which establish the groundwork necessary for subsequent fault-tolerant design and computing to suit the architecture of the 3-D machine.

6.1.2 System Hardware Design

The MC design has been thoroughly investigated. Intraplane transfer has been well established. Interplane transfer has been proposed to be either conductive or non-conductive. Conductive transfer can be handled with the same conventional bussing techniques used for intraplane transfer, however, adds to the pin limitation problem and may make plane replacement a hard task. Non-conductive transfer alleviates the pin limitation problem, but also it presents the designer with some new problems. The plane alignment problem is severe in that a few microns of laser beam misalignment would cause information transfer to the wrong place or even no transfer at all. Also, most of the state-of-the-art work in the area of non-conductive data

transfer is concerned with mass storage and transfer rather than individual word storage and transfer, which are the ultimate concern of this thesis. These add to the difficulties faced with the non-conductive transfer in the 3-D machine.

The CR design is very similar to that of the MC if reconfiguration of control words is to be included; otherwise the CR planes will be much simpler than the MP's. A standard CR set of planes to be provided are those which can handle the algorithms described in Chapter 4. Other CR planes can be added as needed to handle non-standard algorithms.

The IOP design or set of designs to be implemented will depend on the needs presented by the particular type of problems to be handled by the 3-D machine. The same is true of the CP.

LSI is the only way to build a cost-effective machine if the initial investment in the different LSI chips can be alleviated by modularity of chip design.

6.1.3 Software Considerations

SIM3D when applied to matrix-oriented problems provided the following execution rates for nxn matrixes:

1. A matrix transpose required $\left\lceil \frac{n-1}{3} \right\rceil$ execution cycles as compared to n^2-n execution cycles for an equivalent serial machine.

2. A matrix multiplication required 3n cycles as compared to $n^2(2n-1)$ execution cycles on an equivalent serial machine.
3. A matrix inversion required 3n cycles as compared to $n^2(2n-1)$ execution cycles for an equivalent serial machine.
4. A search for a maximum or a minimum is independent of the number of elements of the array as compared to n^2 execution cycles for a linear search in equivalent serial machine.

The actual saving in execution time is greater than the theoretical limits mentioned above if we consider the number of cycles it takes for bookkeeping in the serial machine as compared to the 3-D machine. The bookkeeping includes index calculation and memory accessing for the element whose index is being calculated. To achieve the kind of throughput which matches the execution rate mentioned above, the I/O design and supporting I/O software should be chosen such that the overall 3-D machine throughput is not I/O bound.

6.1.4 Fault-tolerant Computing and Design

Fault-tolerant design for the conventional parts of the machine is achieved through TMR for the most part. NMR or HMR is recommended for control parts of crucial importance, (e.g. the CP control).

The 3-D machine has tremendous capabilities for fault-tolerant computing due to its structure. Replication of data within planes and or multiple planes presents the two extremes of inherited fault-

tolerant computing. In the first case, no time penalty is paid for executing several replicas of the same problem except for the time spent in comparing cell contents at certain bench marks. In the second case the penalty in execution time is correlated to the percentage of non-conductive transfers included in the instruction stream.

6.2 Recommendations for Future Research

Recommendations are as follows:

1. Complete the unfinished work on register-transfer design, especially in the IOP, after collecting enough information about the types of real-time applications to be handled, the methods of gathering data for these applications, and the secondary storage facilities to be interfaced to the 3-D machine. Also, the CP design will need to be completed by deciding on the number and type of micro-programmed arithmetic/logic processors to be used to control the CP register.
2. A gate-level design should be considered if the standard register transfer modules available from computer manufacturers cannot be used to build a prototype of the 3-D machine or if the cost of remodeling the already available modules increases the cost of building the machine

excessively.

3. Additional programming using SIM3D should be done toward the goal of thoroughly investigating the advantages and liabilities of its structure relative to the 3-D machine hardware capabilities.
4. Reorganization of the SIM3D language syntax into field free or order independent field form could possibly simplify and make more transparent the writing of 3-D programs. This would have to be balanced against any additional hardware or software inefficiencies created.
5. An investigation of the need for a high-level parallel processing language and consequently a parallel processing compiler should be made.
6. Much more extensive investigations of fault-tolerant computing and design as applied to the 3-D machine is desirable.
7. Investigation of the nature of the procedure which should be used to write an operating system for the 3-D machine complex is necessary. This investigation should take into consideration the fact that multiprogramming and fault tolerant computing are to be included in the operating system to make the machine worthwhile. This investigation is highly correlated to the hardware capabilities of the supervisory machine.

APPENDIX A

VALID CONNECTION CODES OF MEMORY CUBE INTRA-PLANE TRANSFER AND THEIR CORRESPONDING GATE SETTINGS.

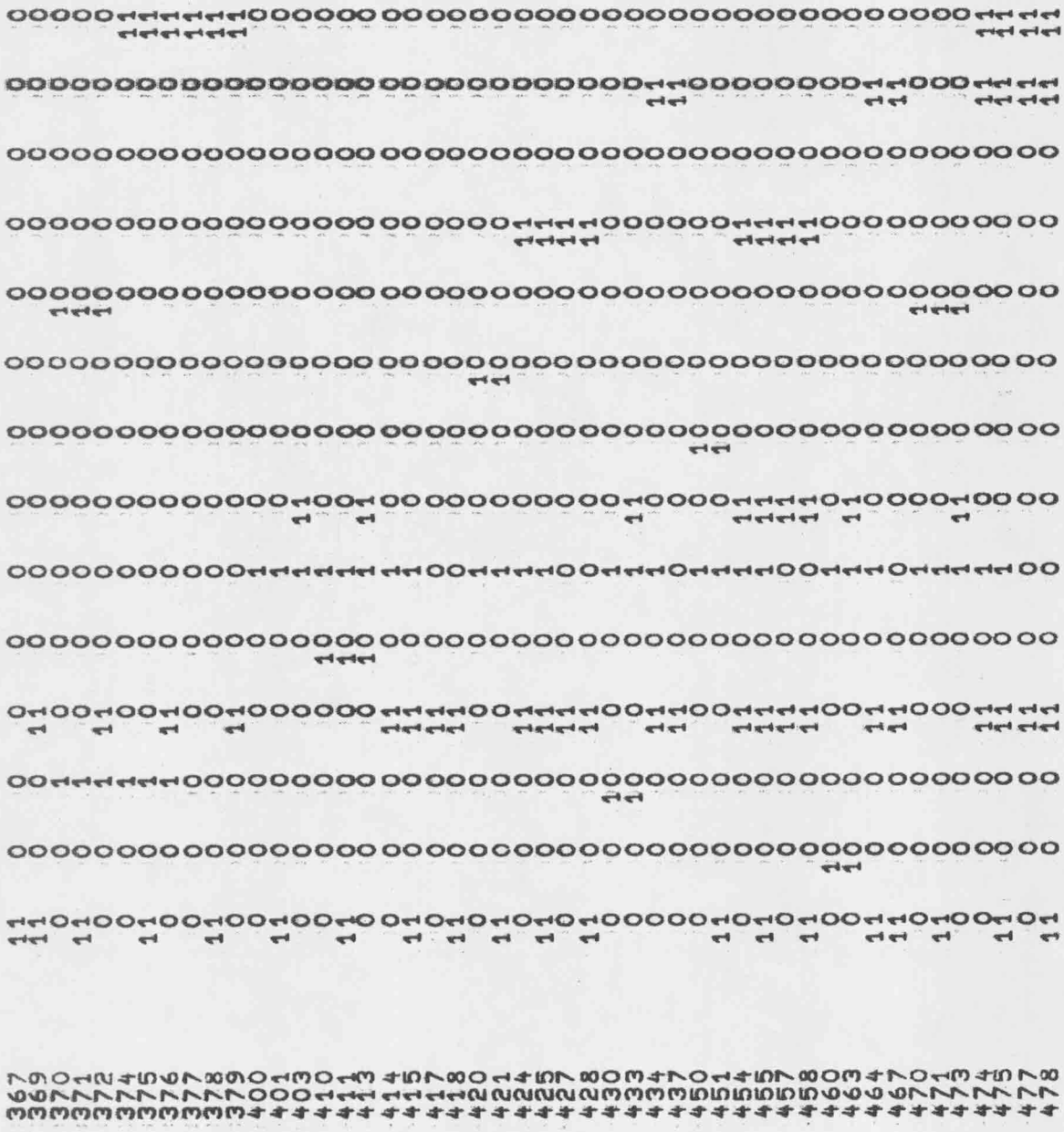
THE VALID INTRAPLANE (CONDUCTIVE) CONNECTION CODES OF THE MEMORY CUBE ARE LISTED IN ASCENDING ORDER

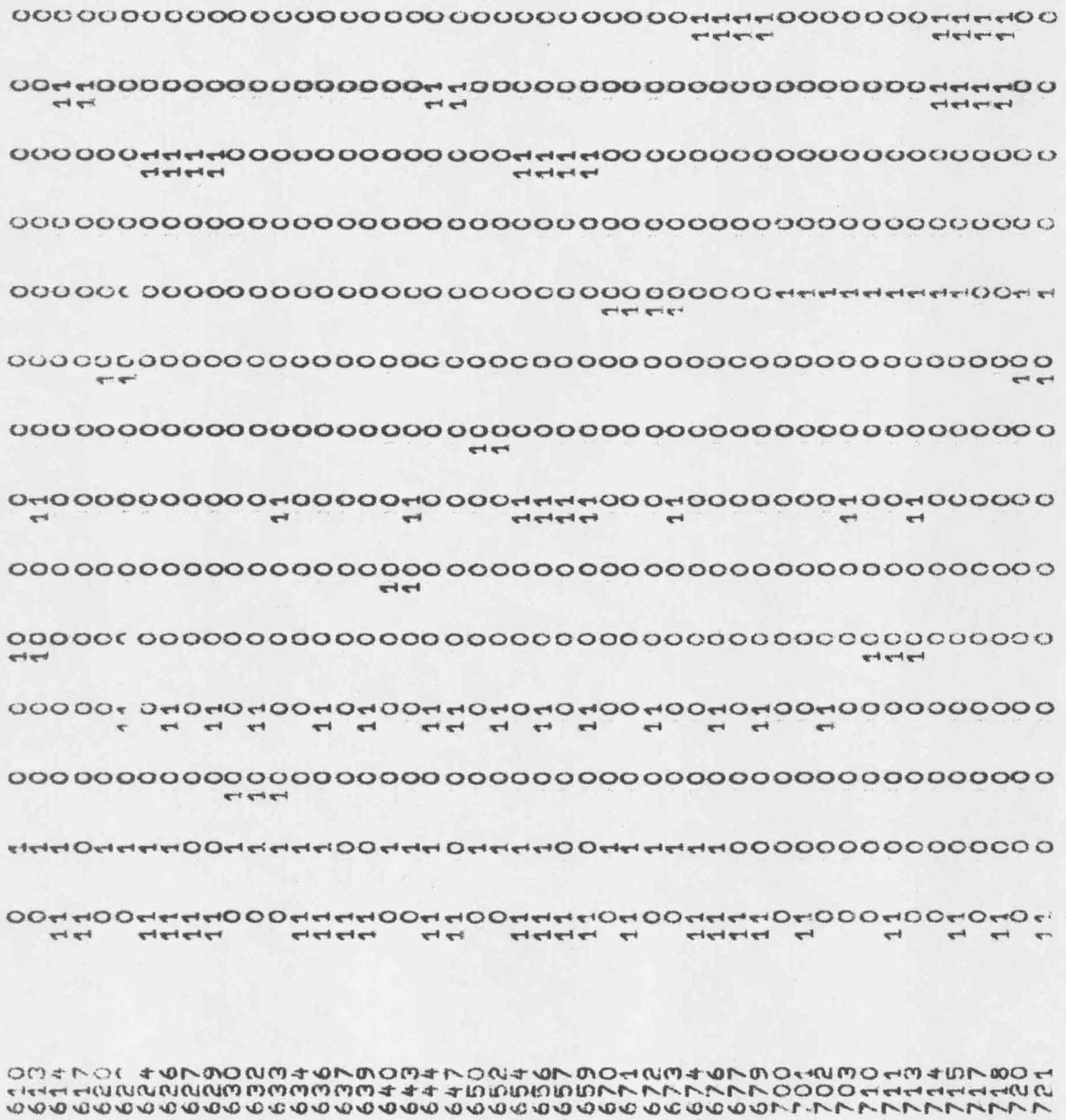
0	1	2	3	5	6	8	9	10	11	13	20	21	22	30	32	33	40	41	43
50	51	52	60	62	63	70	71	72	73	100	101	103	120	121	124	125	127	128	130
133	134	137	140	141	143	144	145	147	148	150	151	154	155	157	158	160	163	164	167
170	171	173	174	175	177	178	200	201	202	210	211	214	215	217	218	230	232	234	236
237	239	240	241	244	245	247	248	250	251	252	254	255	256	257	258	259	260	262	264
266	267	269	270	271	272	273	300	302	303	310	313	314	317	320	322	324	326	327	329
340	343	344	347	350	352	354	356	357	359	360	362	363	364	366	367	369	370	371	372
374	375	376	377	378	379	400	401	403	410	411	413	414	415	417	418	420	421	424	425
427	428	430	433	434	437	450	451	454	455	457	458	460	463	464	467	470	471	473	474
475	477	478	500	501	502	510	511	514	515	517	518	520	521	522	524	525	526	527	528
529	530	532	533	534	537	539	540	541	544	545	547	548	560	562	564	566	567	569	570
571	572	574	575	576	577	578	579	600	602	603	610	613	614	617	620	622	624	626	627
629	630	632	633	634	636	637	639	640	643	644	647	650	652	654	656	657	659	670	671
672	673	674	676	677	679	700	701	702	703	710	711	713	714	715	717	718	720	721	722
724	725	726	727	728	729	730	732	733	734	736	737	739	740	741	743	744	745	747	748
750	751	752	754	755	756	757	758	759	760	762	763	764	766	767	769				

A LISTING OF GATE SETTING CORESPONDING TO THE LEGAL CONNECTION CODES

00 CORESPONDCE TO AN OPEN GATE
11 CORESPONDCE TO A BIDIRECTIONAL GATE
10 CORESPONDCE TO A SOURCE CELL GATE
01 CORESPONDCE TO A DESTINATION CELL GATE

CONNECTION CODE	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	11	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	11	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	11	0	0	0	0	0	0	0
5	11	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	11	0	0	0	0	0	0	0	0	0	0
8	11	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	11	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	10	0	0	0	0	0	0	0	0	0
11	11	0	0	0	10	0	0	0	0	0	0	0	0	0
13	0	0	0	0	10	0	11	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	10	0	0	0	0	0
21	11	0	0	0	0	0	0	0	10	0	0	0	0	0
22	0	0	0	11	0	0	0	0	10	0	0	0	0	0
30	0	0	10	0	0	0	0	0	0	0	0	0	0	0
32	0	0	10	11	0	0	0	0	0	0	0	0	0	0
33	0	0	10	0	0	0	11	0	0	0	0	0	0	0
40	0	0	0	0	0	10	0	0	0	0	0	0	0	0
41	11	0	0	0	0	10	0	0	0	0	0	0	0	0
43	0	0	0	0	0	10	11	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	10	0	0	0	0	0	0
51	11	0	0	0	0	0	0	10	0	0	0	0	0	0
52	0	0	1	0	0	0	0	10	0	0	0	0	0	0
60	0	10	0	0	0	0	0	0	0	0	0	0	0	0
62	0	10	0	11	0	0	0	0	0	0	0	0	0	0
63	0	10	0	0	0	0	11	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	10	0	0	0	0
71	11	0	0	0	0	0	0	0	0	10	0	0	0	0
72	0	0	0	11	0	0	0	0	0	10	0	0	0	0
73	0	0	0	0	0	0	0	0	0	10	0	0	0	0
100	0	0	0	0	1	0	0	0	0	0	0	0	0	0
101	11	0	0	0	1	0	0	0	0	0	0	0	0	0
103	0	0	0	0	1	0	11	0	0	0	0	0	0	0
120	0	0	0	0	1	0	0	0	10	0	0	0	0	0
121	11	0	0	0	1	0	0	0	10	0	0	0	0	0
124	0	0	0	0	1	0	0	0	0	0	11	0	0	0
125	11	0	0	0	1	0	0	0	0	0	11	0	0	0
127	0	0	0	0	0	0	0	0	0	0	11	0	0	0
128	11	0	0	0	0	0	0	0	0	0	0	0	0	0





```
722          0    0    0   11    0    0    0    0   10    1    0    0    0
724          0    0    0    0    0    0    0    0    0    1   11   11   11
725         11    0    0    0    0    0    0    0    0    1   11   11   11
726          0    0    0   11    0    0    0    0    0    1   11   11   11
727          0    0    0    0    0    0    0    0    0    0   11   11   11
728         11    0    0    0    0    0    0    0    0    0   11   11   11
729          0    0    0   11    0    0    0    0    0    1   11   11   11
730          0    0   10    0    0    0    0    0    0    1    0    0    0
732          0    0   10   11    0    0    0    0    0    1    0    0    0
733          0    0   10    0    0    0   11    0    0    1    0    0    0
734          0    0    0    0    0    0    0    0    0    1    0    0   11
736          0    0    0   11    0    0    0    0    0    1    0    0   11
737          0    0    0    0    0    0    0    0    0    0    0    0   11
739          0    0    0   11    0    0    0    0    0    1    0    0   11
740          0    0    0    0   10    0    0    0    0    1    0    0    0
741         11    0    0    0   10    0    0    0    0    1    0    0    0
743          0    0    0    0   10   11    0    0    0    1    0    0    0
744          0    0    0   11    0    0    0    0    0    1    0    0   11
745         11    0    0   11    0    0    0    0    0    1    0    0   11
747          0    0    0   11    0    0    0    0    0    1    0    0   11
748         11    0    0   11    0    0    0    0    0    0    0    0   11
750          0    0    0    0    0    0   10    0    0    1    0    0    0
751         11    0    0    0    0    0   10    0    0    1    0    0    0
752          0    0    0   11    0    0    0    0    0    1    0    0    0
754          0    0    0    0    0   11    0    0    0    1   11    0   11
755         11    0    0    0    0   11    0    0    0    1   11    0   11
756          0    0    0   11    0   11    0    0    0    1   11    0   11
757          0    0    0    0    0   11    0    0    0    0   11    0   11
758         11    0    0    0    0   11    0    0    0    0   11    0   11
759          0    0    0   11    0   11    0    0    0    1   11    0   11
760          0   10    0    0    0    0    0    0    0    1    0    0    0
762          0   10    0   11    0    0    0    0    0    1    0    0    0
763          0   10    0    0    0   11    0    0    0    1    0    0    0
764         11    0    0    0    0    0    0    0    0    1    0    0   11
766         11    0    0   11    0    0    0    0    0    1    0    0   11
767         11    0    0    0    0    0    0    0    0    0    0    0   11
769         11    0    0   11    0    0    0    0    0    1    0    0   11
```

APPENDIX B

A QUINE-MCKLUSKY MINIMIZATION ALGORITHM APPLIED TO THE GATE SETTING AS OUTPUT AND CONNECTION CODES AS INPUT TO A CONNECTION CODE COMBINATION LOGIC DECODER

A LISTING OF CONNECTION CODES AS MINTERMS COREPONDING TOINDIVIDUAL
GATE SETTING AS IT IS CLOSED IN MEMORY CUBE

CASE# 1
CASE 1 IS A BIDIRECTIONAL GATE G1
NO OF MINTERMS= 115
***MINTERMS ARE ***

1	5	8	11	21	41	51	71	101	121	125	128	141	145	148	151	155	158	164	167
171	175	178	201	211	215	218	241	245	248	251	255	258	264	266	267	269	271	364	366
367	369	371	375	378	401	411	415	418	421	425	428	451	455	458	464	467	471	475	478
501	511	515	518	521	525	528	541	545	548	564	566	567	569	571	575	578	614	617	624
626	627	629	634	636	637	639	644	647	654	656	657	659	671	674	676	677	679	701	711
715	718	721	725	728	741	745	748	751	755	758	764	766	767	769					

CASE# 2
CASE 2 IS A BIDIRECTIONAL GATE G4
NO OF MINTERMS= 111
***MINTERMS ARE ***

2	6	9	22	32	62	72	144	145	147	148	202	232	236	239	244	245	247	248	252
256	259	262	266	269	272	302	322	326	329	344	347	352	356	359	362	366	369	372	376
379	414	415	417	418	424	425	427	428	434	437	454	455	457	458	464	467	474	475	477
478	502	522	526	529	532	539	544	545	547	548	562	566	569	572	576	577	578	579	602
622	626	629	632	636	639	644	647	652	656	659	672	676	679	702	722	726	729	732	736
739	744	745	747	748	752	756	759	762	766	769									

CASE# 3
CASE 3 IS A BIDIRECTIONAL GATE G7
NO OF MINTERMS= 88
***MINTERMS ARE ***

3	13	33	43	63	103	133	143	154	155	157	158	163	173	244	245	254	255	256	257
258	259	273	303	313	343	354	356	357	359	363	403	413	433	454	455	457	458	463	473
514	515	517	518	524	525	526	527	528	529	533	534	537	539	544	545	547	548	564	566
567	569	574	575	576	577	578	579	603	613	633	643	654	656	657	659	673	703	713	733
743	754	755	756	757	758	759	763												

CASE# 4
CASE 4 IS A BIDIRECTIONAL GATE G11
NO OF MINTERMS= 31
***MINTERMS ARE ***

124	125	127	154	155	157	158	214	215	217	218	244	245	247	248	424	425	427	428	454
455	457	458	514	515	517	518	544	545	547	548									

CASE# 5
CASE 5 IS A BIDIRECTIONAL GATE G12
NO OF MINTERMS= 45
***MINTERMS ARE ***

264	266	267	269	324	326	327	329	354	356	357	359	534	537	539	564	566	567	569	574
575	576	577	578	579	624	626	627	629	654	656	657	659	724	725	726	727	728	729	754

```
 755 756 757 758 759

CASE# 6
 CASE 6 IS A BIDIRECTIONAL GATE G13
 NO OF MINTERMS=    38
 ***MINTERMS ARE ***
 134 137 164 167 174 175 177 178 234 236 237 239 244 246 314 317 344 347 434 437
 464 467 474 475 477 478 614 617 644 647 714 715 717 718 744 745 747 748

CASE# 7
 CASE 7 IS A BIDIRECTIONAL GATE G14
 NO OF MINTERMS=    52
 ***MINTERMS ARE ***
 174 175 177 178 374 375 376 377 378 379 474 475 477 478 574 575 576 577 578 579
 674 676 677 679 714 715 717 718 724 725 726 727 728 729 734 736 737 739 744 745
 747 748 754 755 756 757 758 759 764 766 767 769

CASE# 8
 CASE 8 IS DESTINATION GATE G2
 NO OF MINTERMS=    28
 ***MINTERMS ARE ***
 600 602 603 610 613 614 620 622 624 626 630 632 633 634 636 640 643 644 650 652
 654 656 670 671 672 673 674 676

CASE# 9
 CASE 9 IS DESTINATION GATE G3
 NO OF MINTERMS=    29
 ***MINTERMS ARE ***
  52 300 302 303 310 313 314 320 322 324 326 340 343 344 350 352 354 356 360 362
 363 364 366 370 371 372 374 375 376

CASE#10
 CASE 10 IS A DESTINATION GATE G5
 NO OF MINTERMS=    27
 ***MINTERMS ARE ***
 100 101 103 120 121 124 125 130 133 134 140 141 143 144 145 150 151 154 155 160
 163 164 170 171 173 174 175

CASE#11
 CASE 11 IS A DESTINATION GATE G6
 NO OF MINTERMS=    27
 ***MINTERMS ARE ***
 400 401 403 410 411 413 414 415 420 421 424 425 430 433 434 450 451 454 455 460
 463 464 470 471 473 474 475

CASE#12
 CASE 12 IS A DESTINATION GATE G8
 NO OF MINTERMS=    31
```

```
 ***MINTERMS ARE ***
 500 501 502 510 511 514 515 520 521 522 524 525 526 530 532 533 534 540 541 544
 545 560 562 564 566 570 571 572 574 575 576

CASE#13
 CASE 13 IS A DESTINATION GATE G9
 NO OF MINTERMS=   29
 ***MINTERMS ARE ***
 200 201 202 210 211 214 219 230 232 234 236 240 241 244 245 250 251 252 254 255
 256 260 262 264 266 270 271 272 273

CASE#14
 CASE 14 IS A DESTINATION GATE G10
 NO OF MINTERMS=   36
 ***MINTERMS ARE ***
 700 701 702 703 710 711 713 714 715 720 721 722 724 725 726 730 732 733 734 736
 740 741 743 744 745 750 751 752 754 755 756 760 762 763 764 766

CASE#15
 CASE 15 IS A SOURCE GATE G2
 NO OF MINTERMS=   17
 ***MINTERMS ARE ***
  60  62  63 160 163 260 262 360 362 363 460 463 560 562 760 762 763

CASE#16
 CASE 16 IS A SOURCE GATE G3
 NO OF MINTERMS=   18
 ***MINTERMS ARE ***
  30  32  33 130 133 230 232 430 433 530 532 533 630 632 633 730 732 733

CASE#17
 CASE 17 IS A SOURCE GATE G5
 NO OF MINTERMS=   17
 ***MINTERMS ARE ***
  10  11  13 210 211 310 313 410 411 413 510 511 610 613 710 711 713

CASE#18
 CASE 18 IS A SOURCE GATE G6
 NO OF MINTERMS=   17
 ***MINTERMS ARE ***
  40  41  43 140 141 143 240 241 340 343 540 541 640 643 740 741 743

CASE#19
 CASE 19 IS A SOURCE GATE G8
 NO OF MINTERMS=   17
 ***MINTERMS ARE ***
  50  51  52 150 151 250 251 252 350 352 450 451 650 652 750 751 752
```

```
CASE#20
 CASE 20 IS A SOURCE GATE G9
 NO OF MINTERMS=   17
 ***MINTERMS ARE ***
  20   21   22 120 121 320 322 420 421 520 521 522 620 622 720 721 722

CASE#21
 CASE 21 IS A SOURCE GATE G10
 NO OF MINTERMS=   24
 ***MINTERMS ARE ***
  70   71   72   73 170 171 173 270 271 272 273 370 371 372 470 471 473 570 571 572
 670 671 672 673
```

```
THIS QMCLUSKY OUTPUT DOES NOT SHOW THE N'S CUBE
BECAUSE OF LIMITED SPACE
IT ALSO DOES NOT SHOW THE BINARY EQUIVALENT
OF ITS TERMS
          CASE #   1
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  128  1     8  1     1  1   528  2   264  2   258  2     5  2   769  3   656  3   644  3
  578  3   548  3   545  3   521  3   518  3   515  3   266  3   164  3   148  3   145  3
   41  3    21  3    11  3   676  4   674  4   657  4   624  4   564  4   525  4   464  4
  418  4   401  4   269  4   267  4   201  4   178  4   141  4   101  4    71  4    51  4
  728  5   721  5   677  5   659  5   654  5   647  5   626  5   617  5   614  5   569  5
  566  5   541  5   458  5   451  5   428  5   425  5   421  5   389  5   364  5   271  5
  248  5   241  5   218  5   211  5   171  5   167  5   158  5   155  5   151  5   121  5
  748  6   745  6   741  6   725  6   718  6   715  6   711  6   679  6   636  6   634  6
  629  6   627  6   571  6   567  6   467  6   455  6   411  6   378  6   371  6   366  6
  245  6   215  6   175  6   125  6   764  7   758  7   755  7   701  7   671  7   637  7
  575  7   501  7   478  7   475  7   471  7   415  7   375  7   367  7   251  7   766  8
  751  8   639  8   255  8   767  9   511  9
# OF BITS=10          # OF MINTERMS=  115        LARGEST MINTERM= 769
LIST OF ESSENTIAL MINTERMS
  128   769   578   545   518   515   148    41   674   464   418   201   178   141   101    71
   51   728   614   458   428   425   421   248   218   158   715   378   701   671   498
LIST OF PI'S    ROOT X'S
    1     8   256
    2     1     4
    3   528   128
    4   264     2
    5   258     8
    6     5    16
    7   656     1
    8   644     2
    9   548   128
   10   548    16
   11   521     4
   12   266     1
   13   164   512
   14   145   512
   15   145   256
   16    11   256
   17   676     1
   18   657    64
   19   657     2
   20   624     2
   21   564     2
   22   525    16
   23   269     2
   24   267     4
   25   721     4
```

```
 26   677      64
 27   677       2
 28   654      64
 29   647      64
 30   647      32
 31   626       8
 32   626       1
 33   617     128
 34   555       2
 35   566       1
 36   369       2
 37   364       2
 38   241       4
 39   171       4
 40   167     512
 41   167       8
 42   155     256
 43   151      64
 44   121       4
 45   748      16
 46   636     128
 47   636       1
 48   629       8
 49   627     128
 50   571       4
 51   567       8
 52   467       8
 53   411      64
 54   411       4
 55   371       4
 56   366       1
 57   244     256
 58   172     512
 59   765       2
 60   755       8
 61   637       2
 62   557      64
 63   537       4
 64   766       1
 65   751      16
 66   639     128
 67   555     512
 68   555     256
 69   551      16       4
 70   211     256       4
       1       8
       1     256
       1       4
```

ESSENTIAL PRIME IMPLICANTS

```
  3:   528   128
  5:   258     8
  6:     5    16
  8:   644    32
 11:   521     4
 13:   164   512
 15:   145   256
 16:    11   256
 19:   657     2
 20:   624     2
 22:   525    16
 23:   269     2
 25:   721     4
 26:   677    64
 28:   654    64
 29:   647    64
 31:   626     8
 33:   617   128
 34:   569     2
 36:   369     2
 37:   364     2
 38:   241     4
 39:   171     4
 42:   155   256
 43:   151    64
 44:   121     4
 45:   748    16
 48:   629     8
 49:   627   128
 54:   411     4
 55:   371     4
 56:   366     1
 57:   245   256
 60:   758     8
 63:   251     4
 65:   751    16
 68:   255   256
 69:   451    16     4
 70:   211   256     4
       2  ESSENTIAL PRIME IMPLICANTS
 10:   548    16
 40:   167   512
 62:   575    64
       3  ESSENTIAL PRIME IMPLICANTS
 35:   566     1
      OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY '||'
 46,52,                                    ||          47,52,
```

46,53,
DISK RECORDS WRITEN= 192

11 47,53,

CASE # 2

LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH

2556	1	32	1	2	1	576	2	544	2	272	2	144	2	72	2	9	2	6	2
7669	3	672	3	656	3	644	3	578	3	577	3	548	3	545	3	532	3	529	3
5522	3	352	3	322	3	266	3	262	3	259	3	148	3	145	3	22	3	736	4
6766	4	652	4	579	4	562	4	547	4	526	4	464	4	424	4	418	4	417	4
3566	4	344	4	329	4	326	4	269	4	232	4	202	4	147	4	752	5	744	5
7222	5	659	5	647	5	632	5	626	5	602	5	572	5	569	5	566	5	539	5
4558	5	457	5	454	5	434	5	428	5	425	5	376	5	372	5	369	5	362	5
3002	6	248	6	244	6	236	6	229	6	756	6	748	6	745	6	739	6	732	6
7229	6	726	6	679	6	636	6	629	6	622	6	474	6	467	6	455	6	437	6
4277	6	414	6	366	6	359	6	347	6	252	6	245	6	762	7	747	7	702	7
5002	7	478	7	477	7	475	7	415	7	379	7	247	7	239	7	766	8	759	8
6398	8																		

OF BITS=10 # OF MINTERMS= 111 LARGEST MINTERM= 769

LIST OF ESSENTIAL MINTERMS

72 9 769 532 529 256 259 464 269 602 569 539 369 62 732 729

629 622 437 359 502 477 239 639

LIST OF PI'S ROOT X'S

1	256	16
2	32	512
3	2	4
4	544	1
5	144	512
6	144	4
7	144	1
8	6	256
9	6	16
10	672	64
11	644	32
12	644	8
13	545	2
14	522	4
15	352	4
16	322	4
17	262	64
18	145	2
19	736	16
20	736	8
21	562	64
22	562	4
23	424	4
24	424	1
25	418	16
26	417	8
27	356	16
28	344	32
29	329	128

| | | |

```
                                                
```

```
      CASE #  3
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  256 1   576 2   544 2   528 2   514 2   258 2   257 2    33 2     3 2   656 3
  578 3   577 3   548 3   545 3   529 3   524 3   518 3   517 3   515 3   273 3
  259 3   133 3    13 3   673 4   657 4   643 4   579 4   564 4   547 4   537 4
  534 4   533 4   526 4   525 4   356 4   354 4   163 4   154 4    43 4   713 5
  659 5   654 5   613 5   569 5   566 5   539 5   527 5   458 5   457 5   454 5
  433 5   403 5   357 5   313 5   244 5   173 5   158 5   157 5   155 5   143 5
  103 5   756 6   754 6   633 6   603 6   574 6   567 6   473 6   455 6   413 6
  363 6   359 6   343 6   303 6   245 6    63 6   758 7   757 7   755 7   743 7
  733 7   575 7   463 7   254 7   763 8   759 8   703 8   255 8
# OF BITS=10          # OF MINTERMS=  88          LARGEST MINTERM= 763
LIST OF ESSENTIAL MINTERMS
  133  354  163   43  713  613  458  433  403  313  173  143  363  343  303  733
LIST OF PIS   ROOT XNS
   1:  544     4
   2:  544     1
   3:  514     4
   4:  257    16
   5:   33   512
   6:    3   512
   7:    3   256
   8:  548    16
   9:  545   128
  10:  545     2
  11:  529     8
  12:  529     4
  13:  518    16
  14:  518     8
  15:  517    16
  16:  517     8
  17:  515   128
  18:  515    32
  19:   13   512
  20:  657     2
  21:  643    16
  22:  564     2
  23:  537    32
  24:  537     2
  25:  534    32
  26:  526   128
  27:  356     1
  28:  154     4
  29:  154     1
  30:  569    64
  31:  539    64
  32:  457    16
  33:  454     1
```

ESSENTIAL PRIME IMPLICANTS

ESSENTIAL PRIME IMPLICANTS

```
   6:     3   512
   8:   548    16
  13:   518    16
  15:   517    16
  18:   515    32
  21:   643    16
      OUTPUT FROM PETRICK (USE ALL PINS OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY '||'
   3,11,                                          ||          46,11,
   3,23,                                          ||          46,23,
   3,24,                                          ||          46,24,
DISK RECORDS WRITEN=     172
```

```
      CASE #   4
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  544  2   514  2   548  3   545  3   518  3   517  3   515  3   547  4   424  4   154  4
  458  5   457  5   454  5   428  5   425  5   248  5   244  5   218  5   217  5   214  5
  158  5   157  5   155  5   124  5   455  6   427  6   245  6   215  6   125  6   247  7
  127  7
# OF BITS=10        # OF MINTERMS=  31        LARGEST MINTERM=1548
LIST OF ESSENTIAL MINTERMS
  517   458   457   248   217   157
LIST OF PI'S    ROOT X'S
    1  5544      4
    2  5544      1
    3  5514      4
    4  5514      1
    5  5545      2
    6  5515     32
    7  4424      4
    8  4424      1
    9  1554     64
   10  1554      4
   11  1554      1
   12  4554      1
   13  4225      2
   14  2444      1
   15  2214      1
   16  1224      1
   17  2455      2
   18  2155     32
   19  1255      2
      PRIME IMPLICANT TABLE
        1 ESSENTIAL PRIME IMPLICANTS
    1  5544      4
    3  5514      4
    7  4424      4
    9  1554     64
   10  1554      4
   11  1554      1
   12  4554      1
   13  4225      2
   14  2444      1
   15  2214      1
   16  1224      1
   19  1255      2
        2 ESSENTIAL PRIME IMPLICANTS
    5  5545      2
    6  5515     32
    OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY ' | '
```

17,
DISK RECORDS WRITEN= 50 11 18,

```
     CASE #   5
LIST OF MINTERMS INCLUDING DONT CARES AND WEIGHTS OF EACH
 576 2   264 2   656 3   578 3   577 3   324 3   266 3   657 4   624 4   579 4
 564 4   537 4   534 4   356 4   354 4   329 4   326 4   269 4   267 4   728 5
 724 5   659 5   654 5   626 5   569 5   566 5   539 5   357 5   327 5   756 6
 754 6   729 6   726 6   725 6   629 6   627 6   574 6   567 6   359 6   758 7
 757 7   755 7   727 7   575 7   759 8
# OF BITS=10          # OF MINTERMS=  45          LARGEST MINTERM= 759
LIST OF ESSENTIAL MINTERMS
  354   329   269   654
LIST OF PI'S    ROOT X'S
   1:  264      2
   2:  656      1
   3:  324     32
   4:  324      2
   5:  266      1
   6:  657      2
   7:  624      2
   8:  564      2
   9:  537     32
  10:  537      2
  11:  534     32
  12:  356      1
  13:  326      1
  14:  728      1
  15:  357      2
  16:  327     32
  17:  629    128
  18:  576      2    1
  19:  626    128    1
  20:  566      8    1
  21:  754      4    1
  22:  724     32    2    1
       1 ESSENTIAL PRIME IMPLICANTS
   1:  264      2
   2:  656      1
   5:  266      1
   6:  657      2
   7:  624      2
   8:  564      2
   9:  537     32
  10:  537      2
  11:  534     32
  14:  728      1
  17:  629    128
  18:  576      2    1
  19:  626    128    1
  20:  566      8    1
```

```
 22:  724   32   2   1
      OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY '||'
   3,  4, 12, 13, 15,                                  ||          4,  3, 12, 13, 15,
   3, 12,  4, 13, 15,                                  ||          4, 12, 13, 15,
   3, 13, 12, 15,                                      ||          4,  3, 13, 12, 15,
   3, 12, 13, 15,                                      ||          4, 12, 13, 15,
   3,  4, 15, 13,                                      ||          4,  3, 15, 13,
   3, 12,  4, 15, 13,                                  ||          4, 12, 15, 13,
   3, 13, 15,                                          ||          4,  3, 13, 15,
   3, 12, 13, 15,                                      ||          4, 12, 13, 15,
   3,  4, 12, 16, 15,                                  ||          4,  3, 12, 16, 15,
   3, 12,  4, 16, 15,                                  ||          4, 12, 16, 15,
   3, 13, 12, 16, 15,                                  ||          4,  3, 13, 12, 16, 15,
   3, 12, 13, 16, 15,                                  ||          4, 12, 13, 16, 15,
   3,  4, 15, 16,                                      ||          4,  3, 15, 16,
   3, 12,  4, 15, 16,                                  ||          4, 12, 15, 16,
   3, 13, 15, 16,                                      ||          4,  3, 13, 15, 16,
   3, 12, 13, 15, 16,                                  ||          4, 12, 13, 15, 16,
   3,  4, 12, 13, 16,                                  ||          4,  3, 12, 13, 16,
   3, 12,  4, 13, 16,                                  ||          4, 12, 13, 16,
   3, 13, 12, 16,                                      ||          4,  3, 13, 12, 16,
   3, 12, 13, 16,                                      ||          4, 12, 13, 16,
   3,  4, 15, 13, 16,                                  ||          4,  3, 15, 13, 16,
   3, 12,  4, 15, 13, 16,                              ||          4, 12, 15, 13, 16,
   3, 13, 15, 16,                                      ||          4,  3, 13, 15, 16,
   3, 12, 13, 15, 16,                                  ||          4, 12, 13, 15, 16,
   3,  4, 12, 16,                                      ||          4,  3, 12, 16,
   3, 12,  4, 16,                                      ||          4, 12, 16,
   3, 13, 12, 16,                                      ||          4,  3, 13, 12, 16,
   3, 12, 13, 16,                                      ||          4, 12, 13, 16,
   3,  4, 15, 16,                                      ||          4,  3, 15, 16,
   3, 12,  4, 15, 16,                                  ||          4, 12, 15, 16,
   3, 13, 15, 16,                                      ||          4,  3, 13, 15, 16,
   3, 12, 13, 15, 16,                                  ||          4, 12, 13, 15, 16,
DISK RECORDS WRITEN=      99
```

```
      CASE #   6
 LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  644 3   164 3   137 3   134 3   464 4   344 4   178 4   177 4   744 5   714 5
  647 5   617 5   614 5   434 5   314 5   244 5   236 5   234 5   174 5   167 5
  748 6   745 6   718 6   717 6   715 6   474 6   467 6   437 6   347 6   317 6
  245 6   237 6   175 6   747 7   478 7   477 7   475 7   239 7
# OF BITS=10          # OF MINTERMS=  38          LARGEST MINTERM= 748
 LIST OF ESSENTIAL MINTERMS
   644   164   137   134   464   344   177   647   614   314   234   717   437   317   477
 LIST OF PI'S    ROOT X'S
    1:  178    256
    2:  744      4
    3:  744      1
    4:  714      4
    5:  714      1
    6:  617    128
    7:  244      1
    8:  236    512
    9:  236      1
   10:  174      1
   11:  167      8
   12:  745      2
   13:  715     32
   14:  474      4
   15:  474      1
   16:  467      8
   17:  347    128
   18:  237      2
   19:  175     64
        1 ESSENTIAL PRIME IMPLICANTS
    1:  178    256
    4:  714      4
    6:  617    128
    7:  244      1
   10:  174      1
   11:  167      8
   14:  474      4
   16:  467      8
   17:  347    128
        2 ESSENTIAL PRIME IMPLICANTS
    2:  744      4
   13:  715     32
   18:  237      2
      OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY '||'
    8,                                  ||          9,
DISK RECORDS WRITEN=     57
```

```
      CASE #  7
LIST OF MINTERMS INCLUDING DONIT CARES AND WEIGHTS OF EACH
 576 2  769 3  578 3  577 3  736 4  676 4  674 4  579 4  178 4  177 4
 744 5  737 5  728 5  724 5  714 5  677 5  376 5  174 5  756 6  754 6
 748 6  745 6  739 6  729 6  726 6  725 6  718 6  717 6  715 6  679 6
 574 6  474 6  378 6  377 6  374 6  175 6  764 7  758 7  757 7  755 7
 747 7  734 7  727 7  575 7  478 7  477 7  475 7  379 7  375 7  766 8
 759 8  767 9
# OF BITS=10        # OF MINTERMS=  52        LARGEST MINTERM=769
LIST OF ESSENTIAL MINTERMS
 769  674  178  177  717  477
LIST OF PIS   ROOT X'S
  1: 676      1
  2: 744      4
  3: 728      1
  4: 714      4
  5: 714      1
  6: 677      2
  7: 174      1
  8: 748     16
  9: 739     16
 10: 718     16
 11: 715     32
 12: 574      1
 13: 474      4
 14: 474      1
 15: 374      1
 16: 576      2   1
 17: 736      8   1
 18: 737      8   2
 19: 376      2   1
 20: 756      8   2
 21: 754      4   1
 22: 726     32   8
 23: 758      8   1
 24: 724     32   2   1
      1 ESSENTIAL PRIME IMPLICANTS
  1: 676      1
  3: 728      1
  6: 677      2
  7: 174      1
 12: 574      1
 13: 474      4
 14: 474      1
 15: 374      1
 16: 576      2   1
 17: 736      8   1
 19: 376      2   1
```

```
 21:  754     4    1
 23:  758     8    1
 24:  724    32    2    1
      2 ESSENTIAL PRIME IMPLICANTS
  8:  748    16
 10:  718    16
 18:  737     8    2
ALL 3 PRIME IMPLICANTS ARE ESSENTIAL
DISK RECORDS WRITTEN=    119
```

```
      CASE #  8
LIST OF MINTERMS INCLUDING DONIT CARES AND WEIGHTS OF EACH
 640 2  672 3  656 3  644 3  676 4  674 4  673 4  652 4  650 4  648 4
 624 4  610 4  600 4  654 5  632 5  626 5  620 5  614 5  613 5  602 5
 670 6  636 6  634 6  633 6  630 6  622 6  603 6  671 7
# OF BITS=10      # OF MINTERMS=  28       LARGEST MINTERM= 676
LIST OF ESSENTIAL MINTERMS
  643  613
LIST OF PI'S    ROOT X'S
   1: 640    16
   2: 672     2
   3: 672     1
   4: 644     8
   5: 652     2
   6: 650     4
   7: 654    16
   8: 632     4
   9: 632     1
  10: 620    16
  11: 620     2
  12: 614     8
  13: 602     1
  14: 670     1
  15: 640    32    4
  16: 624     8    2
  17: 610    16    4
  18: 600    32    2
      1  ESSENTIAL PRIME IMPLICANTS
   1: 640    16
   2: 672     2
   3: 672     1
   6: 650     4
   9: 632     1
  13: 602     1
  14: 670     1
  15: 640    32    4
  16: 624     8    2
  17: 610    16    4
  18: 600    32    2
      2  ESSENTIAL PRIME IMPLICANTS
  10: 620    16
  11: 620     2
      OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY ' | | '
   4,                                          | |          5,
DISK RECORDS WRITEN=     61
```

```
      CASE #  9
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  320 2   352 3   324 3   322 3    52 3   360 4   356 4   354 4   344 4   340 4
  326 4   300 4   376 5   372 5   370 5   364 5   362 5   314 5   313 5   310 5
  302 5   374 6   371 6   366 6   363 6   350 6   343 6   303 6   375 7
# OF BITS=9          # OF MINTERMS=  29          LARGEST MINTERM= 376
LIST OF ESSENTIAL MINTERMS
    52   314   313   350
LIST OF PI'S    ROOT X'S
   1:  360     16
   2:  354     16
   3:  344     32
   4:  372      2
   5:  362      1
   6:  310     64
   7:  302      1
   8:  343     32
   9:  320     32     4
  10:  320     32     2
  11:  320      4     2
  12:  352      8     4
  13:  352      8     2
  14:  324     32    16
  15:  360      4     2
  16:  300     64     2
  17:  370      4     1
       1 ESSENTIAL PRIME IMPLICANTS
   3:  344     32
   5:  362      1
   6:  310     64
   7:  302      1
   8:  343     32
  11:  320      4     2
  14:  324     32    16
  16:  300     64     2
  17:  370      4     1
ALL 2 PRIME IMPLICANTS ARE ESSENTIAL
DISK RECORDS WRITEN=      72
```

```
      CASE # 10
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
 160 2  144 2  130 2  164 3  145 3  140 3  134 3  133 3  100 3  170 4
 163 4  154 4  150 4  141 4  120 4  101 4  174 5  173 5  171 5  155 5
 151 5  143 5  124 5  121 5  103 5  175 6  125 6
# OF BITS= 8        # OF MINTERMS= 27        LARGEST MINTERM= 175
LIST OF ESSENTIAL MINTERMS
          NONE
LIST OF PI'S   ROOT X'S
   1:  160      4
   2:  144      1
   3:  130      4
   4:  140      1
   5:  134     16
   6:  133      8
   7:  100      1
   8:  163      8
   9:  154      1
  10:  150      1
  11:  101      2
  12:  170      4    1
  13:  141     32    2
  14:  120      4    1
      1 ESSENTIAL PRIME IMPLICANTS
   1:  160      4
   2:  144      1
   3:  130      4
   4:  140      1
   6:  133      8
   7:  100      1
   8:  163      8
   9:  154      1
  10:  150      1
  11:  101      2
  12:  170      4    1
  13:  141     32    2
  14:  120      4    1
      ALL MINTERMS ARE COVERED BY  1 ORDER ESSENTIAL PRIME IMPLICANTS OR MINTERMS
DISK RECORDS WRITEN=      53
```

```
      CASE # 11
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
 400 3  464 4  450 4  424 4  420 4  401 4  460 5  454 5  451 5  434 5
 433 5  425 5  421 5  410 5  403 5  474 6  473 6  470 6  455 6  430 6
 414 6  413 6  411 6  475 7  471 7  463 7  415 7
# OF BITS= 9        # OF MINTERMS= 27        LARGEST MINTERM= 475
LIST OF ESSENTIAL MINTERMS
 460  434  430
LIST OF PI'S   ROOT X'S
  1:  400    64
  2:  400     1
  3:  424     1
  4:  420     1
  5:  401    32
  6:  401     2
  7:  403     8
  8:  473     2
  9:  455     8
 10:  413     2
 11:  450     4    1
 12:  454    16    1
 13:  410    64    1
 14:  410     4    1
     1 ESSENTIAL PRIME IMPLICANTS
  1:  400    64
  3:  424     1
  4:  420     1
  5:  401    32
  8:  473     2
  9:  455     8
 10:  413     2
 11:  450     4    1
 12:  454    16    1
 13:  410    64    1
 14:  410     4    1
     OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SERARATED BY '||'.
  6,                                          ||          7,
DISK RECORDS WRITEN=     55
```

```
      CASE # 12
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
 576 2  544 2  520 2  514 2  560 3  545 3  532 3  530 3  524 3  522 3
 521 3  515 3  564 4  562 4  540 4  534 4  533 4  526 4  525 4  572 5
 570 5  566 5  541 5  574 6  571 6  500 6  575 7  502 7  501 7  510 8
 511 9
# OF BITS=10        # OF MINTERMS=  31       LARGEST MINTERM= 576
LIST OF ESSENTIAL MINTERMS
 576
LIST OF PI'S   ROOT X'S
  1: 544    16
  2: 544     1
  3: 514    16
  4: 514     8
  5: 514     1
  6: 500     2
  7: 500     1
  8: 502     8
  9: 510     1
 10: 520     4    2
 11: 520     4    1
 12: 560     4    2
 13: 532    32    8
 14: 532    32    2
 15: 532     8    1
 16: 530    32    4
 17: 524    16    1
 18: 564     8    2
 19: 562     8    4
 20: 570     4    1
      1 ESSENTIAL PRIME IMPLICANTS
  2: 544     1
  5: 514     1
  7: 500     1
  9: 510     1
 10: 520     4    2
 11: 520     4    1
 15: 532     8    1
 20: 570     4    1
      2 ESSENTIAL PRIME IMPLICANTS
 12: 560     4    2
 16: 530    32    4
 18: 564     8    2
      OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY '||'
  6,                                          ||          8,
DISK RECORDS WRITEN=     82
```

```
      CASE # 13
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  256 1   272 2   264 2   260 2   273 3   266 3   262 3   200 3   270 4   240 4
  232 4   210 4   202 4   201 4   271 5   244 5   241 5   236 5   234 5   230 5
  214 5   211 5   252 6   250 6   245 6   215 6   254 7   251 7   255 8
# OF BITS= 9        # OF MINTERMS=  29        LARGEST MINTERM= 273
LIST OF ESSENTIAL MINTERMS
  230
LIST OF PI'S   ROOT X'S
   1:  256    16
   2:  256     8
   3:  256     4
   4:  272     1
   5:  264     2
   6:  260     2
   7:  266     4
   8:  262     8
   9:  200     1
  10:  270     1
  11:  232     4
  12:  244     8
  13:  236    16
  14:  234    16
  15:  252     2
  16:  200    32    2
  17:  240     4    1
  18:  210     4    1
  19:  250     4    1
      1 ESSENTIAL PRIME IMPLICANTS
   4:  272     1
   9:  200     1
  10:  270     1
  16:  200    32    2
  17:  240     4    1
  18:  210     4    1
  19:  250     4    1
      2 ESSENTIAL PRIME IMPLICANTS
   5:  264     2
   6:  260     2
  13:  236    16
      OUTPUT FROM PETRICK (USE ALL PI'S OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY '||'
   2,                                    ||          3,
DISK RECORDS WRITEN=      64
```

```
      CASE # 14
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  736 4   720 4   752 5   744 5   740 5   724 5   722 5   721 5   714 5   713 5
  710 5   760 6   756 6   754 6   745 6   741 6   732 6   730 6   726 6   725 6
  715 6   711 6   700 6   764 7   762 7   755 7   750 7   743 7   734 7   733 7
  702 7   701 7   766 8   763 8   751 8   703 8
# OF BITS=10          # OF MINTERMS=  36        LARGEST MINTERM= 766
LIST OF ESSENTIAL MINTERMS
        NONE
LIST OF PI'S    ROOT X'S
   1:  744      1
   2:  740      1
   3:  714     16
   4:  714      1
   5:  713     32
   6:  713      2
   7:  710     16
   8:  710      1
   9:  741      2
  10:  711     32
  11:  750     16
  12:  750      1
  13:  743      8
  14:  736     16     8
  15:  736     16     4
  16:  720     32     4
  17:  720     32     2
  18:  720      4     2
  19:  720      4     1
  20:  752      8     4
  21:  752      8     2
  22:  724     32     8
  23:  724      8     2
  24:  724      8     1
  25:  722     32     8
  26:  722      8     4
  27:  760      4     2
  28:  754      8     1
  29:  732     32     2
  30:  730     32     4
  31:  700     64     2
  32:  700      2     1
                          0 1
     1 ESSENTIAL PRIME IMPLICANTS
  19:  720      4     1
  24:  724      8     1
  28:  754      8     1
  32:  700      2     1
```

```
     OUTPUT FROM PETRICK (USE ALL PIS OF ONE GROUP)
MORE THAN ONE GROUP PER LINE ARE SEPARATED BY !!!
 14, 1, 2,17, 3, 5, 7,15, 4, 8,20,11, 9,26,12,
 15,14, 1, 2,17, 3, 5, 7, 4, 8,20,11, 9,26,12,
 14,15, 1, 2,17, 3, 5, 7, 4, 8,20,11, 9,26,12,
 15, 1, 2,17, 3, 5, 7,14, 4, 8,20,11, 9,26,12,
 14,17, 1, 2, 3, 5, 7,15, 4, 8,20,11, 9,26,12,
 15,17, 1, 2, 3, 5, 7,14, 4, 8,20,11, 9,26,12,
 14,20, 1, 2,17, 3, 5, 7,15, 4, 8,11, 9,26,12,
 15,20, 1, 2,17, 3, 5, 7,14, 4, 8,11, 9,26,12,
 14, 2,17, 3, 5, 7,15, 1, 4, 8,20,11, 9,26,12,
 15,14, 2,17, 3, 5, 7, 1, 4, 8,20,11, 9,26,12,
 14,15, 2,17, 3, 5, 7, 1, 4, 8,20,11, 9,26,12,
 15,14, 2,17, 3, 5, 7, 1, 4, 8,20,11, 9,26,12,
 14,17, 2, 3, 5, 7,15, 1, 4, 8,20,11, 9,26,12,
 15,17,14, 2, 3, 5, 7, 1, 4, 8,20,11, 9,26,12,
 14,20, 2,17, 3, 5, 7,15, 1, 4, 8,11, 9,26,12,
 15,20,14, 2,17, 3, 5, 7, 1, 4, 8,11, 9,26,12,
 14, 1,15,17, 3, 5, 7, 2, 4, 8,20,11, 9,26,12,
 15,14, 1,17, 3, 5, 7, 2, 4, 8,20,11, 9,26,12,
 14,15, 1,17, 3, 5, 7, 2, 4, 8,20,11, 9,26,12,
 15, 1,17, 3, 5, 7,14, 2, 4, 8,20,11, 9,26,12,
 14,17, 1,15, 3, 5, 7, 2, 4, 8,20,11, 9,26,12,
 15,17, 1, 3, 5, 7,14, 2, 4, 8,20,11, 9,26,12,
 14,20, 1,15,17, 3, 5, 7, 2, 4, 8,11, 9,26,12,
 15,20, 1,17, 3, 5, 7,14, 2, 4, 8,11, 9,26,12,
 14,15,17, 3, 5, 7, 1, 2, 4, 8,20,11, 9,26,12,
 15,14,17, 3, 5, 7, 1, 2, 4, 8,20,11, 9,26,12,
 14,15,17, 3, 5, 7, 1, 2, 4, 8,20,11, 9,26,12,
 15,14,17, 3, 5, 7, 1, 2, 4, 8,20,11, 9,26,12,
 14,17,15, 3, 5, 7, 1, 2, 4, 8,20,11, 9,26,12,
 15,17,14, 3, 5, 7, 1, 2, 4, 8,20,11, 9,26,12,
 14,20,15,17, 3, 5, 7, 1, 2, 4, 8,11, 9,26,12,
 15,20,14,17, 3, 5, 7, 1, 2, 4, 8,11, 9,26,12,
 14, 1, 2,26, 3, 5, 7,15, 4, 8,20,11, 9,12,      15,14, 1, 2,26, 3, 5, 7, 4, 8,2
0,11, 9,12,
 14,15, 1, 2,26, 3, 5, 7, 4, 8,20,11, 9,12,      15, 1, 2,26, 3, 5, 7,14, 4, 8,2
0,11, 9,12,
 14,17, 1, 2,26, 3, 5, 7,15, 4, 8,20,11, 9,12,
 15,17, 1, 2,26, 3, 5, 7,14, 4, 8,20,11, 9,12,
 14,20, 1, 2,26, 3, 5, 7,15, 4, 8,11, 9,12,      15,20, 1, 2,26, 3, 5, 7,14, 4,
8,11, 9,12,
 14, 2,26, 3, 5, 7,15, 1, 4, 8,20,11, 9,12,      15,14, 2,26, 3, 5, 7, 1, 4, 8,2
0,11, 9,12,
 14,15, 2,26, 3, 5, 7, 1, 4, 8,20,11, 9,12,      15,14, 2,26, 3, 5, 7, 1, 4, 8,2
0,11, 9,12,
 14,17, 2,26, 3, 5, 7,15, 1, 4, 8,20,11, 9,12,
 15,17,14, 2,26, 3, 5, 7, 1, 4, 8,20,11, 9,12,
```

14,20, 2,26, 3, 5, 7,15, 1, 4, 8,11, 9,12,
8,11, 9,12,
14, 1,15,26, 3, 5, 7, 2, 4, 8,20,11, 9,12,
0,11, 9,12,
14,15, 1,26, 3, 5, 7, 2, 4, 8,20,11, 9,12,
0,11, 9,12,
14,17, 1,15,26, 3, 5, 7, 2, 4, 8,20,11, 9,12,
15,17, 1,26, 3, 5, 7,14, 2, 4, 8,20,11, 9,12,
14,20, 1,15,26, 3, 5, 7, 2, 4, 8,11, 9,12,
8,11, 9,12,
14,15,26, 3, 5, 7, 1, 2, 4, 8,20,11, 9,12,
0,11, 9,12,
14,15,26, 3, 5, 7, 1, 2, 4, 8,20,11, 9,12,
0,11, 9,12,
14,17,15,26, 3, 5, 7, 1, 2, 4, 8,20,11, 9,12,
15,17,14,26, 3, 5, 7, 1, 2, 4, 8,20,11, 9,12,
14,20,15,26, 3, 5, 7, 1, 2, 4, 8,11, 9,12,
8,11, 9,12,
14, 1, 2,17, 4, 5, 7,15, 3, 8,20,11, 9,26,12,
15,14, 1, 2,17, 4, 5, 7, 3, 8,20,11, 9,26,12,
14,15, 1, 2,17, 4, 5, 7, 3, 8,20,11, 9,26,12,
15, 1, 2,17, 4, 5, 7,14, 3, 8,20,11, 9,26,12,
14,17, 1, 2, 4, 5, 7,15, 3, 8,20,11, 9,26,12,
15,17, 1, 2, 4, 5, 7,14, 3, 8,20,11, 9,26,12,
14,20, 1, 2,17, 4, 5, 7,15, 3, 8,11, 9,26,12,
15,20, 1, 2,17, 4, 5, 7,14, 3, 8,11, 9,26,12,
14, 2,17, 4, 5, 7,15, 1, 3, 8,20,11, 9,26,12,
15,14, 2,17, 4, 5, 7, 1, 3, 8,20,11, 9,26,12,
14,15, 2,17, 4, 5, 7, 1, 3, 8,20,11, 9,26,12,
15,14, 2,17, 4, 5, 7, 1, 3, 8,20,11, 9,26,12,
14,17, 2, 4, 5, 7,15, 1, 3, 8,20,11, 9,26,12,
15,17,14, 2, 4, 5, 7, 1, 3, 8,20,11, 9,26,12,
14,20, 2,17, 4, 5, 7,15, 1, 3, 8,11, 9,26,12,
15,20,14, 2,17, 4, 5, 7, 1, 3, 8,11, 9,26,12,
14, 1,15,17, 4, 5, 7, 2, 3, 8,20,11, 9,26,12,
15,14, 1,17, 4, 5, 7, 2, 3, 8,20,11, 9,26,12,
14,15, 1,17, 4, 5, 7, 2, 3, 8,20,11, 9,26,12,
15, 1,17, 4, 5, 7,14, 2, 3, 8,20,11, 9,26,12,
14,17, 1,15, 4, 5, 7, 2, 3, 8,20,11, 9,26,12,
15,17, 1, 4, 5, 7,14, 2, 3, 8,20,11, 9,26,12,
14,20, 1,15,17, 4, 5, 7, 2, 3, 8,11, 9,26,12,
15,20, 1,17, 4, 5, 7,14, 2, 3, 8,11, 9,26,12,
14,15,17, 4, 5, 7, 1, 2, 3, 8,20,11, 9,26,12,
15,14,17, 4, 5, 7, 1, 2, 3, 8,20,11, 9,26,12,
14,15,17, 4, 5, 7, 1, 2, 3, 8,20,11, 9,26,12,
15,14,17, 4, 5, 7, 1, 2, 3, 8,20,11, 9,26,12,
14,17,15, 4, 5, 7, 1, 2, 3, 8,20,11, 9,26,12,
15,17,14, 4, 5, 7, 1, 2, 3, 8,20,11, 9,26,12,

15,20,14, 2,26, 3, 5, 7, 1, 4,
15,14, 1,26, 3, 5, 7, 2, 4, 8,2
15, 1,26, 3, 5, 7,14, 2, 4, 8,2
15,20, 1,26, 3, 5, 7,14, 2, 4,
15,14,26, 3, 5, 7, 1, 2, 4, 8,2
15,14,26, 3, 5, 7, 1, 2, 4, 8,2
15,20,14,26, 3, 5, 7, 1, 2, 4,

```
 14, 20, 15, 17, 4, 5, 7, 1, 2, 3, 8, 11, 9, 26, 12,
 15, 20, 14, 17, 4, 5, 7, 1, 2, 3, 8, 11, 9, 26, 12,
 14, 1, 2, 26, 4, 5, 7, 15, 3, 8, 20, 11, 9, 12,      15, 14, 1, 2, 26, 4, 5, 7, 3, 8, 2
0, 11, 9, 12,
 14, 15, 1, 2, 26, 4, 5, 7, 3, 8, 20, 11, 9, 12,      15, 1, 2, 26, 4, 5, 7, 14, 3, 8, 2
0, 11, 9, 12,
 14, 17, 1, 2, 26, 4, 5, 7, 15, 3, 8, 20, 11, 9, 12,
 15, 17, 1, 2, 26, 4, 5, 7, 14, 3, 8, 20, 11, 9, 12,
 14, 20, 1, 2, 26, 4, 5, 7, 15, 3, 8, 11, 9, 12,      15, 20, 1, 2, 26, 4, 5, 7, 14, 3,
8, 11, 9, 12,
 14, 2, 26, 4, 5, 7, 15, 1, 3, 8, 20, 11, 9, 12,      15, 14, 2, 26, 4, 5, 7, 1, 3, 8, 2
0, 11, 9, 12,
 14, 15, 2, 26, 4, 5, 7, 1, 3, 8, 20, 11, 9, 12,      15, 14, 2, 26, 4, 5, 7, 1, 3, 8, 2
0, 11, 9, 12,
 14, 17, 2, 26, 4, 5, 7, 15, 1, 3, 8, 20, 11, 9, 12,
 15, 17, 14, 2, 26, 4, 5, 7, 1, 3, 8, 20, 11, 9, 12,
 14, 20, 2, 26, 4, 5, 7, 15, 1, 3, 8, 11, 9, 12,      15, 20, 14, 2, 26, 4, 5, 7, 1, 3,
8, 11, 9, 12,
 14, 1, 15, 26, 4, 5, 7, 2, 3, 8, 20, 11, 9, 12,      15, 14, 1, 26, 4, 5, 7, 2, 3, 8, 2
0, 11, 9, 12,
 14, 15, 1, 26, 4, 5, 7, 2, 3, 8, 20, 11, 9, 12,      15, 1, 26, 4, 5, 7, 14, 2, 3, 8, 2
0, 11, 9, 12,
 14, 17, 1, 15, 26, 4, 5, 7, 2, 3, 8, 20, 11, 9, 12,
 15, 17, 1, 26, 4, 5, 7, 14, 2, 3, 8, 20, 11, 9, 12,
 14, 20, 1, 15, 26, 4, 5, 7, 2, 3, 8, 11, 9, 12,      15, 20, 1, 26, 4, 5, 7, 14, 2, 3,
8, 11, 9, 12,
 14, 15, 26, 4, 5, 7, 1, 2, 3, 8, 20, 11, 9, 12,      15, 14, 26, 4, 5, 7, 1, 2, 3, 8, 2
0, 11, 9, 12,
 14, 15, 26, 4, 5, 7, 1, 2, 3, 8, 20, 11, 9, 12,      15, 14, 26, 4, 5, 7, 1, 2, 3, 8, 2
0, 11, 9, 12,
DISK RECORDS WRITEN=    111
```

```
      CASE # 15
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
 260 2  160 2  560 3  262 3  562 4  360 4  163 4   60 4  460 5  362 5
  62 5  760 6  363 6   63 6  762 7  463 7  763 8
# OF BITS=10        # OF MINTERMS=  17        LARGEST MINTERM= 763
LIST OF ESSENTIAL MINTERMS
 160  163  460  463
LIST OF PI'S   ROOT X'S
   1   260      2
   2   560      2
   3   360      2
   4    60      2
   5   362      1
   6    62      1
   7   760      2
   8   762      1
     1 ESSENTIAL PRIME IMPLICANTS
   1   260      2
   2   560      2
   3   360      2
   4    60      2
   5   362      1
   6    62      1
   7   760      2
   8   762      1
     ALL MINTERMS ARE COVERED BY  1 ORDER ESSENTIAL PRIME IMPLICANTS OR MINTERMS
DISK RECORDS WRITEN=      25
```

```
      CASE # 16
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  32 1  130 2  33 2  532 3  530 3  133 3  533 4  232 4  30 4  632 5
 433 5  230 5  732 6  730 6  633 6  630 6  430 6  733 7
# OF BITS=10        # OF MINTERMS=  18        LARGEST MINTERM=733
LIST OF ESSENTIAL MINTERMS
  130  530  133  232  30  433  230  730  630  430
LIST OF PI'S   ROOT X'S
  1:  32     1
  2:  532    1
  3:  632    1
  4:  732    1
ALL 1 PRIME IMPLICANTS ARE ESSENTIAL
DISK RECORDS WRITEN=     22
```

```
      CASE # 17
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
   10 2    13 3    11 3   610 4   210 4   713 5   710 5   613 5   410 5   313 5
  310 5   211 5   711 6   413 6   411 6   510 8   511 9
# OF BITS=10          # OF MINTERMS=  17          LARGEST MINTERM= 713
LIST OF ESSENTIAL MINTERMS
    13   610   713   613   313   310   413
LIST OF PI"S    ROOT X"S
   1:   10      1
   2:  210      1
   3:  710      1
   4:  410      1
   5:  510      1
ALL 1 PRIME IMPLICANTS ARE ESSENTIAL
DISK RECORDS WRITEN=      22
```

```
      CASE # 18
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
 640 2   40 2   140 3   41 3   643 4   540 4   340 4   240 4   141 4   43 4
 740 5   541 5   241 5   143 5   741 6   343 6   743 7
# OF BITS=10          # OF MINTERMS=  17          LARGEST MINTERM=743
LIST OF ESSENTIAL MINTERMS
  640   643   340   343
LIST OF PI'S   ROOT X'S
  1:   40      1
  2:  140      1
  3:   41      2
  4:  540      1
  5:  240      1
  6:  141      2
  7:  740      1
  8:  741      2
      1 ESSENTIAL PRIME IMPLICANTS
  1:   40      1
  2:  140      1
  3:   41      2
  4:  540      1
  5:  240      1
  6:  141      2
  7:  740      1
  8:  741      2
      ALL MINTERMS ARE COVERED BY  1 ORDER ESSENTIAL PRIME IMPLICANTS OR MINTERMS
DISK RECORDS WRITEN=      25
```

```
      CASE # 19
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
 352 3   52 3   50 3  652 4  650 4  450 4  150 4   51 4  752 5  451 5
 151 5  350 6  252 6  250 6  750 7  251 7  751 8
# OF BITS=10        # OF MINTERMS=  17       LARGEST MINTERM= 752
LIST OF ESSENTIAL MINTERMS
  352   52  652  650  752  350  252
LIST OF PI'S   ROOT X'S
   1:  50     1
   2: 450     1
   3: 150     1
   4: 250     1
   5: 750     1
ALL 1 PRIME IMPLICANTS ARE ESSENTIAL
DISK RECORDS WRITEN=     22
```

```
      CASE # 20
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
  520 2   320 2    20 2   522 3   521 3   322 3    22 3    21 3   720 4   420 4
  120 4   722 5   721 5   620 5   421 5   121 5   622 6
# OF BITS=10          # OF MINTERMS=  17        LARGEST MINTERM= 722
LIST OF ESSENTIAL MINTERMS
        NONE
LIST OF PI"S   ROOT X"S
    1:  520      2
    2:  520      1
    3:  320      2
    4:   20      2
    5:   20      1
    6:  720      2
    7:  720      1
    8:  420      1
    9:  120      1
   10:  620      2
      1 ESSENTIAL PRIME IMPLICANTS
    1:  520      2
    2:  520      1
    3:  320      2
    4:   20      2
    5:   20      1
    6:  720      2
    7:  720      1
    8:  420      1
    9:  120      1
   10:  620      2
      ALL MINTERMS ARE COVERED BY  1 ORDER ESSENTIAL PRIME IMPLICANTS OR MINTERMS
DISK RECORDS WRITEN=      27
```

```
      CASE # 21
LIST OF MINTERMS INCLUDING DON'T CARES AND WEIGHTS OF EACH
 272 2   72 2  672 3  273 3   73 3   70 3  673 4  270 4  170 4   71 4
 572 5  570 5  372 5  370 5  271 5  173 5  171 5  670 6  571 6  473 6
 470 6  371 6  671 7  471 7
# OF BITS=10        # OF MINTERMS=  24       LARGEST MINTERM= 673
LIST OF ESSENTIAL MINTERMS
  572  372  173  473
LIST OF PI'S   ROOT X'S
   1  272     1
   2   72     1
   3  672     1
   4   70     1
   5  270     1
   6  170     1
   7  570     1
   8  370     1
   9  670     1
  10  470     1
ALL 1 PRIME IMPLICANTS ARE ESSENTIAL
DISK RECORDS WRITTEN=      34
```

APPENDIX C

LISTING OF THE 3-D MACHINE'S SIMULATOR

(SIM3DMIX)

A listing of the simulator program (SIM3DMIX) follows. XDS Extended FORTRAN IV (52) is used. All statements with 'S' in column 1 are XDS Sigma 7 assembly language (SYMBOL/METASYMBOL) statements. All cards with '!' in column 1 are monitor command (57) cards. The card reader is device 105 and the line printer is device 108. Devices 109, 110, and 115 are disk files.

This program can be executed on any GP computer if appropriate changes are made to:

1. address the appropriate devices,
2. replace every Extended FORTRAN IV statement by its corresponding set of standard FORTRAN IV statements.
3. replace the inline assembly language statements by either the inline assembly language statements of the computer utilized or an equivalent FORTRAN callable assembly language subroutine.

A minimum memory requirement of 22 K words of core memory (this does not include the core required by the operating system of the computer), makes it possible to simulate an 8 x 8 x 8 MC. The listed program which simulates an 16 x 16 x 16 MC requires 40 K words of core. One assumes that the GP computer utilized has a FORTRAN IV compiler.

```
!JOB 667,HANNA,3.
!LIMIT (TIME,10),(UO,200),(LO,200),(CORE,40)
!ASSIGN F:109,(FILE,SIMLOAD),(OUT),(SAVE)
!ASSIGN F:110,(FILE,SIMPROG),(OUT),(SAVE)
!ASSIGN F:115,(FILE,DATA),(OUT),(SAVE)
!ASSIGN M:BO,(FILE,TEMP)
!FORTRAN LS,S,BO
      INTEGER INST(20,200),TEMP(20),SOURCE                                    MAIN0000
      INTEGER ILABL (10),ISOURCE (10)                                         MAIN0010
      INDX=1                                                                  MAIN0020
1     READ(105,15)(INST(I,INDX),I=1,20)                                       MAIN0030
15    FORMAT(20A4)                                                            MAIN0040
      DO 16 I=1,20                                                            MAIN0050
16    TEMP(I)=INST(I,INDX)                                                    MAIN0060
      DECODE(1,17,TEMP)IFLAG                                                  MAIN0070
17    FORMAT(A1)                                                              MAIN0080
      IF(IFLAG.EQ.1HC)WRITE(109,15)(TEMP(I),I=1,20),INDX=INDX+1,GO TO 1       MAIN0090
      IF(IFLAG.NE.1HD)GO TO 18                                                MAIN0100
      ITEMP=TEMP(1)                                                           MAIN0110
S     LB,2    X'40'                                                           MAIN0120
S     STB,2    ITEMP                                                          MAIN0130
      WRITE(109,19)ITEMP,(TEMP(I),I=2,20)                                     MAIN0140
19    FORMAT(A4,19A4)                                                         MAIN0150
      INDX=INDX+1                                                             MAIN0160
      GO TO 1                                                                 MAIN0170
18    WRITE(109,2)                                                            MAIN0180
C                                                                             MAIN0190
CC   BUILD THE OVER HEAD REQUIRED                                             MAIN0200
CC   BUILD THE OVER HEAD REQUIRED                                             MAIN0210
C                                                                             MAIN0220
    2 FORMAT(6X,56HINTEGER CR(10,16,16),TEMP(20),INST(20,20C),MRC(16,16,      MAIN0230
     $16))                                                                    MAIN0240
      WRITE(109,3)                                                            MAIN0250
    3 FORMAT(6X,31HINTEGER SCOUNT,PCOUNT,TR(16,16))                           MAIN0260
      WRITE(109,4)                                                            MAIN0270
    4 FORMAT(6X,32HREAL MR1(16,16,16),MR2(16,16,16))                          MAIN0280
      WRITE(109,5)                                                            MAIN0290
    5 FORMAT(6X,26HREAL CPA(16,16),CPQ(16,16))                                MAIN0300
      WRITE(109,6)                                                            MAIN0310
    6 FORMAT(6X,20HCOMMON SCOUNT,PCOUNT)                                      MAIN0320
      WRITE(109,7)                                                            MAIN0330
    7 FORMAT(6X,8HSCOUNT=0)                                                   MAIN0340
      WRITE(109,8)                                                            MAIN0350
    8 FORMAT(6X,8HPCOUNT=0)                                                   MAIN0360
      WRITE(109,9)                                                            MAIN0370
    9 FORMAT(6X,11HDO 4 I=1,16)                                               MAIN0380
      WRITE(109,10)                                                           MAIN0390
   10 FORMAT(6X,11HDO 4 J=1,16)                                               MAIN0400
```

```
      WRITE(109,11)                                                     MAIN0410
   11 FORMAT(6X,11HDO 4 K=1,16)                                         MAIN0420
      WRITE(109,12)                                                     MAIN0430
   12 FORMAT(6X,12HMRC(I,J,K)=0/6X,13HMR1(I,J,K)=0/6X,13HMR2(I,J,R)=0/  MAIN0440
     $6X,11HCRA(J,K)=0/6X,11HCPG(J,K)=0/1H4,5X,8HCONTINUE)              MAIN0450
      WRITE(109,13)                                                     MAIN0460
   13 FORMAT(6X,11HDO 5 I=1,20/6X,12HDO 5 J=1,200/6X,11HINST(I,J)=0/1H5 MAIN0470
     $,5X,8HCONTINUE)                                                   MAIN0480
      WRITE(109,14)                                                     MAIN0490
   14 FORMAT(6X,11HDO 6 I=1,10/6X,11HDO 6 J=1,16/6X,11HDO 6 K=1,16/1H6, MAIN0500
     $5X,11HCR(I,J,K)=0)                                                MAIN0510
  113 READ(105,111)(INST(I,INDX),I=1,20)                                MAIN0520
  111 FORMAT(20A4)                                                      MAIN0530
      WRITE(108,112)(INST(I,INDX),I=1,20)                               MAIN0540
  112 FORMAT(1X,20A4)                                                   MAIN0550
      IF(INST(1,INDX).EQ.4H EOD)MINDX=INDX INDX=0 GO TO 55              MAIN0560
      INDX=INDX+1                                                       MAIN0570
      GO TO 113                                                         MAIN0580
   55 DO 200 J=1,MINDX                                                  MAIN0590
C                                                                       MAIN0600
C  BUILD A FILE OF THE MIXED MODE PROGRAM                               MAIN0610
C                                                                       MAIN0620
  200 WRITE(110,201)(INST(I,J),I=1,20)                                  MAIN0630
  201 FORMAT(20A4)                                                      MAIN0640
C                                                                       MAIN0650
C                                                                       MAIN0660
      WRITE(109,202)MINDX                                               MAIN0670
  202 FORMAT(6X,6HMINDX=,I5)                                            MAIN0680
      WRITE(109,114)                                                    MAIN0690
  114 FORMAT(6X,15HDO 13 J=1,MINDX/     ,6X,12HREAD(110,14),18H(INST(I,J)MAIN0700
     $,I=1,20)/2H14,4X,12HFORMAT(20A4))                                 MAIN0710
      WRITE(109,115)                                                    MAIN0720
  115 FORMAT(2H13,4X,31HWRITE(108,15)(INST(I,J),I=1,20)/2H15,4X,        MAIN0730
     $15HFORMAT(1X,20A4))                                               MAIN0740
C                                                                       MAIN0750
C                                                                       MAIN0760
C   BUILD FORTRAN EQUIVALENT PROGRAM                                    MAIN0770
C   SIM30 STATMENTS ARE REPLACED BY APPROPRIATE CALLS                   MAIN0780
      DO 51 I=1,MINDX                                                   MAIN0790
      DO 52 J=1,20                                                      MAIN0800
   52 TEMP(J)=INST(J,I)                                                 MAIN0810
      IF(TEMP(5).EQ.4HSTOP)LINDX=I GO TO 54                             MAIN0820
   51 CONTINUE                                                          MAIN0830
   54 DO 57 INDX=1,LINDX                                                MAIN0840
   53 DO 56 I=1,20                                                      MAIN0850
   56 TEMP(I)=INST(I,INDX)                                              MAIN0860
      DECODE(1,116,TEMP)IFLAG                                           MAIN0870
  116 FORMAT(T1,A1)                                                     MAIN0880
```

```
C                                                                                MAIN0890
CC                                                                               MAIN0900
C     SKIP THE FORTRAN DECLARATION STATMENT                                      MAIN0910
C      DECLARATION HAS BEEN TAKEN CARE OF IN THE OVERHEAD                        MAIN0920
       IF(IFLAG.EQ.1HD)GO TO 57                                                  MAIN0930
CC                                                                               MAIN0940
CC                                                                               MAIN0950
       IF(IFLAG.EQ.1HC.OR.IFLAG.EQ.1HS)WRITE(109,117)TEMP;GO TO 57               MAIN0960
  117  FORMAT(20A4)                                                              MAIN0970
       IF(IFLAG.NE.1HF)GO TO 125                                                 MAIN0980
       ITEMP=TEMP(1)                                                             MAIN0990
S      LI,2    X'40'                                                             MAIN1000
S      STB,2    ITEMP                                                            MAIN1010
       WRITE(109,118)ITEMP,(TEMP(I),I=2,20)                                      MAIN1020
  118  FORMAT(A4,19A4)                                                           MAIN1030
       GO TO 57                                                                  MAIN1040
  125  IF(IFLAG.NE.1H(.AND.IFLAG.NE.1H )GO TO 123                                MAIN1050
       IRT=IR                                                                    MAIN1060
       JRT=JR                                                                    MAIN1070
       ICT=IC                                                                    MAIN1080
       JCT=JC                                                                    MAIN1090
       DECODE(80,119,TEMP)II,JJ                                                  MAIN1100
  119  FORMAT(T37,I2,T53,I2)                                                     MAIN1110
       CALL FIELD1(IR,JR,IC,JC,TEMP)                                             MAIN1120
       DECODE(80,121,TEMP)ILABL                                                  MAIN1130
  121  FORMAT(T71,10A1)                                                          MAIN1140
       IF(IR.EQ.0.AND.JR.EQ.0.AND.IC.EQ.0.AND.JC.EQ.0)IR=IRT;JR=JRT             MAIN1150
      $IC=ICT;JC=JCT                                                             MAIN1160
       CALL SLABL(ILABL,LABEL)                                                   MAIN1170
       IF(TEMP(5).EQ.4HSTOP)GO TO 1100                                           MAIN1180
       IF(TEMP(5).EQ.4HINPT)GO TO 1200                                           MAIN1190
       IF(TEMP(5).EQ.4HOUTP)GO TO 1300                                           MAIN1200
       IF(TEMP(5).EQ.4HCTRN)GO TO 1400                                           MAIN1210
       IF(TEMP(5).EQ.4HDTRN)GO TO 1500                                           MAIN1220
       IF(TEMP(5).EQ.4HTRAN)GO TO 1600                                           MAIN1230
       IF(TEMP(5).EQ.4HBRNU)DECODE(80,126,TEMP)ISOURCE;GO TO 1700                MAIN1240
       IF(TEMP(5).EQ.4HBRNZ)DECODE(80,126,TEMP)ISOURCE;GO TO 1800                MAIN1250
       IF(TEMP(5).EQ.4HBRNF)DECODE(80,126,TEMP)ISOURCE;GO TO 1900                MAIN1260
       IF(TEMP(5).EQ.4HMAXF)GO TO 2000                                           MAIN1270
  123  IF(TEMP(5).EQ.4HSTOP)GO TO 1100                                           MAIN1280
       WRITE(108,124)TEMP                                                        MAIN1290
  124  FORMAT('0ILLEGAL 3-D INSTRUCTION'/1X,20A4)                                MAIN1300
       GO TO 9999                                                                MAIN1310
 1100  WRITE(109,1101)                                                           MAIN1320
 1101  FORMAT(4H9999,2X,14HWRITE(108,999)/3H999,3X,39HFORMAT('0STOP INSTRMAIN1330
      $UCTION ENCOUNTERED')/6X,14HWRITE(108,998)/3H998,3X,11HFORMAT('1')/MAIN1340
      $6X,9HCALL EXIT/6X,3HEND)                                                  MAIN1350
C                                                                                MAIN1360
```

```
C  BUILD DATA FILE (DATA) THROUGH F1115                                          MAIN1370
C  BUILD DATA FILE (DATA) THROUGH F1115                                          MAIN1380
C                                                                              MAIN1390
 1102 INDX=INDX+1                                                              MAIN1400
      DO 1113 I=1,20                                                           MAIN1410
1113  TEMP(I)=INST(I,INDX)                                                     MAIN1420
      DECODE(4,1103,TEMP)IFLAG                                                 MAIN1430
 1103 FORMAT(A4)                                                               MAIN1440
      IF(IFLAG.EQ.4H EOD)WRITE(115,1105) &GO TO 9999                           MAIN1450
1105  FORMAT(4H$EOD)                                                           MAIN1460
      WRITE(115,1104)TEMP                                                      MAIN1470
 1104 FORMAT(20A4)                                                             MAIN1480
C                                                                              MAIN1490
C                                                                              MAIN1500
      GO TO 1102                                                               MAIN1510
C                                                                              MAIN1520
CC    BUILD A FORTRAN EQUIVALENT PROGRAM TO THE MIXED MODE ONE IN FILE 81M     MAIN1530
C  THROUGH F1109                                                               MAIN1540
C                                                                              MAIN1550
1200  IF(LABEL.EQ.0)GO TO 1202                                                 MAIN1560
      WRITE(109,1201)LABEL,IR,JR,IC,JC,INDX                                    MAIN1570
 1201 FORMAT(I5,1X,11HCALL SINPT(,4(I2,1H,),7HINST(1,,I5,2H),,18HMR1,MR2MAIN1580
     $,MRC,CR,TR))                                                             MAIN1590
      GO TO 57                                                                 MAIN1600
1202  WRITE(109,1203)       IR,JR,IC,JC,INDX                                   MAIN1610
1203  FORMAT(5X,1X,11HCALL SINPT(,4(I2,1H,),7HINST(1,,I5,2H),,18HMR1,MR2MAIN1620
     $,MRC,CR,TR))                                                             MAIN1630
      GO TO 57                                                                 MAIN1640
1300  IF(LABEL.EQ.0)GO TO 1302                                                 MAIN1650
      WRITE(109,1301)LABEL,IR,JR,IC,JC,INDX                                    MAIN1660
 1301 FORMAT(I5,1X,11HCALL SOUTP(,4(I2,1H,),7HINST(1,,I5, 2H),,8HMR1,MR2MAIN1670
     $))                                                                       MAIN1680
      GO TO 57                                                                 MAIN1690
1302  WRITE(109,1303)       IR,JR,IC,JC,INDX                                   MAIN1700
1303  FORMAT(5X,1X,11HCALL SOUTP(,4(I2,1H,),7HINST(1,,I5, 2H),,8HMR1,MR2MAIN1710
     $))                                                                       MAIN1720
      GO TO 57                                                                 MAIN1730
1400  IF(LABEL.EQ.0)GO TO 1402                                                 MAIN1740
      WRITE(109,1401)LABEL,IR,JR,IC,JC,INDX                                    MAIN1750
1401  FORMAT(I5,1X,11HCALL SCTRN(,4(I2,1H,),7HINST(1,,I5,2H),,12HMR1,MR2MAIN1760
     $,MRC))                                                                   MAIN1770
      GO TO 57                                                                 MAIN1780
1402  WRITE(109,1403)       IR,JR,IC,JC,INDX                                   MAIN1790
1403  FORMAT(5X,1X,11HCALL SCTRN(,4(I2,1H,),7HINST(1,,I5,2H),,12HMR1,MR2MAIN1800
     $,MRC))                                                                   MAIN1810
      GO TO 57                                                                 MAIN1820
1500  IF(LABEL.EQ.0)GO TO 1502                                                 MAIN1830
      WRITE(109,1501)LABEL,IR,JR,IC,JC,INDX                                    MAIN1840
```

```
 1501  FORMAT(I5,1X,11HCALL SDTRN(,4(I2,1H),),7HINST(1,,I2,2H),,12HMR1,MR2MAIN1850
     $,MRC))                                                            MAIN1860
       GO TO 57                                                         MAIN1870
 1502  WRITE(109,1503)      IR,JR,IC,JC,INDX                            MAIN1880
 1503  FORMAT(5X,1X,11HCALL SDTRN(,4(I2,1H),),7HINST(1,,I2,2H),,12HMR1,MR2MAIN1890
     $,MRC))                                                            MAIN1900
       GO TO 57                                                         MAIN1910
 1600  IF(LABEL.EQ.0)GO TO 1602                                         MAIN1920
       WRITE(109,1601)LABEL,IR,JR,IC,JC,INDX                            MAIN1930
 1601  FORMAT(I5,1X,11HCALL STRAN(,4(I2,1H),),7HINST(1,,I5,2H),,23HMR1,MR2MAIN1940
     $,MRC,CPA,CPQ,CR))                                                 MAIN1950
       GO TO 57                                                         MAIN1960
 1602  WRITE(109,1603)      IR,JR,IC,JC,INDX                            MAIN1970
 1603  FORMAT(5X,1X,11HCALL STRAN(,4(I2,1H),),7HINST(1,,I5,2H),,23HMR1,MR2MAIN1980
     $,MRC,CPA,CPQ,CR))                                                 MAIN1990
       GO TO 57                                                         MAIN2000
 1700  CALL SLABL(ISOURCE,SOURCE)                                       MAIN2010
       IF(LABEL.EQ.0)GO TO 1702                                         MAIN2020
       WRITE(109,1701)LABEL,INDX,SOURCE                                 MAIN2030
 1701  FORMAT(I5,1X,17HCALL SBUN(INST(1,,I5,2H),,I5,2HS))               MAIN2040
       GO TO 57                                                         MAIN2050
 1702  WRITE(109,1703)      INDX,SOURCE                                 MAIN2060
 1703  FORMAT(5X,1X,17HCALL SBUN(INST(1,,I5,2H),,I5,2HS))               MAIN2070
       GO TO 57                                                         MAIN2080
 1800  CALL SLABL(ISOURCE,SOURCE)                                       MAIN2090
       IF(LABEL.EQ.0)GO TO 1802                                         MAIN2100
       WRITE(109,1801)LABEL,IR,JR,IC,JC,INDX,II,SOURCE                  MAIN2110
 1801  FORMAT(I5,1X,10HCALL SBNZ(,4(I2,1H),),8HCPA,CPQ,,7HINST(1,,I5,2H),,MAIN2120
     $I2,I5,2HS))                                                       MAIN2130
       GO TO 57                                                         MAIN2140
 1802  WRITE(109,1803)      IR,JR,IC,JC,INDX,II,SOURCE                  MAIN2150
 1803  FORMAT(5X,1X,10HCALL SBNZ(,4(I2,1H),),8HCPA,CPQ,,7HINST(1,,I5,2H),,MAIN2160
     $I2,I5,2HS))                                                       MAIN2170
       GO TO 57                                                         MAIN2180
 1900  CALL SLABL(ISOURCE,SOURCE)                                       MAIN2190
       IF(LABEL.EQ.0)GO TO 1902                                         MAIN2200
       WRITE(109,1901)LABEL,IR,JR,IC,JC,INDX,II,SOURCE                  MAIN2210
 1901  FORMAT(I5,1X,10HCALL SBNF(,4(I2,1H),),8HCPA,CPQ,,7HINST(1,,I5,2H),,MAIN2220
     $I2,I5,2HS))                                                       MAIN2230
       GO TO 57                                                         MAIN2240
 1902  WRITE(109,1903)      IR,JR,IC,JC,INDX,II,SOURCE                  MAIN2250
 1903  FORMAT(5X,1X,10HCALL SBNF(,4(I2,1H),),8HCPA,CPQ,,7HINST(1,,I5,2H),,MAIN2260
     $I2,I5,2HS))                                                       MAIN2270
       GO TO 57                                                         MAIN2280
 2000  DECODE(80,2001,TEMP)SOURCE                                       MAIN2290
 2001  FORMAT(T33,A3)                                                   MAIN2300
       IF(LABEL.EQ.0)GO TO 2003                                         MAIN2310
       WRITE(109,2002)LABEL,IR,JR,IC,JC,INDX,SOURCE                     MAIN2320
```

```
 2002  FORMAT(I5,1X,10HCALL SRCH(,4(I2,1H,),7HINST(1,I5,2H),,A3,23HMRC,MA   MAIN2330
      $SK,MAX,JMAX,KMAX)                                                    MAIN2340
       GO TO 57                                                             MAIN2350
 2003  WRITE(109,2004)IR,JR,IC,JC,INDX,SOURCE                               MAIN2360
 2004  FORMAT(  6X ,10HCALL SRCH(,4(I2,1H,),7HINST(1,I5,2H),,A3,23HMRC,MA   MAIN2370
      $SK,MAX,JMAX,KMAX)                                                    MAIN2380
  126  FORMAT(T33,10A1)                                                     MAIN2390
   57  CONTINUE                                                             MAIN2400
 9999  WRITE(108,9998)                                                      MAIN2410
 9998  FORMAT(1H1)                                                          MAIN2420
  999  WRITE(108,998)                                                       MAIN2430
  998  FORMAT(' FORTRAN AND 360 INSTRUCTION MIX!)                           MAIN2440
       WRITE(108,997)((INST(I,J),I=1,20),J=1,MINDX)                         MAIN2450
  997  FORMAT(1X,20A4)                                                      MAIN2460
C                                                                          MAIN2470
CC    BUILD A FORTRAN EQUIVALENT PROGRAM TO THE MIXED MODE ONE IN FILE SIM  MAIN2480
CC   THROUGH F1109                                                         MAIN2490
C                                                                          MAIN2500
       CALL EXIT                                                           MAIN2510
       END                                                                 MAIN2520
```

```
C  $$$$
```

```
      IF(K.EQ.1H0.OR.K.EQ.1H1.OR.K.EQ.1H2.OR.K.EQ.1H3.OR.K.EQ.1H4.OR.K.
     $EQ.1H5.OR.K.EQ.1H6.OR.K.EQ.1H7.OR.K.EQ.1H8.OR.K.EQ.1H9)JC1=I)
     $GO TO 70
65    CONTINUE
70    DO 75 I=JC1+1,16
      K=F1(I)
      IF(K.NE.1H0.AND.K.NE.1H1.AND.K.NE.1H2.AND.K.NE.1H3.AND.K.NE.1H4.
     $AND.K.NE.1H5.AND.K.NE.1H6.AND.K.NE.1H7.AND.K.NE.1H8.AND.K.NE.1H9)
     $JC2=I-1 GO TO 80
75    CONTINUE
80    IR=0
      JR=0
      IC=0
      JC=0
      IF(IR1.EQ.0)RETURN
      DO 85 I=IR1,IR2
      IR=IR*10+J1(I)
85    CONTINUE
      DO 90 I=JR1,JR2
      JR=JR*10+J1(I)
90    CONTINUE
      DO 95 I=IC1,IC2
      IC=IC*10+J1(I)
95    CONTINUE
      DO 100 I=JC1,JC2
      JC=JC*10+J1(I)
100   CONTINUE
      RETURN
      END
```

```
SFLD00480
SFLD00490
SFLD00500
SFLD00510
SFLD00520
SFLD00530
SFLD00540
SFLD00550
SFLD00560
SFLD00570
SFLD00580
SFLD00590
SFLD00600
SFLD00610
SFLD00620
SFLD00630
SFLD00640
SFLD00650
SFLD00660
SFLD00670
SFLD00680
SFLD00690
SFLD00700
SFLD00710
SFLD00720
SFLD00730
SFLD00740
SFLD00750
SFLD00760
```

```
C  $$$$
```

```
!ASSIGN MIBO,(FILE,SUBP)
!FORTRAN LS,S,BO
C  $$$$
```

```
C       ***********************
        SUBROUTINE SMASK(IR,JR,IC,JC,CN1,MRC,MASK,II)                    SMSK0000
        INTEGER MRC(16,16,16),MASK(16,16),CN1                            SMSK0010
        LOGICAL CC(4)                                                    SMSK0020
        DO 5 I=1,4                                                       SMSK0030
 5      CC(I)=.FALSE.                                                    SMSK0040
C                                                                        SMSK0050
CCC   BREAK DOWN THE CANCELLATION CODE INTO ITS INTEGER POWERS OF TWO.   SMSK0060
CC                                                                       SMSK0070
        CN1R=CN1                                                         SMSK0080
        IF(CN1.GE.8)CN1R=MOD(CN1,8)CC(4)=.TRUE.                          SMSK0090
        IF(CN1R.GE.4)CN1R=MOD(CN1R,4)CC(3)=.TRUE.                        SMSK0100
        IF(CN1R.GE.2)CN1R=MOD(CN1R,2)CC(2)=.TRUE.                        SMSK0110
        IF(CN1R.GE.1)CC(1)=.TRUE.                                        SMSK0120
C                                                                        SMSK0130
CCC   IF THE CANCELLATION NUMBER CORRESPONDS TO A POSITION OF THE CANCELLATSMSK0140
CCCC  THAT IS A '1' RATHER THAN A '0', SET THE MASK TO ZERO AT THAT GRID POSMSK0150
CCCCC  A '1' IN THE MASK ALLOWS EXECUTION WHILE A '0' INHIBITS EXECUTION.  SMSK0160
CC                                                                       SMSK0170
        DO 10 I=IR,JR                                                    SMSK0180
        DO 10 J=IC,JC                                                    SMSK0190
        INDEX=MOD(MRC(II,I,J)/4*16**5,4)+1                               SMSK0200
        MASK(I,J)=1                                                      SMSK0210
        IF(CC(INDEX)) MASK(I,J)=0                                        SMSK0220
 10     CONTINUE                                                         SMSK0230
C*******DEBUGGING*****************                                      SMSK0240
        WRITE(108,12)                                                    SMSK0250
 12     FORMAT('0DEBUGGING CHECK OF MASK MATRIX')                        SMSK0260
        DO 14 I=IR,JR                                                    SMSK0270
 14     WRITE(108,13) (MASK(I,J),J=IC,JC)                                SMSK0280
 13     FORMAT('0',16I4)                                                 SMSK0290
C*******DEBUGGING*****************                                      SMSK0300
        RETURN                                                           SMSK0310
        END                                                              SMSK0320
                                                                         SMSK0330
```

```
C      $$$$$$$$$$$$$$$$$$$$$$$$
       SUBROUTINE SINPT(IR,JR,IC,JC,TEMP,MR1,MR2,MRC,CR,TR)                    SIP00000
       REAL MR1(16,16,16),MR2(16,16,16),RTEMP(16,16)                           SIP00010
       INTEGER MRC(16,16,16),CR(10,16,16),TEMP(20),CN1,CN2,ITEMP(16,16),       SIP00020
      $MASK(16,16),SOURCE,TR(16,16)                                            SIP00030
       COMMON SCOUNT,PCOUNT                                                    SIP00040
C*********DEBUGGING**********************                                     SIP00050
       WRITE(108,999)TEMP                                                      SIP00060
 999   FORMAT('1',1X,20A4)                                                     SIP00070
C                                                                              SIP00080
       DECODE(80,1,TEMP)CN1,CN2,SOURCE,II                                      SIP00090
 1     FORMAT(T22,I2,1X,I2,T33,A2,T37,I2)                                      SIP00100
C*********DEBUGGING**********************                                     SIP00110
       WRITE(108,17)IR,JR,IC,JC,II                                             SIP00120
 17    FORMAT('0DEBUGGING CHECK ON ARRAY INDEXES'/'IR=',I2,5X,                 SIP00130
      $'JR=',I2,5X,'IC=',I2,5X,'JC=',I2,5X,'II=',I2)                           SIP00140
C*********DEBUGGING**********************                                     SIP00150
       IF(SOURCE.EQ.3HMR )GO TO 5                                              SIP00160
       IF(SOURCE.EQ.3HMC )GO TO 15                                             SIP00170
       IF(SOURCE.EQ.3HCR )GO TO 25                                             SIP00180
       WRITE(108,2)TEMP                                                        SIP00190
 2     FORMAT('ILLEGAL INPUT CODE'/1X,20A4)                                    SIP00200
 5     IF(15-CN1)7,7,6                                                         SIP00210
 7     IF(15-CN2)8,8,9                                                         SIP00220
 6     CALL SMASK(IR,JR,IC,JC,CN1,MRC,MASK,II)                                 SIP00230
       READ(105,3)(((RTEMP(I,J),J=IC,JC),I=IR,JR)                              SIP00240
 3     FORMAT(8E10.3)                                                          SIP00250
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                   SIP00260
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES          SIP00270
C*********DEBUGGING**********************                                     SIP00280
       WRITE(108,16)                                                           SIP00290
 16    FORMAT('0DEBUGGING CHECK ON RTEMP DURING SINPT')                        SIP00300
       DO 18 I=IR,JR                                                           SIP00310
 18    WRITE(108,19)(RTEMP(I,J),J=IC,JC)                                       SIP00320
 19    FORMAT('0',8F15.6)                                                      SIP00330
C*********DEBUGGING**********************                                     SIP00340
       DO 105 I=IR,JR                                                          SIP00350
       DO 105 J=IC,JC                                                          SIP00360
       IF(MASK(I,J))105,105,106                                                SIP00370
 106   MR1(II,I,J)=RTEMP(I,J)                                                  SIP00380
 105   CONTINUE                                                                SIP00390
C****DEBUGGING**********************                                          SIP00400
       WRITE(108,4)II                                                          SIP00410
 4     FORMAT('0DEBUGGING CHECK OF MR1(',I2,') REGISTERS AFTER SINPT')         SIP00420
       DO 20 I=IR,JR                                                           SIP00430
 20    WRITE(108,19)(MR1(II,I,J),J=IC,JC)                                      SIP00440
C****DEBUGGING**********************                                          SIP00450
       IF(15-CN2)10,10,9                                                       SIP00460
                                                                               SIP00470
```

```
 9      CALL SMASK(IR,JR,IC,JC,CN2,MRC,MASK,II)                                  SIP0480
        READ(105,3) ((RTEMP(I,J),J=IC,JC),I=IR,JR)                              SIP0490
        CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                   SIP0500
C   USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           SIP0510
        DO 107 I=IR,JR                                                          SIP0520
        DO 107 J=IC,JC                                                          SIP0530
        IF(MASK(I,J))107,107,108                                                SIP0540
108     MR2(II,I,J)=RTEMP(I,J)                                                  SIP0550
107     CONTINUE                                                                SIP0560
C****DEBUGGING*********************                                            SIP0570
        WRITE(108,14) II                                                        SIP0580
14      FORMAT('0DEBUGGING CHECK OF MR2('I2') REGISTERS AFTER SINPT')           SIP0590
        DO 21 I=IR,JR                                                           SIP0600
21      WRITE(108,19) (MR2(II,I,J),J=IC,JC)                                     SIP0610
C****DEBUGGING*********************                                            SIP0620
10      RETURN                                                                  SIP0630
8       WRITE(108,11)TEMP                                                       SIP0640
11      FORMAT('0IMPROPER ATTEMPT TO LOAD MEMORY REGISTERS'/1X,20A4)            SIP0650
        RETURN                                                                  SIP0660
 15     IF(15-CN1)118,118,119                                                   SIP0670
119     CALL PACK(IR,JR,IC,JC,TEMP,TR,MASK,II,CN1)                              SIP0680
        DO 30 I=IR,JR                                                           SIP0690
        DO 30 J=IC,JC                                                           SIP0700
30      MRC(II,I,J)=TR(I,J)                                                     SIP0710
        RETURN                                                                  SIP0720
118     WRITE(108,12)TEMP                                                       SIP0730
12      FORMAT('0IMPROPER ATTEMPT TO LOAD CONTROL REGISTERS'/                   SIP0740
     $20A4)                                                                     SIP0750
        RETURN                                                                  SIP0760
 25     IF(15-CN1)128,128,129                                                   SIP0770
129     CALL PACK(IR,JR,IC,JC,TEMP,TR,MASK,II,CN1)                              SIP0780
        DO 130 I=IR,JR                                                          SIP0790
        DO 130 J=IC,JC                                                          SIP0800
130     CR(II,I,J)=TR(I,J)                                                      SIP0810
        RETURN                                                                  SIP0820
128     WRITE(108,13)TEMP                                                       SIP0830
13      FORMAT('0IMPROPER ATTEMPT TO LOAD CONNECTION PATTERN REGISTERS'/1X      SIP0840
     $,20A4)                                                                    SIP0850
        RETURN                                                                  SIP0860
        END                                                                     SIP0870
```

```
C       $$$$$$$$$$$$$$$$$$$$$$$$$$                                                    SCPPT0000
        SUBROUTINE SCOMPUTE(IR,JR,IC,JC,CPA,CPQ,MRC,II,MASK,CN3)                       SCPPT0010
        INTEGER MRC(16,16,16),MASK(16,16),CN3                                          SCPPT0020
        REAL CPA(16,16),CPQ(16,16)                                                     SCPPT0030
        COMMON SCOUNT,PCOUNT                                                           SCPPT0040
        CALL SMASK (IR,JR,IC,JC,CN3,MRC,MASK,II)                                       SCPPT0050
        DO 16 I=IR,JR                                                                  SCPPT0060
        DO 16 J=IC,JC                                                                  SCPPT0070
        OPC=MRC(II,I,J)/16**6                                                          SCPPT0080
        IF(MASK(I,J))16,16,14                                                          SCPPT0090
14      IF(OPC.EQ.0)GO TO 16                                                           SCPPT0100
        X=CPA(I,J)/Y=CPQ(I,J)                                                          SCPPT0110
        GO TO (100,200,300,400,500,600,700,800,900,1000,1100,1200,1300,                SCPPT0120
       $1400,1500),OPC                                                                 SCPPT0130
 100    CPA(I,J)=INT(CPA(I,J))+INT(CPQ(I,J))                                           SCPPT0140
        GO TO 16                                                                       SCPPT0150
 200    CPA(I,J)=INT(CPA(I,I))-INT(CPQ(I,J))                                           SCPPT0160
        GO TO 16                                                                       SCPPT0170
 300    CPA(I,J)=INT(CPA(I,J))*INT(CPQ(I,J))                                           SCPPT0180
        GO TO 16                                                                       SCPPT0190
 400    CPA(I,J)=INT(CPA(I,J))/INT(CPQ(I,J))                                           SCPPT0200
        GO TO 16                                                                       SCPPT0210
S 500   LW,1  X                                                                       SCPPT0220
S       SLS,1   1                                                                      SCPPT0230
S       STW,1  X                                                                       SCPPT0240
        GO TO 15                                                                       SCPPT0250
S 600   LW,1  X                                                                       SCPPT0260
S       SLS,1   -1                                                                     SCPPT0270
S       STW,1  X                                                                       SCPPT0280
        GO TO 15                                                                       SCPPT0290
S 700   LW,1  X                                                                       SCPPT0300
S       AND,1  Y                                                                       SCPPT0310
S       STW,1  X                                                                       SCPPT0320
        GO TO 15                                                                       SCPPT0330
S 800   LW,1  X                                                                       SCPPT0340
S       ORS,1  Y                                                                       SCPPT0350
S       STW,1  X                                                                       SCPPT0360
        GO TO 15                                                                       SCPPT0370
S 900   LW,1  X                                                                       SCPPT0380
S       EOR,1  Y                                                                       SCPPT0390
S       STW,1  X                                                                       SCPPT0400
        GO TO 15                                                                       SCPPT0410
 1000   CPA(I,J)=CPA(I,J)+CPQ(I,J)                                                     SCPPT0420
        GO TO 16                                                                       SCPPT0430
 1100   CPA(I,J)=CPA(I,J)-CPQ(I,J)                                                     SCPPT0440
        GO TO 16                                                                       SCPPT0450
 1200   CPA(I,J)=CPA(I,J)*CPQ(I,J)                                                     SCPPT0460
        GO TO 16                                                                       SCPPT0470
```

```
 1300 CPA(I,J)=CRA(I,J)/CRO(I,J)                                                SCPT04800
      GO TO 16                                                                  SCPT04900
S1400 LCW,1 X                                                                   SCPT05000
S     STW,1 X                                                                   SCPT05100
      GO TO 15                                                                  SCPT05200
S1500 LW,1  X                                                                   SCPT05300
S     EOR,1 =1                                                                  SCPT05400
S     STW,1 X                                                                   SCPT05500
      GO TO 15                                                                  SCPT05600
 15   CPA(I,J)=X                                                                SCPT05700
 16   CONTINUE                                                                  SCPT05800
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                     SCPT05900
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL JONES          SCPT06000
      RETURN                                                                    SCPT06100
      END                                                                       SCPT06200
```

```
C      $$$$$$$$$$$$$$$$$$$$$$$$                                                  SOTP00000
       SUBROUTINE SOUTP(IR,JR,IC,JC,TEMP,MR1,MR2)                                 SOTP00010
       INTEGER TEMP(20),SOURCE                                                    SOTP00020
       REAL MR1(16,16,16),MR2(16,16,16)                                           SOTP00030
       COMMON SCOUNT,PCOUNT                                                       SOTP00040
C**********  DEBUGGING  ***********************                                  SOTP00050
       WRITE(108,999)TEMP                                                         SOTP00060
 999   FORMAT('1',1X,20A4)                                                        SOTP00070
C                                                                                 SOTP00080
       DECODE(80,1,TEMP)SOURCE,II                                                 SOTP00090
 1     FORMAT(T33,A3,T37,I2)                                                      SOTP00100
       IF(SOURCE .EQ.3HMR1)GO TO 5                                                SOTP00110
       IF(SOURCE .EQ.3HMR2)GO TO 15                                               SOTP00120
       IF(SOURCE .EQ.3HMR )GO TO 25                                               SOTP00130
       WRITE(108,2)TEMP                                                           SOTP00140
 2     FORMAT('0ILLEGAL OUTPUT CODE',1X,20A4)                                     SOTP00150
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                      SOTP00160
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES             SOTP00170
       RETURN                                                                     SOTP00180
 5     WRITE(108,3)II                                                             SOTP00190
 3     FORMAT('0CONTENTS OF MR1 REGISTERS OF PLANE ',I2)                          SOTP00200
       DO 4 I=IR,JR                                                               SOTP00210
 4     WRITE(108,7) (MR1(II,I,J),J=IC,JC)                                         SOTP00220
 7     FORMAT('0',/(8E15.6/))                                                     SOTP00230
       IF(SOURCE .EQ.3HMR )GO TO 15                                               SOTP00240
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                      SOTP00250
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES             SOTP00260
       RETURN                                                                     SOTP00270
 15    WRITE(108,6)II                                                             SOTP00280
 6     FORMAT('0CONTENTS OF MR2 REGISTERS OF PLANE ',I2)                          SOTP00290
       DO 8 I=IR,JR                                                               SOTP00300
 8     WRITE(108,7) (MR2(II,I,J),J=IC,JC)                                         SOTP00310
       RETURN                                                                     SOTP00320
 25    GO TO 5                                                                    SOTP00330
       END                                                                        SOTP00340
```

```
C      $$$$$$$$$$$$$$$$$$$$$$$$$$                                          SCON0000
       SUBROUTINE SCON(IR,JR,IC,JC,MRC,CON,II,REG,MASK,CN1)                 SCON0010
       INTEGER MRC(16,16,16),CON(2,16,16),MASK(16,16),REG,CN1,A,B,C,D,H1    SCON0020
C                                                                           SCON0030
C  IS THIS TRANSFER FROM MR1 TO MR1?                                        SCON0040
       IF(REG.EQ.1)A=0,B=1,C=4,D=3,GO TO 20                                 SCON0050
C                                                                           SCON0060
C  IS THIS TRANSFER FROM MR1 TO MR2?                                        SCON0070
       IF(REG.EQ.2)A=0,B=1,C=2,D=1,GO TO 20                                 SCON0080
C                                                                           SCON0090
C  IS THIS TRANSFER FROM MR2 TO MR1?                                        SCON0100
       IF(REG.EQ.3)A=2,B=3,C=4,D=3,GO TO 20                                 SCON0110
C                                                                           SCON0120
C  IS THIS TRANSFER FROM MR2 TO MR2?                                        SCON0130
       IF(REG.EQ.4)A=0,B=1,C=2,D=1,GO TO 20                                 SCON0140
C                                                                           SCON0150
       WRITE(108,7)REG,TEMP                                                 SCON0160
7      FORMAT('0ILLEGAL VALUE OF REGISTER TO REGISTER FLAG (REG=',I2,')',/  SCON0170
     $20A4)                                                                 SCON0180
  20   CALL SMASK(IR,JR,IC,JC,CN1,MRC,MASK,II)                              SCON0190
       DO 5 I=IR,JR                                                         SCON0200
       DO 5 J=IC,JC                                                         SCON0210
       IF(MASK(I,J))15,15,25                                                SCON0220
C                                                                           SCON0230
C  BREAK DOWN MRC(II,I,J) TO GET H1.                                        SCON0240
C                                                                           SCON0250
  25   H1=MOD(MRC(II,I,J)/16**5,16)                                         SCON0260
C                                                                           SCON0270
C  ESTABLISH THE REGISTER SOURCES IN THE CON MATRICES.                      SCON0280
C                                                                           SCON0290
       IF(MOD(H1,4).EQ.A.OR.MOD(H1,4).EQ.B)GO TO 15                         SCON0300
       CON(1,I,J)=I                                                         SCON0310
       CON(2,I,J)=J                                                         SCON0320
       GO TO 5                                                              SCON0330
  15   CON(1,I,J)=MOD(MRC(II,I,J)/16**C,16)+1                               SCON0340
       CON(2,I,J)=MOD(MRC(II,I,J)/16**D,16)+1                               SCON0350
   5   CONTINUE                                                             SCON0360
C**********DEBUGGING*****************                                       SCON0370
       WRITE(108,30)                                                        SCON0380
  30   FORMAT('0DEBUGGING CHECK ON CON(1,I,J) REGISTERS!')                  SCON0390
       DO 31 I=IR,JR                                                        SCON0400
  31   WRITE(108,33) (CON(1,I,J),J=IC,JC)                                   SCON0410
  33   FORMAT('0',8I8)                                                      SCON0420
C**********DEBUGGING*****************                                       SCON0430
C**********DEBUGGING*****************                                       SCON0440
       WRITE(108,32)                                                        SCON0450
  32   FORMAT('0DEBUGGING CHECK ON CON(2,I,J) REGISTERS!')                  SCON0460
       DO 34 I=IR,JR                                                        SCON0470
```

```
34      WRITE(108,33) ((CON(2,I,J),J=IC,JC)
C*********DEBUGGING*****************
        RETURN
        END
```

```
CCON0480
CCON0490
CCON0500
CCON0510
```

```
C       $$$$$$$$$$$$$$$$$$$$$$$$                                                STRN0000
        SUBROUTINE STRAN(IR,JR,IC,JC,TEMP,MR1,MR2,MRC,CPA,CPQ,CR)                STRN0010
        INTEGER MRC(16,16,16),TEMP(20),MASK(16,16),CN1,CN2,CN3,EXOP              STRN0020
        INTEGER CR(10,16,16),SOURCE,DEST                                         STRN0030
        REAL MR1(16,16,16),MR2(16,16,16),CPA(16,16),CPQ(16,16)                   STRN0040
        COMMON SCOUNT,PCOUNT                                                     STRN0050
C*********** DEBUGGING ********************                                     STRN0060
        WRITE(108,999)TEMP                                                       STRN0070
  999   FORMAT('1',1X,20A4)                                                      STRN0080
C                                                                                STRN0090
        DECODE(80,1,TEMP)CN1,CN2,CN3,SOURCE,II,DEST,JJ,EXOP                      STRN0100
    1   FORMAT(T22,I2,1X,I2,1X,I2,T33,A2,T37,I2,T49,A2,T53,I2,T65,A3)            STRN0110
        IF(SOURCE.EQ.2HMR.AND.DEST    .EQ.2HMR)GO TO 5                           STRN0120
        IF(SOURCE.EQ.2HMR.AND.DEST    .EQ.2HCP)GO TO 15                          STRN0130
        IF(SOURCE.EQ.2HCP.AND.DEST    .EQ.2HMR)GO TO 25                          STRN0140
        IF(SOURCE.EQ.2HCR.AND.DEST    .EQ.2HMC)GO TO 35                          STRN0150
        IF(SOURCE.EQ.2H  .AND.DEST    .EQ.2HCP)GO TO 124                         STRN0160
        WRITE(108,2)TEMP                                                         STRN0170
    2   FORMAT('0ILLEGAL INTERPLANE TRANSFER ADDRESS'/1X,20A4)                   STRN0180
    5   IF(CN1.EQ.15.AND.CN2.EQ.15)GO TO 100                                     STRN0190
        IF(15-CN1)111,111,106                                                    STRN0200
  106   CALL SMASK(IR,JR,IC,JC,CN1,MRC,MASK,II)                                  STRN0210
        DO 107 I=IR,JR                                                           STRN0220
        DO 107 J=IC,JC                                                           STRN0230
        IF(MASK(I,J))107,107,108                                                 STRN0240
  108   MR1(JJ,I,J)=MR1(II,I,J)                                                  STRN0250
  107   CONTINUE                                                                 STRN0260
        CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                    STRN0270
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES            STRN0280
        IF(15-CN2)100,114,111                                                    STRN0290
  111   CALL SMASK(IR,JR,IC,JC,CN2,MRC,MASK,II)                                  STRN0300
        DO 112 I=IR,JR                                                           STRN0310
        DO 112 J=IC,JC                                                           STRN0320
        IF(MASK(I,J))112,112,113                                                 STRN0330
  113   MR2(JJ,I,J)=MR2(II,I,J)                                                  STRN0340
  112   CONTINUE                                                                 STRN0350
  114   RETURN                                                                   STRN0360
  100   WRITE(108,101)TEMP                                                       STRN0370
  101   FORMAT('0IMPROPER ATTEMPT AT INTERPLANE TRANSFER'/1X,20A4)               STRN0380
        RETURN                                                                   STRN0390
   15   IF(CN1.EQ.15.AND.CN2.EQ.15.AND.CN3.GT.15)GO TO 16                        STRN0400
        IF(CN1.EQ.15.AND.CN2.EQ.15.AND.CN3.LT.16)GO TO 100                       STRN0410
        IF(15-CN1)121,121,116                                                    STRN0420
  116   CALL SMASK(IR,JR,IC,JC,CN1,MRC,MASK,II)                                  STRN0430
        DO 117 I=IR,JR                                                           STRN0440
        DO 117 J=IC,JC                                                           STRN0450
        IF(MASK(I,J))117,117,118                                                 STRN0460
  118   CPA(I,J)=MR1(II,I,J)                                                     STRN0470
```

```
  117   CONTINUE                                                                  STRN0480
        CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                   STRN0490
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES            STRN0500
  121   IF(15-CN2)124,124,119                                                     STRN0510
  119   CALL SMASK(IR,JR,IC,JC,CN2,MRC,MASK,II)                                   STRN0520
        DO 115 I=IR,JR                                                          STRN0530
        DO 115 J=IC,JC                                                          STRN0540
        IF (MASK(I,J))115,115,123                                               STRN0550
  123   CRQ(I,J)=MR2(II,I,J)                                                      STRN0560
  115   CONTINUE                                                                  STRN0570
        CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                   STRN0580
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES            STRN0590
  124   IF(CN3.GT.15)GO TO 16                                                     STRN0600
        CALL SCOMPUTE(IR,JR,IC,JC,CPA,CRQ,MRC,II,MASK,CN3)                        STRN0610
        RETURN                                                                  STRN0620
   16   CN3=MOD(CN3,16)                                                          STRN0630
        CALL EXCOMPUTE(IR,JR,IC,JC,CPA,CRQ,MRC,II,MASK,CN3,EXOP,TEMP)             STRN0640
        RETURN                                                                  STRN0650
  120   RETURN                                                                    STRN0660
   25   IF(CN1.EQ.15.AND.CN2.EQ.15)GO TO 100                                      STRN0670
        IF(15-CN1)131,131,126                                                   STRN0680
  126   CALL SMASK(IR,JR,IC,JC,CN1,MRC,MASK,JJ)                                   STRN0690
        DO 127 I=IR,JR                                                          STRN0700
        DO 127 J=IC,JC                                                          STRN0710
        IF(MASK(I,J))127,127,128                                                STRN0720
  128   MR1(JJ,I,J)=CPA(I,J)                                                      STRN0730
  127   CONTINUE                                                                  STRN0740
        CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                   STRN0750
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES            STRN0760
        IF(15-CN2)100,138,131                                                   STRN0770
  131   CALL SMASK(IR,JR,IC,JC,CN2,MRC,MASK,JJ)                                   STRN0780
        DO 132 I=IR,JR                                                          STRN0790
        DO 132 J=IC,JC                                                          STRN0800
        IF(MASK(I,J))132,132,133                                                STRN0810
  133   MR2(JJ,I,J)=CRQ(I,J)                                                      STRN0820
  132   CONTINUE                                                                  STRN0830
        CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                   STRN0840
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES            STRN0850
  138   RETURN                                                                    STRN0860
   35   IF(CN1.EQ.15.AND.CN2.EQ.15)GO TO 100                                      STRN0870
        IF(15-CN1)135,135,136                                                   STRN0880
  136   DO 137 I=IR,JR                                                          STRN0890
        DO 137 J=IC,JC                                                          STRN0900
        MRC(JJ,I,J)=CR(II,I,J)                                                  STRN0910
  137   CONTINUE                                                                  STRN0920
        CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                   STRN0930
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES            STRN0940
C**********DEBUGGING**********                                                   STRN0950
```

```
      WRITE(108,129)                                                    STRN0960
129   FORMAT('0DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB ',STRN0970
     $'STRAN')                                                          STRN0980
      DO 130 I=IR,JR                                                    STRN0990
      WRITE(108,134)(MRC(JJ,I,J),J=IC,JC)                               STRN1000
134   FORMAT('0',/,(8(4X,Z8)/))                                         STRN1010
130   CONTINUE                                                          STRN1020
C**********DEBUGGING**************                                      STRN1030
      RETURN                                                            STRN1040
135   WRITE(108,139)TEMP                                                STRN1050
139   FORMAT('0ILLEGAL ATTEMPT AT CONNECTION PATTERN TRANSFER',/1X,20A4) STRN1060
      RETURN                                                            STRN1070
      END                                                               STRN1080
```

```
C       $$$$$$$$$$$$$$$$$$$$$$$$                                                SDRN0000
        SUBROUTINE SDTRN(IR,JR,IC,JC,TEMP,MR1,MR2,MRC)                           SDRN0010
        INTEGER MRC(16,16,16),MASK(16,16),TEMP(20),CN1,SOURCE,DEST               SDRN0020
        REAL MR1(16,16,16),MR2(16,16,16)                                         SDRN0030
        COMMON SCOUNT,PCOUNT                                                     SDRN0040
C*********** DEBUGGING **********************                                  SDRN0050
        WRITE(108,999)TEMP                                                       SDRN0060
  999   FORMAT('1',1X,20A4)                                                      SDRN0070
C                                                                                SDRN0080
        DECODE(80,1,TEMP)CN1,SOURCE,II,DEST,JJ                                   SDRN0090
  1     FORMAT(T22,I2,T33,A3,T37,I2,T49,A3,T53,I2)                               SDRN0100
        IF(II-JJ)6,5,6                                                           SDRN0110
  6     WRITE(108,4)TEMP                                                         SDRN0120
  4     FORMAT('0ILLEGAL SELECTION OF PLANE IN DIRECT TRANSFER'/1X,20A4)         SDRN0130
        RETURN                                                                   SDRN0140
  5     CALL SMASK(IR,JR,IC,JC,CN1,MRG,MASK,II)                                  SDRN0150
        IF(SOURCE .EQ.3HMR1.AND.DEST  .EQ.3HMR2)GO TO 15                         SDRN0160
        IF(SOURCE .EQ.3HMR2.AND.DEST  .EQ.3HMR1)GO TO 25                         SDRN0170
        WRITE(108,2)TEMP                                                         SDRN0180
  2     FORMAT('0ILLEGAL REGISTER SELECTION IN DIRECT TRANSFER'/1X,20A4)         SDRN0190
        RETURN                                                                   SDRN0200
 15     DO 115 I=IR,JR                                                           SDRN0210
        DO 115 J=IC,JC                                                           SDRN0220
        IF(MASK(I,J))115,115,116                                                 SDRN0230
 116    MR2(II,I,J)=MR1(II,I,J)                                                  SDRN0240
 115    CONTINUE                                                                 SDRN0250
        CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                    SDRN0260
C   USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           SDRN0270
        RETURN                                                                   SDRN0280
 25     DO 125 I=IR,JR                                                           SDRN0290
        DO 125 J=IC,JC                                                           SDRN0300
        IF(MASK(I,J))125,125,126                                                 SDRN0310
 126    MR1(II,I,J)=MR2(II,I,J)                                                  SDRN0320
 125    CONTINUE                                                                 SDRN0330
        CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                    SDRN0340
C   USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           SDRN0350
        RETURN                                                                   SDRN0360
        END                                                                      SDRN0370
```

```
C      $$$$$$$$$$$$$$$$$$$$$$$$$$                                        SCRN0000
       SUBROUTINE SCTRN(IR,JR,IC,JC,TEMP,MR1,MR2,MRC)                     SCRN0010
       INTEGER CON(2,16,16),TEMP(20),MASK(16,16),CN1,SOURCE,DEST,REG      SCRN0020
       INTEGER H1,H2,H3,H4,H5,TEMP1(16,16),TEMP2(16,16)                   SCRN0030
       INTEGER MRC(16,16,16)                                              SCRN0040
       REAL MR1(16,16,16),MR2(16,16,16),RTEMP(16,16)                      SCRN0050
       COMMON SCOUNT,PCOUNT                                               SCRN0060
C**********  DEBUGGING **********************                            SCRN0070
       WRITE(108,999)TEMP                                                 SCRN0080
  999  FORMAT('1',1X,20A4)                                                SCRN0090
C                                                                         SCRN0100
       DECODE(80,1,TEMP)CN1,SOURCE,II,DEST,JJ                             SCRN0110
    1  FORMAT(T22,I2,T33,A3,T37,I2,T49,A3,T53,I2)                         SCRN0120
       IF(II-JJ)6,4,6                                                     SCRN0130
    6  WRITE(108,2)TEMP                                                   SCRN0140
    2  FORMAT('0ILLEGAL SELECTION OF PLANE IN INTRAPLANE TRANSFER'/1X,20A SCRN0150
      *4)                                                                 SCRN0160
       RETURN                                                             SCRN0170
    4  IF(SOURCE .EQ.3HMR1.AND.DEST    .EQ.3HMR1)REG=1,GO TO 5            SCRN0180
       IF(SOURCE .EQ.3HMR1.AND.DEST    .EQ.3HMR2)REG=2,GO TO 15           SCRN0190
       IF(SOURCE .EQ.3HMR2.AND.DEST    .EQ.3HMR1)REG=3,GO TO 25           SCRN0200
       IF(SOURCE .EQ.3HMR2.AND.DEST    .EQ.3HMR2)REG=4,GO TO 35           SCRN0210
       IF(SOURCE .EQ.3HMR .AND.DEST    .EQ.3HMR )REG=5,GO TO 45           SCRN0220
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES     SCRN0230
       WRITE(108,3)TEMP                                                   SCRN0240
    3  FORMAT('0ILLEGAL INTRAPLANE REGISTER SELECTION'/1X,20A4)           SCRN0250
       RETURN                                                             SCRN0260
    5  CALL SCON(IR,JR,IC,JC,MRC,CON,II,REG,MASK,CN1)                     SCRN0270
       DO 105 I=IR,JR                                                     SCRN0280
       DO 105 J=IC,JC                                                     SCRN0290
       IF(MASK(I,J))105,105,106                                           SCRN0300
  106  RTEMP(I,J)=MR1(II,CON(1,I,J),CON(2,I,J))                           SCRN0310
  105  CONTINUE                                                           SCRN0320
       DO 107 I=IR,JR                                                     SCRN0330
       DO 107 J=IC,JC                                                     SCRN0340
       IF(MASK(I,J))107,107,108                                           SCRN0350
  108  MR1(II,I,J)=RTEMP(I,J)                                             SCRN0360
  107  CONTINUE                                                           SCRN0370
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                              SCRN0380
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES     SCRN0390
       RETURN                                                             SCRN0400
   15  CALL SCON(IR,JR,IC,JC,MRC,CON,II,REG,MASK,CN1)                     SCRN0410
       DO 115 I=IR,JR                                                     SCRN0420
       DO 115 J=IC,JC                                                     SCRN0430
       IF(MASK(I,J))115,115,116                                           SCRN0440
  116  MR2(II,I,J)=MR1(II,CON(1,I,J),CON(2,I,J))                          SCRN0450
  115  CONTINUE                                                           SCRN0460
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                              SCRN0470
```

```
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES   SCRN0480
      RETURN                                                            SCRN0490
25    CALL SCON(IR,JR,IC,JC,MRC,CON,II,REG,MASK,CN1)                    SCRN0500
      DO 125 I=IR,JR                                                    SCRN0510
      DO 125 J=IC,JC                                                    SCRN0520
      IF(MASK(I,J))125,125,126                                          SCRN0530
126   MR1(II,I,J)=MR2(II,CON(1,I,J),CON(2,I,J))                         SCRN0540
125   CONTINUE                                                          SCRN0550
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                             SCRN0560
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES   SCRN0570
      RETURN                                                            SCRN0580
35    CALL SCON(IR,JR,IC,JC,MRC,CON,II,REG,MASK,CN1)                    SCRN0590
      DO 135 I=IR,JR                                                    SCRN0600
      DO 135 J=IC,JC                                                    SCRN0610
      IF(MASK(I,J))135,135,136                                          SCRN0620
136   RTEMP(I,J)=MR2(II,CON(1,I,J),CON(2,I,J))                          SCRN0630
135   CONTINUE                                                          SCRN0640
      DO 137 I=IR,JR                                                    SCRN0650
      DO 137 J=IC,JC                                                    SCRN0660
      IF(MASK(I,J))137,137,138                                          SCRN0670
138   MR2(II,I,J)=RTEMP(I,J)                                            SCRN0680
137   CONTINUE                                                          SCRN0690
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                             SCRN0700
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES   SCRN0710
      RETURN                                                            SCRN0720
45    DO 55 I=IR,JR                                                     SCRN0730
      DO 55 J=IC,JC                                                     SCRN0740
      H1=MOD(MRC(II,I,J)/16**5,16)                                      SCRN0750
      H2=MOD(MRC(II,I,J)/16**4,16)+1                                    SCRN0760
      H3=MOD(MRC(II,I,J)/16**3,16)+1                                    SCRN0770
      H4=MOD(MRC(II,I,J)/16**2,16)+1                                    SCRN0780
      H5=MOD(MRC(II,I,J)/16,16)+1                                       SCRN0790
C NO  INPUT TO ANY REGISTER                                             SCRN0800
      IF(H1.LT.4) GO TO 65                                              SCRN0810
C  MR2 HAS INPUT                                                        SCRN0820
      IF(H1.LT.8)    GO TO 75                                           SCRN0830
C  MR1 HAS INPUT                                                        SCRN0840
      IF(H1.LT.12)GO TO 85                                              SCRN0850
C  BOTH MR1 AND MR2 HAS INPUT                                           SCRN0860
      GO TO 95                                                          SCRN0870
65    TEMP1(I,J)=MR1(II,I,J)                                            SCRN0880
      TEMP2(I,J)=MR2(II,I,J)                                            SCRN0890
      GO TO 55                                                          SCRN0900
75    TEMP1(I,J)=MR1(II,I,J)                                            SCRN0910
      IF(MOD(H1,4).EQ.0.OR.MOD(H1,4).EQ.2) TEMP2(I,J)=MR1(II,H4,H5)     SCRN0920
     *GO TO 55                                                          SCRN0930
      TEMP2(I,J)=MR2(II,H4,H5)                                          SCRN0940
      GO TO 55                                                          SCRN0950
```

```
 85     TEMP2(I,J)=MR2(II,I,J)                                                  SCRN0960
        IF(MOD(H1,4).EQ.0.OR.MOD(H1,4).EQ.1)TEMP1(I,J)=MR1(II,H2,H3)            SCRN0970
       $GO TO 55                                                                SCRN0980
        TEMP1(I,J)=MR2(II,H2,H3)                                                SCRN0990
        GO TO 55                                                                SCRN1000
 95     IF(MOD(H1,4).EQ.0.OR.MOD(H1,4).EQ.1)TEMP1(I,J)=MR1(II,H2,H3)            SCRN1010
       $GO TO 96                                                                SCRN1020
        TEMP1(I,J)=MR2(II,H2,H3)                                                SCRN1030
 96     IF(MOD(H1,4).EQ.1.OR.MOD(H1,4).EQ.3)TEMP2(I,J)=MR2(II,H4,H5)            SCRN1040
       $GO TO 55                                                                SCRN1050
        TEMP2(I,J)=MR1(II,H4,H5)                                                SCRN1060
 55     CONTINUE                                                                SCRN1070
        DO 205 I=IR,JR                                                          SCRN1080
        DO 205 J=IC,JC                                                          SCRN1090
        MR1(II,I,J)=TEMP1(I,J)                                                  SCRN1100
        MR2(II,I,J)=TEMP2(I,J)                                                  SCRN1110
 205    CONTINUE                                                                SCRN1120
        CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                 SCRN1130
        CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                 SCRN1140
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES          SCRN1150
        RETURN                                                                  SCRN1160
        END                                                                     SCRN1170
```

```
C      $$$$$$$$$$$$$$$$$$$$$$$$
       SUBROUTINE EXCOMPUTE(IR,JR,IC,JC,CPA,CPQ,MRC,II,MASK,
      $CN3,EXOP,TEMP)
       INTEGER MRC(16,16,16),MASK(16,16),CN3,EXOP
       INTEGER TEMP(20)
       REAL CPA(16,16),CPQ(16,16)
       COMMON SCOUNT,PCOUNT
       CALL SMASK(JR,JR,JC,JC,CN3,MRC,MASK,II)
       IF(EXOP.EQ.3H   )GO TO 100
       IF(EXOP.EQ.3HADD)GO TO 200
       IF(EXOP.EQ.3HSUB)GO TO 300
       IF(EXOP.EQ.3HMPY)GO TO 400
       IF(EXOP.EQ.3HDIV)GO TO 500
       IF(EXOP.EQ.3HSLI)GO TO 600
       IF(EXOP.EQ.3HSRI)GO TO 700
       IF(EXOP.EQ.3HAND)GO TO 800
       IF(EXOP.EQ.3HIOR)GO TO 900
       IF(EXOP.EQ.3HXOR)GO TO 1000
       IF(EXOP.EQ.3HFAD)GO TO 1100
       IF(EXOP.EQ.3HFSB)GO TO 1200
       IF(EXOP.EQ.3HFMY)GO TO 1300
       IF(EXOP.EQ.3HFDV)GO TO 1400
       IF(EXOP.EQ.3HCPM)GO TO 1500
       IF(EXOP.EQ.3HNOT)GO TO 1600
       WRITE(108,1)TEMP
     1 FORMAT(' ILLEGAL EXTERNAL OPERATION CODE',1X,20A4)
   200 DO 201 I=IR,JR
       DO 201 J=IC,JC
       IF(MASK(I,J))201,201,202
   202 CPA(I,J)=INT(CPA(I,J))+INT(CPQ(I,J))
   201 CONTINUE
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)
       RETURN
   300 DO 301 I=IR,JR
       DO 301 J=IC,JC
       IF(MASK(I,J))301,301,302
   302 CPA(I,J)=INT(CPA(I,J))-INT(CPQ(I,J))
   301 CONTINUE
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES
       RETURN
   400 DO 401 I=IR,JR
       DO 401 J=IC,JC
       IF(MASK(I,J))401,401,402
   402 CPA(I,J)=INT(CPA(I,J))*INT(CPQ(I,J))
   401 CONTINUE
       CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)
```

```
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES
      RETURN
  500 DO 501 I=IR,JR
      DO 501 J=IC,JC
      IF(MASK(I,J))501,501,502
  502 CPA(I,J)=INT(CPA(I,J))/INT(CPQ(I,J))
  501 CONTINUE
      CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES
      RETURN
  600 DO 601 I=IR,JR
      DO 601 J=IC,JC
      IF(MASK(I,J))601,601,602
  602 X=CPA(I,J)
S     LW,1 X
S     SLS,1 1
S     STW,1 X
      CPA(I,J)=X
  601 CONTINUE
      CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES
      RETURN
  700 DO 701 I=IR,JR
      DO 701 J=IC,JC
      IF(MASK(I,J))701,701,702
  702 X=CPA(I,J)
S     LW,1 X
S     SLS,1 -1
S     STW,1 X
      CPA(I,J)=X
  701 CONTINUE
      CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES
      RETURN
  800 DO 801 I=IR,JR
      DO 801 J=IC,JC
      IF(MASK(I,J))801,801,802
  802 X=CPA(I,J);Y=CPQ(I,J)
S     LW,1 X
S     AND,1 Y
S     STW,1 X
      CPA(I,J)=X
  801 CONTINUE
      CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES
      RETURN
  900 DO 901 I=IR,JR
      DO 901 J=IC,JC
```

```
      IF(MASK(I,J))901,901,902                                                  CCCPPTT0960
 902  X=CPA(I,J)+Y=CRQ(I,J)                                                     CCCPPTT0970
S     LW,1 X                                                                    CCCPPTT0980
S     OR,1 Y                                                                    CCCPPTT0990
S     STW,1 X                                                                   CCCPPTT1000
      CPA(I,J)=X                                                                CCCPPTT1010
 901  CONTINUE                                                                  CCCPPTT1020
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                     CCCPPTT1030
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           CCCPPTT1040
      RETURN                                                                    CCCPPTT1050
1000  DO 1001 I=IR,JR                                                           CCCPPTT1060
      DO 1001 J=IC,JC                                                           CCCPPTT1070
      IF(MASK(I,J))1001,1001,1002                                               CCCPPTT1080
1002  X=CPA(I,J)+Y=CRQ(I,J)                                                     CCCPPTT1090
S     LW,1 X                                                                    CCCPPTT1100
S     EOR,1 Y                                                                   CCCPPTT1110
S     STW,1 X                                                                   CCCPPTT1120
      CPA(I,J)=X                                                                CCCPPTT1130
1001  CONTINUE                                                                  CCCPPTT1140
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                     CCCPPTT1150
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           CCCPPTT1160
      RETURN                                                                    CCCPPTT1170
1100  DO 1101 I=IR,JR                                                           CCCPPTT1180
      DO 1101 J=IC,JC                                                           CCCPPTT1190
      IF(MASK(I,J))1101,1101,1102                                               CCCPPTT1200
1102  CPA(I,J)=CPA(I,J)+CRQ(I,J)                                                CCCPPTT1210
1101  CONTINUE                                                                  CCCPPTT1220
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                     CCCPPTT1230
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           CCCPPTT1240
      RETURN                                                                    CCCPPTT1250
1200  DO 1201 I=IR,JR                                                           CCCPPTT1260
      DO 1201 J=IC,JC                                                           CCCPPTT1270
      IF(MASK(I,J))1201,1201,1202                                               CCCPPTT1280
1202  CPA(I,J)=CPA(I,J)-CRQ(I,J)                                                CCCPPTT1290
1201  CONTINUE                                                                  CCCPPTT1300
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                     CCCPPTT1310
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           CCCPPTT1320
      RETURN                                                                    CCCPPTT1330
1300  DO 1301 I=IR,JR                                                           CCCPPTT1340
      DO 1301 J=IC,JC                                                           CCCPPTT1350
      IF(MASK(I,J))1301,1301,1302                                               CCCPPTT1360
1302  CPA(I,J)=CPA(I,J)*CRQ(I,J)                                                CCCPPTT1370
1301  CONTINUE                                                                  CCCPPTT1380
      CALL COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                     CCCPPTT1390
C  USING COUNT SUBROUTINE TO COMPARE THE 3-D MACHINE WITH SERIAL ONES           CCCPPTT1400
      RETURN                                                                    CCCPPTT1410
1400  DO 1401 I=IR,JR                                                           CCCPPTT1420
      DO 1401 J=IC,JC                                                           CCCPPTT1430
```

```
      IF(MASK(I,J))1401,1401,1402                                                    ECPT1440
 1402 CPA(I,J)=CPA(I,J)/CRQ(I,J)                                                     ECPT1450
 1401 CONTINUE                                                                       ECPT1460
      CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                         ECPT1470
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES               ECPT1480
      RETURN                                                                         ECPT1490
 1500 DO 1501 I=IR,JR                                                                ECPT1500
      DO 1501 J=IC,JC                                                                ECPT1510
      IF(MASK(I,J))1501,1501,1502                                                    ECPT1520
 1502 X=CPA(I,J)                                                                     ECPT1530
S     LCW,1 X                                                                        ECPT1540
S     STW,1 X                                                                        ECPT1550
      CPA(I,J)=X                                                                     ECPT1560
 1501 CONTINUE                                                                       ECPT1570
      CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                         ECPT1580
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES               ECPT1590
      RETURN                                                                         ECPT1600
 1600 DO 1601 I=IR,JR                                                                ECPT1610
      DO 1601 J=IC,JC                                                                ECPT1620
      IF(MASK(I,J))1601,1601,1602                                                    ECPT1630
 1602 X=CPA(I,J)                                                                     ECPT1640
S     LW,1 X                                                                         ECPT1650
S     EOR,1 =-1                                                                      ECPT1660
S     STW,1 X                                                                        ECPT1670
      CPA(I,J)=X                                                                     ECPT1680
 1601 CONTINUE                                                                       ECPT1690
      CALL  COUNT(IR,JR,IC,JC,PCOUNT,SCOUNT)                                         ECPT1700
C  USING COUNT SUBROUTINE  TO COMPARE THE 3-D MACHINE WITH SERIAL ONES               ECPT1710
  100 RETURN                                                                         ECPT1720
      END                                                                            ECPT1730
```

```
C      $$$$$$$$$$$$$$$$$$$$$$$$                                          PACK0000
       SUBROUTINE PACK(IR,JR,IC,JC,TEMP,TR,MASK,II,CN1)                   PACK0010
       INTEGER A(16,16),Q(16,16),OP(16,16),CN(16,16),TR(16,16)     ,      PACK0020
      $TEMP(20),SEARCH,FLAG,REG(2),MASK(16,16),CN1                        PACK0030
C                                                                         PACK0040
C  SEARCH:  TELLS WHICH REGISTERS ARE BEING SEARCHED AS A SOURCE.         PACK0050
C           IF SEARCH = 1, THE =1 REGISTERS ARE BEING SEARCHED.           PACK0060
C           IF SEARCH = 2, THE =2 REGISTERS ARE BEING SEARCHED.           PACK0070
C  FLAG:    TELLS WHICH REGISTER IS THE DESTINATION FOR WHICH A SOURCE IPACK0080
C           IF FLAG = 1, A =1 REGISTER IS THE DESTINATION.                PACK0090
C           IF FLAG = 2, A =2 REGISTER IS THE DESTINATION.                PACK0100
C                                                                         PACK0110
      IMPLICIT INTEGER(H)                                                 PACK0120
C                                                                         PACK0130
C  DONT BEGIN ANY REAL NAME WITH AN H                                     PACK0140
C                                                                         PACK0150
      DATA REG/' ONE',' TWO'/                                             PACK0160
      WRITE(108,3)                                                        PACK0170
3     FORMAT('0REGISTER CONNECTION'/4(2X'A    Q    OP'5X))              PACK0180
      READ(105,2)((A(I,J),Q(I,J),OP(I,J),J=IC,JC),I=IR,JR)               PACK0190
2     FORMAT(4(I3,1X,I3,1X,A4,8X))                                        PACK0200
      WRITE(108,6)((A(I,J),Q(I,J),OP(I,J),J=IC,JC),I=IR,JR)               PACK0210
6     FORMAT(('0',4(I3,2X,I3,2X,A4,5X)))                                  PACK0220
      CALL SMASK (IR,JR,IC,JC,CN1,CR,MASK,II)                             PACK0230
      DO 5 I=IR,JR                                                        PACK0240
      DO 5 J=IC,JC                                                        PACK0250
      IF (MASK(I,J))5,5,4                                                 PACK0260
C                                                                         PACK0270
C  SOURCE TRACE ROUTINE FOR REGISTER SET =1.                              PACK0280
C                                                                         PACK0290
4     K1=I                                                                PACK0300
      KJ=J                                                                PACK0310
      SEARCH=1                                                            PACK0320
      FLAG=1                                                              PACK0330
      IM=MOD(A(KI,KJ),10)                                                 PACK0340
      JM=MOD(A(KI,KJ),100)/10                                             PACK0350
      KM=A(KI,KJ)/100                                                     PACK0360
      IF(JM.GE.8.OR.KM.GE.8.OR.KM.LT.0)GO TO 1001                         PACK0370
C                                                                         PACK0380
C  IS THE INPUT IN THE 000 SERIES (NO INPUT)?  IF SO, SET H1,H2 + H3 FOR  PACK0390
C  CASE OF NO INPUT TO REG =1.  THEN GO TO REG =2 SEGMENT.                PACK0400
C                                                                         PACK0410
      IF(IM.LT.7.AND.KM.GE.1)GO TO 100                                    PACK0420
      H1=0                                                                PACK0430
      H2=I-1                                                              PACK0440
      H3=J-1                                                              PACK0450
      GO TO 2000                                                          PACK0460
C                                                                         PACK0470
                                                                          PACK0480
```

```
C   IS THE INPUT IN THE 100 SERIES (FROM ABOVE)?                                  PACK0490
C                                                                                 PACK0500
100   IF(KM.GT.1)GO TO 200                                                        PACK0510
      GO TO 1400                                                                  PACK0520
C                                                                                 PACK0530
CC  IS THE INPUT IN THE 200 SERIES (FROM THE UPPER RIGHT)?                        PACK0540
C                                                                                 PACK0550
200   IF(KM.GT.2)GO TO 300                                                        PACK0560
      GO TO 1500                                                                  PACK0570
C                                                                                 PACK0580
CC  IS THE INPUT IN THE 300 SERIES (FROM THE RIGHT)?                              PACK0590
C                                                                                 PACK0600
300   IF(KM.GT.3)GO TO 400                                                        PACK0610
      GO TO 1600                                                                  PACK0620
C                                                                                 PACK0630
CC  IS THE INPUT IN THE 400 SERIES (FROM BELOW)?                                  PACK0640
C                                                                                 PACK0650
400   IF(KM.GT.4)GO TO 500                                                        PACK0660
      GO TO 1100                                                                  PACK0670
C                                                                                 PACK0680
CC  IS THE INPUT IN THE 500 SERIES (FROM THE LOWER LEFT)?                         PACK0690
C                                                                                 PACK0700
500   IF(KM.GT.5)GO TO 600                                                        PACK0710
      GO TO 1200                                                                  PACK0720
C                                                                                 PACK0730
CC  IS THE INPUT IN THE 600 SERIES (FROM THE LEFT)?                               PACK0740
C                                                                                 PACK0750
600   IF(KM.GT.6)GO TO 700                                                        PACK0760
      GO TO 1300                                                                  PACK0770
C                                                                                 PACK0780
CC  IS THE INPUT IN THE 700 SERIES (FROM REG 2)?                                  PACK0790
700   IF(KM.NE.7)GO TO 1001                                                       PACK0810
      SEARCH=2 GO TO 3700                                                         PACK0820
C                                                                                 PACK0830
CC  REGISTER OUTPUT IS IN THE 10 SERIES (UPWARD).                                 PACK0840
C                                                                                 PACK0850
1100  KI=KI+1                                                                     PACK0860
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001                 PACK0870
      IM=MOD(A(KI,KJ),10)                                                         PACK0880
      JM=MOD(A(KI,KJ),100)/10                                                     PACK0890
      KM=A(KI,KJ)/100                                                             PACK0900
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                          PACK0910
      IF(JM.GT.7.OR.KM.GT.7.OR.A(KI,KJ).LT.0)GO TO 1001                           PACK0920
C  IS A VERTICAL BUSS PRESENT?  IF SO, IS IT CONFLICT FREE?                       PACK0930
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 1102                               PACK0940
      IF(JM.NE.1.AND.JM.NE.4.AND.KM.NE.1.AND.KM.NE.4)GO TO 1100                   PACK0950
      GO TO 1001                                                                  PACK0960
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED DOWNWARD.            PACK0970
```

```
C  INTERCHANGE JM AND KM IF NECESSARY.                                    PACK0980
1102  IF(JM.NE.1.AND.IM.GE.7)LM=JM;JM=KM;KM=LM                           PACK0990
      IF(JM.NE.1) GO TO 1001                                             PACK1000
      GO TO(1001,1120,1130,1140,1150,1160,1170),KM                       PACK1010
C  THE INPUT IS FROM THE UPPER RIGHT.                                     PACK1020
1120  IF(IM.LE.1)GO TO 6000                                              PACK1030
      IF(IM.EQ.3)GO TO 1001                                              PACK1040
      GO TO 1500                                                         PACK1050
C  THE INPUT IS FROM THE RIGHT.                                           PACK1060
1130  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                   PACK1070
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1600                                   PACK1080
      GO TO 1001                                                         PACK1090
1140  IF(IM.LE.1.OR.IM.EQ.3)GO TO 6000                                   PACK1100
      GO TO 1100                                                         PACK1120
C  THE INPUT IS FROM THE LOWER LEFT.                                      PACK1130
1150  IF(IM.LE.1)GO TO 6000                                              PACK1140
      IF(IM.EQ.3)GO TO 1001                                              PACK1150
      GO TO 1200                                                         PACK1160
C  THE INPUT IS FROM THE LEFT.                                            PACK1170
1160  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                   PACK1180
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1300                                   PACK1190
      GO TO 1001                                                         PACK1200
C  THE INPUT IS FROM THE =2 REGISTER.                                     PACK1210
1170  IF(IM.LE.1.OR.IM.EQ.3)GO TO 6000                                   PACK1220
      SEARCH=2;GO TO 3700                                                PACK1230
C                                                                        PACK1240
C  REGISTER OUTPUT IS IN THE =20 SERIES (TO THE UPPER RIGHT).             PACK1250
C                                                                        PACK1260
1200  KI=KI+1                                                            PACK1270
      KJ=KJ-1                                                            PACK1280
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001        PACK1290
      IM=MOD(A(KI,KJ),10)                                                PACK1300
      JM=MOD(A(KI,KJ),1001)/10                                           PACK1310
      KM=A(KI,KJ)/100                                                    PACK1320
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                 PACK1330
      IF(JM.GT.7.OR.KM.GT.7.OR.A(KI,KJ).LT.0)GO TO 1001                  PACK1340
      IF(IM.NE.3)GO TO 1202                                              PACK1360
C  THE INPUT IS FROM BELOW.                                               PACK1100
C                                                                        PACK0800
C  IS THERE A DIAGONAL BUSS?  IF SO, IS IT CONFLICT FREE?                 PACK1350
      IF(JM.NE.2.AND.JM.NE.5.AND.KM.NE.2.AND.KM.NE.5)GO TO 1200          PACK1370
      GO TO 1001                                                         PACK1380
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED TO THE       PACK1390
C  UPPER RIGHT.  INTERCHANGE JM AND KM IF NECESSARY.                      PACK1400
1202  IF(JM.NE.2.AND.IM.GE.7)LM=JM;JM=KM;KM=LM                           PACK1410
      IF(JM.NE.2)GO TO 1001                                              PACK1420
      GO TO(1210,1001,1230,1240,1250,1260,1270),KM                       PACK1430
C  THE INPUT IS FROM ABOVE.                                               PACK1440
```

```
1210  IF(IM.LE.1)GO TO 6000                                                  PACK1450
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 1400                          PACK1460
      GO TO 1001                                                             PACK1470
C  THE INPUT IS FROM THE RIGHT.                                              PACK1480
1230  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                       PACK1490
      IF(IM.NE.5.AND.IM.NE.8)GO TO 1600                                      PACK1500
      GO TO 1001                                                             PACK1510
C  THE INPUT IS FROM BELOW.                                                  PACK1520
1240  IF(IM.LE.1)GO TO 6000                                                  PACK1530
      IF(IM.EQ.2.OR.IM.EQ.6)GO TO 1001                                       PACK1540
      GO TO 1100                                                             PACK1550
C  THE INPUT IS FROM THE LOWER LEFT.                                         PACK1560
1250  IF(IM.LE.2)GO TO 6000                                                  PACK1570
      GO TO 1200                                                             PACK1580
C  THE INPUT IS FROM THE LEFT.                                               PACK1590
1260  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                       PACK1600
      IF(IM.EQ.1.OR.IM.EQ.5.OR.IM.EQ.8)GO TO 1001                            PACK1610
      GO TO 1300                                                             PACK1620
C  THE INPUT IS FROM THE =2 REGISTER.                                        PACK1630
1270  IF(IM.LE.2)GO TO 6000                                                  PACK1640
      SEARCH=2)GO TO 3700                                                    PACK1650
C                                                                            PACK1660
C  REGISTER OUTPUT IS IN THE =30 SERIES (TO RIGHT).                          PACK1670
C                                                                            PACK1680
1300  KJ=KU=1                                                                PACK1690
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KU.LT.IC.OR.KU.GT.JC)GO TO 1001             PACK1700
      IM=MOD(A(KI,KU),10)                                                    PACK1710
      JM=MOD(A(KI,KU),1000)/10                                               PACK1720
      KM=A(KI,KU)/1000                                                       PACK1730
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                     PACK1740
      IF(JM.GT.7.OR.KM.GT.7.OR.A(KI,KU).LT.0)GO TO 1001                      PACK1750
C  IS THERE A HORIZONTAL BUSS?  IF SO, IS IT CONFLICT FREE?                  PACK1760
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 1302                          PACK1770
      IF(JM.NE.3.AND.JM.NE.6.AND.KM.NE.3.AND.KM.NE.6)GO TO 1300              PACK1780
      GO TO 1001                                                             PACK1790
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED TO THE RIGH     PACK1800
C  INTERCHANGE JM AND KM IF NECESSARY.                                       PACK1810
1302  IF(JM.NE.3.AND.IM.GE.7)LM=JM!JM=KM!KM=LM                               PACK1820
      IF(JM.NE.3)GO TO 1001                                                  PACK1830
      GO TO(1310,1320,1001,1340,1350,1360,1370),KM                           PACK1840
C  THE INPUT IS FROM ABOVE.                                                  PACK1850
1310  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                       PACK1860
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1400                                       PACK1870
      GO TO 1001                                                             PACK1880
C  THE INPUT IS FROM THE UPPER RIGHT.                                        PACK1890
1320  IF(IM.LE.2)GO TO 6000                                                  PACK1900
      IF(IM.EQ.3)GO TO 1001                                                  PACK1910
      GO TO 1500                                                             PACK1920
```

```
C  THE INPUT IS FROM BELOW.
1340  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1100
      GO TO 1001
C  THE INPUT IS FROM THE LOWER LEFT.
1350  IF(IM.LE.2)GO TO 6000
      IF(IM.EQ.3)GO TO 1001
      GO TO 1200
C  THE INPUT IS FROM THE LEFT.
1360  IF(IM.LE.3)GO TO 6000
      GO TO 1300
C  THE INPUT IS FROM THE =2 REGISTER.
1370  IF(IM.LE.3)GO TO 6000
      SEARCH=2:GO TO 3700
C
CC  REGISTER OUTPUT IS IN THE =40 SERIES (DOWNWARD).
C
1400  KI=KI+1
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001
      IM=MOD(A(KI,KJ),10)
      JM=MOD(A(KI,KJ),100)/10
      KM=A(KI,KJ)/100
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001
      IF(JM.GT.7.OR.KM.GT.7.OR.A(KI,KJ).LT.0)GO TO 1001
C  IS THERE A VERTICAL BUSS?  IF SO, IS IT CONFLICT FREE?
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 1402
      IF(JM.NE.1.AND.JM.NE.4.AND.KM.NE.1.AND.KM.NE.4)GO TO 1400
      GO TO 1001
C  IF A BUSS THROUGH IS PRESENT, MAKE SURE THAT ONE END OF THE BUSS
CC  IS DIRECTED DOWN.  INTERCHANGE JM AND KM IF NECESSARY.
1402  IF(JM.NE.4.AND.IM.GE.7)LM=JM:JM=KM:KM=LM
      IF(JM.NE.4)GO TO 1001
      GO TO(1410,1420,1430,1001,1450,1460,1470),KM
C  FOR ALL CASES BELOW, THERE IS NO VERTICAL BUSSING.
CC  THE INPUT IS FROM ABOVE.
1410  IF(IM.LE.2)GO TO 6000
      GO TO 1400
C  THE INPUT IS FROM THE UPPER RIGHT.
1420  IF(IM.LE.1)GO TO 6000
      IF(JM.EQ.3)GO TO 1001
      GO TO 1500
C  THE INPUT IS FROM THE RIGHT.
1430  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1600
      GO TO 1001
C  THE INPUT IS FROM THE LOWER LEFT.
1450  IF(IM.LE.1)GO TO 6000
      IF(IM.EQ.3)GO TO 1001
```

```
      GO TO 1200                                                        PACK2410
C  THE INPUT IS FROM THE LEFT.                                         PACK2420
1460  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                  PACK2430
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1300                                  PACK2440
      GO TO 1001                                                        PACK2450
C  THE INPUT IS FROM THE =2 REGISTER.                                   PACK2460
1470  IF(IM.LE.3)GO TO 6000                                             PACK2470
      SEARCH=2.GO TO 3700                                               PACK2480
C                                                                       PACK2490
C  REGISTER OUTPUT IS IN THE =50 SERIES (TO LOWER LEFT).                PACK2500
C                                                                       PACK2510
1500  KI=KI+1                                                           PACK2520
      KJ=KJ+1                                                           PACK2530
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001       PACK2540
      IM=MOD(A(KI,KJ),10)                                               PACK2550
      JM=MOD(A(KI,KJ),100)/10                                           PACK2560
      KM=A(KI,KJ)/100                                                   PACK2570
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                PACK2580
      IF(JM.GT.7.OR.KM.GT.7.OR.A(KI,KJ).LT.0)GO TO 1001                 PACK2590
C  IS THERE A DIAGONAL BUSS?  IF SO, IS IT CONFLICT FREE?               PACK2600
      IF(IM.NE.3)GO TO 1502                                             PACK2610
      IF(JM.NE.2.AND.JM.NE.5.AND.KM.NE.2.AND.KM.NE.5)GO TO 1500         PACK2620
      GO TO 1001                                                        PACK2630
C  IF A BUSS IS PRESENT MAKE SURE ONE END OF THE BUSS IS DIRECTED       PACK2640
C  DOWN AND LEFT.  INTERCHANGE JM AND KM IF NECESSARY.                  PACK2650
1502  IF(JM.NE.5.AND.IM.GE.7)LM=JM.JM=KM.KM=LM                          PACK2660
      IF(JM.NE.5)GO TO 1001                                             PACK2670
      GO TO(1510,1520,1530,1540,1001,1560,1570),KM                      PACK2680
C  FOR ALL CASES BELOW, THERE IS NO DIAGONAL BUSSING.                   PACK2690
C  THE INPUT IS FROM ABOVE.                                             PACK2700
1510  IF(IM.LE.1)GO TO 6000                                             PACK2710
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 1400                     PACK2720
      GO TO 1001                                                        PACK2730
C  THE INPUT IS FROM THE UPPER RIGHT.                                   PACK2740
1520  IF(IM.LE.2)GO TO 6000                                             PACK2750
      GO TO 1500                                                        PACK2760
C  THE INPUT IS FROM THE RIGHT.                                         PACK2770
1530  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                  PACK2780
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 1600                     PACK2790
      GO TO 1001                                                        PACK2800
C  THE INPUT IS FROM BELOW.                                             PACK2810
1540  IF(IM.LE.1)GO TO 6000                                             PACK2820
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 1100                     PACK2830
      GO TO 1001                                                        PACK2840
C  THE INPUT IS FROM THE LEFT.                                          PACK2850
1560  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                  PACK2860
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 1300                     PACK2870
      GO TO 1001                                                        PACK2880
```

```
C  THE INPUT IS FROM THE =2 REGISTER.                                    BACK 2890
1570  IF(IM.LE.3)GO TO 6000                                             BACK 2900
      SEARCH=2;GO TO 3700                                               BACK 2910
C                                                                       BACK 2920
CC  REGISTER OUTPUT IS IN THE =60 SERIES (TO LEFT).                     BACK 2930
C                                                                       BACK 2940
1600  KJ=KJ+1                                                           BACK 2950
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001       BACK 2960
      IM=MOD(A(KI,KJ),10)                                               BACK 2970
      JM=MOD(A(KI,KJ),100)/10                                           BACK 2980
      KM=A(KI,KJ)/100                                                   BACK 2990
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                BACK 3000
      IF(JM.GT.7.OR.KM.GT.7.OR.A(KI,KJ).LT.0)GO TO 1001                 BACK 3010
C  IS THERE A HORIZONTAL BUSS?  IF SO, IS IT CONFLICT FREE?             BACK 3020
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 1602                     BACK 3030
      IF(JM.NE.3.AND.JM.NE.6.AND.KM.NE.3.AND.KM.NE.6)GO TO 1600         BACK 3040
      GO TO 1001                                                        BACK 3050
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED TO THE LEFT BACK 3060
C  INTERCHANGE JM AND KM IF NECESSARY.                                  BACK 3070
1602  IF(JM.NE.6.AND.IM.GE.7)LM=JM;JM=KM;KM=LM                          BACK 3080
      IF(JM.NE.6)GO TO 1001                                             BACK 3090
      GO TO(1610,1620,1630,1640,1650,1001,1670),KM                      BACK 3100
C  FOR ALL CASES BELOW, THERE IS NO HORIZONTAL BUSSING.                 BACK 3110
C  THE INPUT IS FROM ABOVE.                                             BACK 3120
1610  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                  BACK 3130
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1400                                  BACK 3140
      GO TO 1001                                                        BACK 3150
C  THE INPUT IS FROM THE UPPER RIGHT.                                   BACK 3160
1620  IF(IM.LE.2)GO TO 6000                                             BACK 3170
      IF(IM.EQ.3)GO TO 1001                                             BACK 3180
      GO TO 1500                                                        BACK 3190
C  THE INPUT IS FROM THE RIGHT.                                         BACK 3200
1630  IF(IM.LE.3)GO TO 6000                                             BACK 3210
      GO TO 1600                                                        BACK 3220
C  THE INPUT IS FROM BELOW.                                             BACK 3230
1640  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                  BACK 3240
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 1100                                  BACK 3250
      GO TO 1001                                                        BACK 3260
C  THE INPUT IS FROM THE LOWER LEFT.                                    BACK 3270
1650  IF(IM.LE.2)GO TO 6000                                             BACK 3280
      IF(IM.EQ.3)GO TO 1001                                             BACK 3290
      GO TO 1200                                                        BACK 3300
C  THE INPUT IS FROM THE =2 REGISTER.                                   BACK 3310
1670  IF(IM.LE.3)GO TO 6000                                             BACK 3320
      SEARCH=2;GO TO 3700                                               BACK 3330
C                                                                       BACK 3340
C  REGISTER OUTPUT IS IN THE =70 SERIES (TO THE =2 REGISTER).           BACK 3350
C  THIS SEGMENT CAN BE REACHED ONLY FROM A SEARCH THROUGH THE =2 REGISTE BACK 3360
```

```
C                                                                               PACK3370
1700    IM=MOD(A(KI,KJ),10)                                                     PACK3380
        JM=MOD(A(KI,KJ),100)/10                                                 PACK3390
        KM=A(KI,KJ)/100                                                         PACK3400
        IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                      PACK3410
        IF(JM.GT.7.OR.KM.GT.7.OR.A(KI,KJ).LT.0)GO TO 1001                       PACK3420
C  SINCE THIS =1 REGISTER CONNECTS TO A =2 REGISTER,                            PACK3430
C  EITHER JM OR KM MUST EQUAL 7.                                                PACK3440
        IF(JM.NE.7.AND.IM.GE.7)LM=JM;JM=KM;KM=LM                                PACK3450
        IF(JM.NE.7)GO TO 1001                                                   PACK3460
        GO TO(1710,1720,1730,1740,1750,1760,1001),KM                            PACK3470
C  THE INPUT IS FROM ABOVE.                                                     PACK3480
1710    IF(IM.LE.3.AND.IM.NE.2)GO TO 6000                                       PACK3490
        IF(IM.NE.6.AND.IM.NE.9)GO TO 1400                                       PACK3500
        GO TO 1001                                                              PACK3510
C  THE INPUT IS FROM THE UPPER RIGHT.                                           PACK3520
1720    IF(IM.LE.2)GO TO 6000                                                   PACK3530
        IF(IM.EQ.3)GO TO 1001                                                   PACK3540
        GO TO 1500                                                              PACK3550
C  THE INPUT IS FROM THE RIGHT.                                                 PACK3560
1730    IF(IM.LE.3.AND.IM.NE.1)GO TO 6000                                       PACK3570
        IF(IM.NE.5.AND.IM.NE.8)GO TO 1600                                       PACK3580
        GO TO 1001                                                              PACK3590
C  THE INPUT IS FROM BELOW.                                                     PACK3600
1740    IF(IM.LE.3.AND.IM.NE.2)GO TO 6000                                       PACK3610
        IF(IM.NE.6.AND.IM.NE.9)GO TO 1100                                       PACK3620
        GO TO 1001                                                              PACK3630
C  THE INPUT IS FROM THE LOWER LEFT.                                            PACK3640
1750    IF(IM.LE.2)GO TO 6000                                                   PACK3650
        IF(IM.EQ.3)GO TO 1001                                                   PACK3660
        GO TO 1200                                                              PACK3670
C  THE INPUT IS FROM THE LEFT.                                                  PACK3680
1760    IF(IM.LE.3.AND.IM.NE.1)GO TO 6000                                       PACK3690
        GO TO 1300                                                              PACK3700
C                                                                               PACK3710
C  SOURCE TRACE ROUTINE FOR REGISTER SET =2.                                    PACK3720
C                                                                               PACK3730
2000    SEARCH=2                                                                PACK3740
        FLAG=2                                                                  PACK3750
        KI=I                                                                    PACK3760
        KJ=J                                                                    PACK3770
        IM=MOD(Q(KI,KJ),10)                                                     PACK3780
        JM=MOD(Q(KI,KJ),100)/10                                                 PACK3790
        KM=Q(KI,KJ)/100                                                         PACK3800
        IF(JM.GE.8.OR.KM.GE.8.OR.KM.LT.0)GO TO 1001                             PACK3810
C                                                                               PACK3820
C  IS THE INPUT IN THE 000 SERIES (NO INPUT)?  IF SO, SET IH1,IH4 + IH5 FOR     PACK3830
C  CASE OF NO INPUT TO REG =2 AND TRANSFER TO CONTROL WORD PACKING SEGME        PACK3840
```

```
C
      IF(IM.LT.7.AND.KM.GE.1)GO TO 2100
      H1=H141
      H4=J=1
      H5=J=1
      GO TO 4000
C
CC   IS THE INPUT IN THE 100 SERIES (FROM ABOVE)?
C
2100  IF(KM.GT.1)GO TO 2200
      GO TO 3400
C
CC   IS THE INPUT IN THE 200 SERIES (FROM THE UPPER RIGHT)?
C
2200  IF(KM.GT.2)GO TO 2300
      GO TO 3500
C
CC   IS THE INPUT IN THE 300 SERIES (FROM THE RIGHT)?
C
2300  IF(KM.GT.3)GO TO 2400
      GO TO 3600
C
CC   IS THE INPUT IN THE 400 SERIES (FROM BELOW)?
C
2400  IF(KM.GT.4)GO TO 2500
      GO TO 3100
C
CC   IS THE INPUT IN THE 500 SERIES (FROM THE LOWER LEFT)?
C
2500  IF(KM.GT.5)GO TO 2600
      GO TO 3200
C
CC   IS THE INPUT IN THE 600 SERIES (FROM THE LEFT)?
C
2600  IF(KM.GT.6)GO TO 2700
      GO TO 3300
C
CC   IS THE INPUT IN THE 700 SERIES (FROM REG 1)?
C
2700  IF(KM.NE.7)GO TO 1001
      SEARCH=1 GO TO 1700
C
CC   REGISTER OUTPUT IS IN THE 710 SERIES (UPWARD).
C
3100  KI=KI+1
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001
      IM=MOD(Q(KI,KJ),10)
      JM=MOD(Q(KI,KJ),100)/10
```

```
PACK3850
PACK3860
PACK3870
PACK3880
PACK3890
PACK3900
PACK3910
PACK3920
PACK3930
PACK3940
PACK3950
PACK3960
PACK3970
PACK3980
PACK3990
PACK4000
PACK4010
PACK4020
PACK4030
PACK4040
PACK4050
PACK4060
PACK4070
PACK4080
PACK4090
PACK4100
PACK4110
PACK4120
PACK4130
PACK4140
PACK4150
PACK4160
PACK4170
PACK4180
PACK4190
PACK4200
PACK4210
PACK4220
PACK4230
PACK4240
PACK4250
PACK4260
PACK4270
PACK4280
PACK4290
PACK4300
PACK4310
PACK4320
```

```
      KM=Q(KI,KJ)/100                                                          PACK4330
      IF((JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                      PACK4340
      IF((JM.GT.7.OR.KM.GT.7.OR.Q(KI,KJ).LT.0)GO TO 1001                       PACK4350
C  IS THERE A VERTICAL BUSS?  IF SO, IS IT CONFLICT FREE?                       PACK4360
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 3102                            PACK4370
      IF(JM.NE.1.AND.JM.NE.4.AND.KM.NE.1.AND.KM.NE.4)GO TO 3100                PACK4380
      GO TO 1001                                                               PACK4390
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED UPWARD.            PACK4400
C  INTERCHANGE JM AND KM IF NECESSARY.                                          PACK4410
3102  IF(JM.NE.1.AND.IM.GE.7)LM=JM,JM=KM,KM=LM                                 PACK4420
      IF(JM.NE.1) GO TO 1001                                                   PACK4430
      GO TO(1001,3120,3130,3140,3150,3160,3170),KM                             PACK4440
C  THE INPUT IS FROM THE UPPER RIGHT.                                           PACK4450
3120  IF(IM.LE.1)GO TO 6000                                                    PACK4460
      IF(IM.EQ.3)GO TO 1001                                                    PACK4470
      GO TO 3500                                                               PACK4480
C  THE INPUT IS FROM THE RIGHT.                                                 PACK4490
3130  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                         PACK4500
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3600                                         PACK4510
      GO TO 1001                                                               PACK4520
C  THE INPUT IS FROM BELOW.                                                     PACK4530
3140  IF(IM.LE.1.OR.IM.EQ.3)GO TO 6000                                         PACK4540
      GO TO 3100                                                               PACK4550
C  THE INPUT IS FROM THE LOWER LEFT.                                            PACK4560
3150  IF(IM.LE.1)GO TO 6000                                                    PACK4570
      IF(IM.EQ.3)GO TO 1001                                                    PACK4580
      GO TO 3200                                                               PACK4590
C  THE INPUT IS FROM THE LEFT.                                                  PACK4600
3160  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                         PACK4610
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3300                                         PACK4620
      GO TO 1001                                                               PACK4630
C  THE INPUT IS FROM THE #1 REGISTER.                                           PACK4640
3170  IF(IM.LE.1.OR.IM.EQ.3)GO TO 6000                                         PACK4650
      SEARCH=1,GO TO 1700                                                      PACK4660
C                                                                              PACK4670
C  REGISTER OUTPUT IS IN THE #20 SERIES (TO THE UPPER RIGHT).                   PACK4680
C                                                                              PACK4690
3200  KI=KI+1                                                                  PACK4700
      KJ=KJ+1                                                                  PACK4710
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001              PACK4720
      IM=MOD(Q(KI,KJ),10)                                                      PACK4730
      JM=MOD(Q(KI,KJ),1001/10                                                  PACK4740
      KM=Q(KI,KJ)/100                                                          PACK4750
      IF((JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                      PACK4760
      IF((JM.GT.7.OR.KM.GT.7.OR.Q(KI,KJ).LT.0)GO TO 1001                       PACK4770
C  IS THRE A DIAGONAL BUSS?  IF SO, IS IT CONFLICT FREE?                        PACK4780
      IF(IM.NE.3)GO TO 3202                                                    PACK4790
      IF(JM.NE.2.AND.JM.NE.5.AND.KM.NE.2.AND.KM.NE.5)GO TO 3200                PACK4800
```

```
      GO TO 1001                                                                PACK48410
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED TO THE UPPE       PACK48420
C  RIGHT.  INTERCHANGE JM AND KM IF NECESSARY.                                  PACK48430
3202  IF(JM.NE.2.AND.IM.GE.7)LM=JM;JM=KM;KM=LM                                  PACK48440
      IF(JM.NE.2)GO TO 1001                                                     PACK48450
      GO TO(3210,1001,3230,3240,3250,3260,3270),KM                              PACK48460
C  THE INPUT IS FROM ABOVE.                                                     PACK48470
3210  IF(IM.LE.1)GO TO 6000                                                     PACK48480
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 3400                             PACK48490
      GO TO 1001                                                                PACK49000
C  THE INPUT IS FROM THE RIGHT.                                                 PACK49100
3230  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                          PACK49200
      IF(IM.NE.5.AND.IM.NE.8)GO TO 3600                                         PACK49300
      GO TO 1001                                                                PACK49400
C  THE INPUT IS FROM BELOW.                                                     PACK49500
3240  IF(IM.LE.1)GO TO 6000                                                     PACK49600
      IF(IM.EQ.2.OR.IM.EQ.6)GO TO 1001                                          PACK49700
      GO TO 3100                                                                PACK49800
C  THE INPUT IS FROM THE LOWER LEFT.                                            PACK49900
3250  IF(IM.LE.2)GO TO 6000                                                     PACK50000
      GO TO 3200                                                                PACK50100
C  THE INPUT IS FROM THE LEFT.                                                  PACK50200
3260  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                          PACK50300
      IF(IM.EQ.1.OR.IM.EQ.5.OR.IM.EQ.8)GO TO 1001                               PACK50400
      GO TO 3300                                                                PACK50500
C  THE INPUT IS FROM THE -1 REGISTER.                                           PACK50600
3270  IF(IM.LE.2)GO TO 6000                                                     PACK50700
      SEARCH=1;GO TO 1700                                                       PACK50800
C                                                                               PACK50900
C  REGISTER OUTPUT IS IN THE -30 SERIES (TO RIGHT).                             PACK51000
C                                                                               PACK51100
3300  KJ=KJ-1                                                                   PACK51200
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001               PACK51300
      IM=MOD(Q(KI,KJ),10)                                                       PACK51400
      JM=MOD(Q(KI,KJ),100)/10                                                   PACK51500
      KM=Q(KI,KJ)/100                                                           PACK51600
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                        PACK51700
      IF(JM.GT.7.OR.KM.GT.7.OR.Q(KI,KJ).LT.0)GO TO 1001                         PACK51800
C  IS THERE A HORIZONTAL BUSS?  IF SO, IS IT CONNECTED FREE?                    PACK51900
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 3302                             PACK52000
      IF(JM.NE.3.AND.JM.NE.6.AND.KM.NE.3.AND.KM.NE.6)GO TO 3300                 PACK52100
      GO TO 1001                                                                PACK52200
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED TO THE RIGH        PACK52300
C  INTERCHANGE JM AND KM IF NECESSARY.                                          PACK52400
3302  IF(JM.NE.3.AND.IM.GE.7)LM=JM;JM=KM;KM=LM                                  PACK52500
      IF(JM.NE.3)GO TO 1001                                                     PACK52600
      GO TO(3310,3320,1001,3340,3350,3360,3370),KM                              PACK52700
C  THE INPUT IS FROM ABOVE.                                                     PACK52800
```

```
3310  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                  PACK52 90
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3400                                  PACK53 00
      GO TO 1001                                                        PACK53 10
C  THE INPUT IS FROM THE UPPER RIGHT.                                   PACK53 20
3320  IF(IM.LE.2)GO TO 6000                                             PACK53 30
      IF(IM.EQ.3)GO TO 1001                                             PACK53 40
      GO TO 3500                                                        PACK53 50
C  THE INPUT IS FROM BELOW.                                             PACK53 60
3340  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                  PACK53 70
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3100                                  PACK53 80
      GO TO 1001                                                        PACK53 90
C  THE INPUT IS FROM THE LOWER LEFT.                                    PACK54 00
3350  IF(IM.LE.2)GO TO 6000                                             PACK54 10
      IF(IM.EQ.3)GO TO 1001                                             PACK54 20
      GO TO 3200                                                        PACK54 30
C  THE INPUT IS FROM THE LEFT.                                          PACK54 40
3360  IF(IM.LE.3)GO TO 6000                                             PACK54 50
      GO TO 3300                                                        PACK54 60
C  THE INPUT IS FROM THE -1 REGISTER.                                   PACK54 70
3370  IF(IM.LE.3)GO TO 6000                                             PACK54 80
      SEARCH=1.GO TO 1700                                               PACK54 90
C                                                                       PACK55 00
C  REGISTER OUTPUT IS IN THE -40 SERIES (DOWNWARD).                     PACK55 10
C                                                                       PACK55 20
3400  KI=KI-1                                                           PACK55 30
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001       PACK55 40
      IM=MOD(Q(KI,KJ),10)                                               PACK55 50
      JM=MOD(Q(KI,KJ),100)/10                                           PACK55 60
      KM=Q(KI,KJ)/100                                                   PACK55 70
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                PACK55 80
      IF(JM.GT.7.OR.KM.GT.7.OR.Q(KI,KJ).LT.0)GO TO 1001                 PACK55 90
C  IS THERE A VERTICAL BUSS?  IF SO, IS IT CONFLICT FREE?               PACK56 00
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 3402                     PACK56 10
      IF(JM.NE.1.AND.JM.NE.4.AND.KM.NE.1.AND.KM.NE.4)GO TO 3400         PACK56 20
      GO TO 1001                                                        PACK56 30
C  IF A BUSS THROUGH IS PRESENT, MAKE SURE THAT ONE END OF THE BUSS     PACK56 40
C  IS DIRECTED DOWN.  INTERCHANGE JM AND KM IF NECESSARY.               PACK56 50
3402  IF(JM.NE.4.AND.IM.GE.7)LM=JM.JM=KM.KM=LM                          PACK56 60
      IF(JM.NE.4)GO TO 1001                                             PACK56 70
      GO TO(3410,3420,3430,1001,3450,3460,3470),KM                      PACK56 80
C  FOR ALL CASES BELOW, THERE IS NO VERTICAL BUSSING.                   PACK56 90
C  THE INPUT IS FROM ABOVE.                                             PACK57 00
3410  IF(IM.LE.2)GO TO 6000                                             PACK57 10
      GO TO 3400                                                        PACK57 20
C  THE INPUT IS FROM THE UPPER RIGHT.                                   PACK57 30
3420  IF(IM.LE.1)GO TO 6000                                             PACK57 40
      IF(IM.EQ.3)GO TO 1001                                             PACK57 50
      GO TO 3500                                                        PACK57 60
```

```
C  THE INPUT IS FROM THE RIGHT.                                                  PACK5770
3430  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                          PACK5780
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3600                                          PACK5790
      GO TO 1001                                                                PACK5800
C  THE INPUT IS FROM THE LOWER LEFT.                                            PACK5810
3450  IF(IM.LE.1)GO TO 6000                                                     PACK5820
      IF(IM.EQ.3)GO TO 1001                                                     PACK5830
      GO TO 3200                                                                PACK5840
C  THE INPUT IS FROM THE LEFT.                                                  PACK5850
3460  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                          PACK5860
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3300                                          PACK5870
      GO TO 1001                                                                PACK5880
C  THE INPUT IS FROM THE =1 REGISTER.                                           PACK5890
3470  IF(IM.LE.3)GO TO 6000                                                     PACK5900
      SEARCH=1 GO TO 1700                                                       PACK5910
C                                                                               PACK5920
C  REGISTER OUTPUT IS IN THE =50 SERIES (TO LOWER LEFT).                        PACK5930
C                                                                               PACK5940
3500  KI=KI=1                                                                   PACK5950
      KJ=KJ+1                                                                   PACK5960
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001               PACK5970
      IM=MOD(Q(KI,KJ),10)                                                       PACK5980
      JM=MOD(Q(KI,KJ),100)/10                                                   PACK5990
      KM=Q(KI,KJ)/100                                                           PACK6000
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                        PACK6010
      IF(JM.GT.7.OR.KM.GT.7.OR.Q(KI,KJ).LT.0)GO TO 1001                         PACK6020
C  IS THERE A DIAGONAL BUSS?  IF SO, IS IT CONFLICT FREE?                       PACK6030
      IF(IM.NE.3)GO TO 3502                                                     PACK6040
      IF(JM.NE.2.AND.JM.NE.5.AND.KM.NE.2.AND.KM.NE.5)GO TO 3500                 PACK6050
      GO TO 1001                                                                PACK6060
C  IF A BUSS IS PRESENT MAKE SURE ONE END OF THE BUSS IS DIRECTED               PACK6070
C  DOWN AND LEFT.  INTERCHANGE JM AND KM IF NECESSARY.                          PACK6080
3502  IF(JM.NE.5.AND.IM.GE.7)LM=JM JM=KM KM=LM                                  PACK6090
      IF(JM.NE.5)GO TO 1001                                                     PACK6100
      GO TO(3510,3520,3530,3540,1001,3560,3570),KM                              PACK6110
C  FOR ALL CASES BELOW, THERE IS NO DIAGONAL BUSSING.                           PACK6120
C  THE INPUT IS FROM ABOVE.                                                     PACK6130
3510  IF(IM.LE.1)GO TO 6000                                                     PACK6140
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 3400                             PACK6150
      GO TO 1001                                                                PACK6160
C  THE INPUT IS FROM THE UPPER RIGHT.                                           PACK6170
3520  IF(IM.LE.2)GO TO 6000                                                     PACK6180
      GO TO 3500                                                                PACK6190
C  THE INPUT IS FROM THE RIGHT.                                                 PACK6200
3530  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                          PACK6210
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 3600                             PACK6220
      GO TO 1001                                                                PACK6230
C  THE INPUT IS FROM BELOW.                                                     PACK6240
```

```
3540  IF(IM.LE.1)GO TO 6000                                                  PACK6250
      IF(IM.NE.2.AND.IM.NE.6.AND.IM.NE.9)GO TO 3100                          PACK6260
      GO TO 1001                                                             PACK6270
C  THE INPUT IS FROM THE LEFT.                                               PACK6280
3560  IF(IM.EQ.0.OR.IM.EQ.2)GO TO 6000                                       PACK6290
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 3300                          PACK6300
      GO TO 1001                                                             PACK6310
C  THE INPUT IS FROM THE =1 REGISTER.                                        PACK6320
3570  IF(IM.LE.3)GO TO 6000                                                  PACK6330
      SEARCH=1)GO TO 1700                                                    PACK6340
C                                                                            PACK6350
C  REGISTER OUTPUT IS IN THE =60 SERIES (TO LEFT).                           PACK6360
C                                                                            PACK6370
3600  KJ=KJ+1                                                                PACK6380
      IF(KI.LT.IR.OR.KI.GT.JR.OR.KJ.LT.IC.OR.KJ.GT.JC) GO TO 1001            PACK6390
      IM=MOD(Q(KI,KJ),10)                                                    PACK6400
      JM=MOD(Q(KI,KJ),1001/10                                                PACK6410
      KM=Q(KI,KJ)/100                                                        PACK6420
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                     PACK6430
      IF(JM.GT.7.OR.KM.GT.7.OR.Q(KI,KJ).LT.0)GO TO 1001                      PACK6440
C  IS THERE A HORIZONTAL BUSS?  IF SO, IS IT CONFLICT FREE?                  PACK6450
      IF(IM.NE.1.AND.IM.NE.5.AND.IM.NE.8)GO TO 3602                          PACK6460
      IF(JM.NE.3.AND.JM.NE.6.AND.KM.NE.3.AND.KM.NE.6)GO TO 3600              PACK6470
      GO TO 1001                                                             PACK6480
C  IF A BUSS IS PRESENT, MAKE SURE ONE END OF IT IS DIRECTED TO THE LEFT.    PACK6490
C  INTERCHANGE JM AND KM IF NECESSARY.                                       PACK6500
3602  IF(JM.NE.6.AND.IM.GE.7)LM=JM;JM=KM;KM=LM                               PACK6510
      IF(JM.NE.6)GO TO 1001                                                  PACK6520
      GO TO(3610,3620,3630,3640,3650,1001,3670),KM                           PACK6530
C  FOR ALL CASES BELOW, THERE IS NO HORIZONTAL BUSSING.                      PACK6540
C  THE INPUT IS FROM ABOVE.                                                  PACK6550
3610  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                       PACK6560
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3400                                       PACK6570
      GO TO 1001                                                             PACK6580
C  THE INPUT IS FROM THE UPPER RIGHT.                                        PACK6590
3620  IF(IM.LE.2)GO TO 6000                                                  PACK6600
      IF(IM.EQ.3)GO TO 1001                                                  PACK6610
      GO TO 3500                                                             PACK6620
C  THE INPUT IS FROM THE RIGHT.                                              PACK6630
3630  IF(IM.LE.3)GO TO 6000                                                  PACK6640
      GO TO 3600                                                             PACK6650
C  THE INPUT IS FROM BELOW.                                                  PACK6660
3640  IF(IM.EQ.0.OR.IM.EQ.3)GO TO 6000                                       PACK6670
      IF(IM.EQ.4.OR.IM.EQ.7)GO TO 3100                                       PACK6680
      GO TO 1001                                                             PACK6690
C  THE INPUT IS FROM THE LOWER LEFT.                                         PACK6700
3650  IF(IM.LE.2)GO TO 6000                                                  PACK6710
      IF(IM.EQ.3)GO TO 1001                                                  PACK6720
```

```
      GO TO 3200                                                        PACK6730
C  THE INPUT IS FROM THE =1 REGISTER.                                   PACK6740
3670  IF(IM.LE.3)GO TO 6000                                             PACK6750
      SEARCH=1;GO TO 1700                                               PACK6760
C                                                                       PACK6770
CC  REGISTER OUTPUT IS IN THE =70 SERIES (TO THE =1 REGISTER).          PACK6780
CC  THIS SEGMENT CAN BE REACHED ONLY FROM A SEARCH THROUGH THE =1 REGISTEPACK6790
C                                                                       PACK6800
3700  IM=MOD(Q(KI,KJ),10)                                               PACK6810
      JM=MOD(Q(KI,KJ),100)/10                                           PACK6820
      KM=Q(KI,KJ)/100                                                   PACK6830
      IF(JM.GT.0.AND.JM.EQ.KM)GO TO 1001                                PACK6840
      IF(JM.GT.7.OR.KM.GT.7.OR.Q(KI,KJ).LT.0)GO TO 1001                 PACK6850
C  SINCE THIS =1 REGISTER CONNECTS TO A =2 REGISTER,                    PACK6860
C  EITHER JM OR KM MUST EQUAL 7.                                        PACK6870
      IF(JM.NE.7.AND.IM.GE.7)IM=JM;JM=KM;KM=IM                          PACK6880
      IF(JM.NE.7)GO TO 1001                                             PACK6890
      GO TO(3710,3720,3730,3740,3750,3760,1001),KM                      PACK6900
C  THE INPUT IS FROM ABOVE.                                             PACK6910
3710  IF(IM.LE.3.AND.IM.NE.2)GO TO 6000                                 PACK6920
      IF(IM.NE.6.AND.IM.NE.9)GO TO 3400                                 PACK6930
      GO TO 1001                                                        PACK6940
C  THE INPUT IS FROM THE UPPER RIGHT.                                   PACK6950
3720  IF(IM.LE.2)GO TO 6000                                             PACK6960
      IF(IM.EQ.3)GO TO 1001                                             PACK6970
      GO TO 3500                                                        PACK6980
C  THE INPUT IS FROM THE RIGHT.                                         PACK6990
3730  IF(IM.LE.3.AND.IM.NE.1)GO TO 6000                                 PACK7000
      IF(IM.NE.5.AND.IM.NE.8)GO TO 3600                                 PACK7010
      GO TO 1001                                                        PACK7020
C  THE INPUT IS FROM BELOW.                                             PACK7030
3740  IF(IM.LE.3.AND.IM.NE.2)GO TO 6000                                 PACK7040
      IF(IM.NE.6.AND.IM.NE.9)GO TO 3100                                 PACK7050
      GO TO 1001                                                        PACK7060
C  THE INPUT IS FROM THE LOWER LEFT.                                    PACK7070
3750  IF(IM.LE.2)GO TO 6000                                             PACK7080
      IF(IM.EQ.3)GO TO 1001                                             PACK7090
      GO TO 3200                                                        PACK7100
C  THE INPUT IS FROM THE LEFT.                                          PACK7110
3760  IF(IM.LE.3.AND.IM.NE.1)GO TO 6000                                 PACK7120
      GO TO 3300                                                        PACK7130
C                                                                       PACK7140
CC  ERROR MESSAGE SET.                                                  PACK7150
C                                                                       PACK7160
1001  WRITE(108,66)TEMP,I,J,REG(FLAG),KI,KJ,REG(SEAROH)                PACK7170
66     FORMAT('0ERROR IN LOADING CONTROL REGISTER'/1X,20A4/             PACK7180
     $' SEEKING SOURCE FOR LOCATION',I3,',',I2,' OF REGISTER SET',A4/   PACK7190
     $' ERROR ENCOUNTERED AT LOCATION',I3,',',I2,' OF REGISTER SET',A4/)PACK7200
```

```
      GO TO 5                                                                  PACK7210
C                                                                              PACK7220
C   SET THE CANCELLATION NUMBER AND SOURCE REGISTER (H1).  ALSO SET THE        PACK7230
C   ROW AND COLUMN FOR REGISTER =1 (H2 + H3) AND FOR REGISTER =2 (H4 + H5      PACK7240
C                                                                              PACK7250
6000  IF(FLAG=1)1001,6001,6002                                                 PACK7260
6001  H1=(SEARCH=1)*2+8                                                        PACK7270
      H2=KI=1                                                                  PACK7280
      H3=KJ=1                                                                  PACK7290
      GO TO 2000                                                               PACK7300
6002  H1=H1+(SEARCH=1)+4                                                       PACK7310
      H4=KI=1                                                                  PACK7320
      H5=KJ=1                                                                  PACK7330
C                                                                              PACK7340
C   SET UP THE CODE WORD PARTS FOR THE CANCELLATION CODE (H8)                  PACK7350
C   AND THE OP CODE (H6 + H7).                                                 PACK7360
C                                                                              PACK7370
C                                                                              PACK7380
C     H8 IS UNUSED FIELD OF MRC WHICH ALLOWS FOR FUTURE EXPANSION
4000  H8=0
C
      IF(OP(I,J).EQ.3H   )H6=H7=0   )GO TO 55                                  PACK7390
      IF(OP(I,J).EQ.3HNOP)H6=H7=0   )GO TO 55                                  PACK7400
      IF(OP(I,J).EQ.3HADD)H6=0)H7=1 )GO TO 55                                  PACK7410
      IF(OP(I,J).EQ.3HSUB)H6=0)H7=2 )GO TO 55                                  PACK7420
      IF(OP(I,J).EQ.3HMPY)H6=0)H7=3 )GO TO 55                                  PACK7430
      IF(OP(I,J).EQ.3HDIV)H6=0)H7=4 )GO TO 55                                  PACK7440
      IF(OP(I,J).EQ.3HSLI)H6=0)H7=5 )GO TO 55                                  PACK7450
      IF(OP(I,J).EQ.3HSRI)H6=0)H7=6 )GO TO 55                                  PACK7460
      IF(OP(I,J).EQ.3HAND)H6=0)H7=7)GO TO 55                                   PACK7470
      IF(OP(I,J).EQ.3HIOR)H6=0)H7=8)GO TO 55                                   PACK7480
      IF(OP(I,J).EQ.3HXOR)H6=0)H7=9)GO TO 55                                   PACK7490
      IF(OP(I,J).EQ.3HREAD)H6=0)H7=10)GO TO 55                                 PACK7500
      IF(OP(I,J).EQ.3HFSB)H6=0)H7=11)GO TO 55                                  PACK7510
      IF(OP(I,J).EQ.3HFMY)H6=0)H7=12)GO TO 55                                  PACK7520
      IF(OP(I,J).EQ.3HFDV)H6=0)H7=13)GO TO 55                                  PACK7530
      IF(OP(I,J).EQ.3HCPM)H6=0)H7=14)GO TO 55                                  PACK7540
      IF(OP(I,J).EQ.3HNOT)H6=0)H7=15)GO TO 55                                  PACK7550
      WRITE(108,7000)TEMP                                                      PACK7560
 7000 FORMAT('0ILLEGAL OPERATION CODE'/1X,20A4)                                PACK7570
C                                                                              PACK7580
C   PACK THE CONTROL WORD FOUR BITS AT A TIME (IN HEXADECIMAL).                PACK7590
C                                                                              PACK7600
55    TR(I,J)=H8*16*(H5*16*(H4*16*(H3*16*(H2*16*                               PACK7610
     $ (H1*16*(H7*16*H6))))))                                                  PACK7620
  5   CONTINUE                                                                 PACK7630
C*********DEBUGGING*****************                                          PACK7640
      WRITE(108,8000)                                                          PACK7650
```

```
 8000  FORMAT(1HODEBUGGING CHECK ON TR REGISTER GENERATED BY PACK1)      PACK7660
       DO 8001 I=IR,JR                                                  PACK7670
       WRITE(108,8002) (TR(I,J),J=IC,JC)                                PACK7680
 8002  FORMAT(1H0,1X/((8(4X,Z8))/))                                    PACK7690
 8001  CONTINUE                                                         PACK7700
C**********DEBUGGING*****************                                  PACK7710
       RETURN                                                           PACK7720
       END                                                              PACK7730
```

```
C  $$$$
```

```
C  $$$$
```

```
C  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$                                  SBNZ00000
      SUBROUTINE SBNZ(IR,JR,IC,JC,CPA,CPQ,MRC,TEMP,II,SOURCE)              SBNZ00010
      INTEGER MRC(16,16,16),TEMP(20),MASK(16,16),SOURCE                    SBNZ00020
      REAL CPA(16,16),CPQ(16,16)                                           SBNZ00030
C*********** DEBUGGING ***************************                       SBNZ00040
      WRITE(108,999)TEMP                                                   SBNZ00050
  999 FORMAT(1H1,1X,20A4)                                                  SBNZ00060
C                                                                          SBNZ00070
      DECODE(80,1,TEMP)CN1,CN2                                             SBNZ00080
    1 FORMAT(I2,I2,1X,I2)                                                  SBNZ00090
      IF(CN1.EQ.15.AND.CN2.EQ.15)GO TO 100                                 SBNZ00100
      IF(15-CN1)7,7,8                                                      SBNZ00110
    8 CALL SMASK(IR,JR,IC,JC,CN1,MRC,MASK,II)                              SBNZ00120
      DO 5 I=IR,JR                                                         SBNZ00130
      DO 5 J=IC,JC                                                         SBNZ00140
      IF(MASK(I,J))5,5,6                                                   SBNZ00150
    6 IF(INT(CPA(I,J)))100,5,100                                           SBNZ00160
    5 CONTINUE                                                             SBNZ00170
    7 IF(15-CN2)17,17,18                                                   SBNZ00180
   17 RETURN SOURCE                                                        SBNZ00190
   18 CALL SMASK(IR,JR,IC,JC,CN2,MRC,MASK,II)                              SBNZ00200
      DO 15 I=IR,JR                                                        SBNZ00210
      DO 15 J=IC,JC                                                        SBNZ00220
      IF(MASK(I,J))15,15,16                                                SBNZ00230
   16 IF(INT(CPQ(I,J)))100,15,100                                          SBNZ00240
   15 CONTINUE                                                             SBNZ00250
      RETURN SOURCE                                                        SBNZ00260
  100 RETURN                                                               SBNZ00270
      END                                                                  SBNZ00280
```

```
C  $$$$
```

```
15      CONTINUE                                                          SRCH0480
        IF(FLAGS.EQ.1)GO TO 25                                            SRCH0490
        IF(FLAGS.LT.1)GO TO 5                                             SRCH0500
        DO 6 J=IR,JR                                                      SRCH0510
        DO 6 K=IC,JC                                                      SRCH0520
6       MASK(J,K)=IMASK(J,K)                                              SRCH0530
5       CONTINUE                                                          SRCH0540
        DO 65 J=IR,JR                                                     SRCH0550
        DO 65 K=IC,JC                                                     SRCH0560
        IF(MASK(J,K))65,65,36                                             SRCH0570
65      CONTINUE                                                          SRCH0580
25      DO 35 J=IR,JR                                                     SRCH0590
        DO 35 K=IC,JC                                                     SRCH0600
        IF(IMASK(J,K))35,35,36                                            SRCH0610
35      CONTINUE                                                          SRCH0620
        WRITE(108,66)TEMP                                                 SRCH0630
66      FORMAT(1ERROR ENCOUNTERED IN EXECUTING THE 3-D INSTRUCTION',1X,   SRCH0640
       $20A4)                                                             SRCH0650
        RETURN                                                            SRCH0660
36      JMAX=J                                                            SRCH0670
        KMAX=K                                                            SRCH0680
        RETURN                                                            SRCH0690
        END                                                               SRCH0700
LOAD (EF,(SUBP),(TEMP)),(LMN,SIM3DMIX),(PERM),(MAP)
```

```
!RUN (LMNASIM3DMIX)
!DATA
C    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
C    3D PROGRAM    DECK
C
!EOD
!EOD
```

```
!ASSIGN M:SI,(FILE,SIMLOAD),(IN)
!ASSIGN F:110,(FILE,SIMPROG),(IN)
!ASSIGN F:105,(FILE,DATA),(IN)
!FORTRAN LS,GO,S
!LOAD (GO),(EF,(SUBP)),(EXEC),(MAP)
```

APPENDIX D

A PARALLEL MATRIX TRANSPOSE ALGORITHM (MAT) AS IMPLEMENTED ON SIM3DMIX

```
C    A MATRIX TRANSPOSE ALGORITHM FOR THE 3D MACHINE
C    EXECUTION TIME IS INDEPENDENT ON THE SIZE OF THE
C    ARRAY FOR A 4 X 4 OR LESS
C    OTHERWISE IT IS PROPORTIONAL TO N FOR AN N X N
C    NOTICE THAT AN N**2 FACTOR IS TYPICAL FOR SERIAL
C    MACHINES
C    THIS PROGRAM HAPPENED TO BE MADE OF ALL 3D INSTRUCTIONS
C    NOTICE THE SIMPLICITY OF THE PROGRAM
C    ALTHOUGH OF THE COMPLIXITY OF THE CONNECTION PATTERN
C    IT IS A PART OF A READ ONLY STORE WHICH HAS TO BE
C    WRITTEN ONCE AND ONLY ONCE
(01,04:01,04)    INPT(00,00)      MC (01)
                 INPT(00,15)      MR (01)
                 CTRN             MR1(01)          MR1(01)
                 OUTP(00,15)      MR1(01)
                 STOP

437 000 NOP          571 757 NOP          571 757 NOP          670 747 NOP
722 277 NOP          003 003 NOP          570 757 NOP          570 759 NOP
722 277 NOP          720 277 NOP          003 003              570 769
710 377 NOP          720 278 NOP          720 278 NOP          000 167 NOP
11.       12.        13.       14.        21.       22.        23.        24.

31.       32.        33.       34.        41.       42.        43.
  EOD
```

```
      INTEGER CR(10,16,16),TEMP(20),INST(20,200),MRC(16,16,16)
      INTEGER SCOUNT,PCOUNT,TR(16,16)
      REAL MR1(16,16,16),MR2(16,16,16)
      REAL CPA(16,16),CPQ(16,16)
      COMMON SCOUNT,PCOUNT
      SCOUNT=0
      PCOUNT=0
      DO 4 I=1,16
      DO 4 J=1,16
      DO 4 K=1,16
      MRC(I,J,K)=0
      MR1(I,J,K)=0.
      MR2(I,J,K)=0.
      CPA(J,K)=0.
      CPQ(J,K)=0.
4     CONTINUE
      DO 5 I=1,20
      DO 5 J=1,200
      INST(I,J)=0
5     CONTINUE
      DO 6 I=1,10
      DO 6 J=1,16
      DO 6 K=1,16
6     CR(I,J,K)=0
      MINDX=   23
      DO 13 J=1,MINDX
      READ(110,14)(INST(I,J),I=1,20)
14    FORMAT(20A4)
13    WRITE(108,15)(INST(I,J),I=1,20)
15    FORMAT(1X,20A4)
C     A MATRIX TRANSPOSE ALGORITHM FOR THE 3D MACHINE
C     EXECUTION TIME IS INDEPENDENT ON THE SIZE OF THE
C     ARRAY FOR A 4 X 4 OR LESS
C     OTHERWISE IT IS PROPORTIONAL TO N FOR AN N X N
C     NOTICE THAT AN N**2 FACTOR IS TYPICAL FOR SERIAL
C     MACHINES
C     THIS PROGRAM HAPPENED TO BE MADE OF ALL 3D INSTRUCTIONS
C     NOTICE THE SIMPLICITY OF THE PROGRAM
C     ALTHOUGH OF THE COMPLIXITY OF THE CONNECTION PATTERN
C     IT IS A PART OF A READ ONLY STORE WHICH HAS TO BE
C     WRITTEN ONCE AND ONLY ONCE
      CALL SINPT( 1, 4, 1, 4,INST(1,   12),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 4, 1, 4,INST(1,   13),MR1,MR2,MRC,CR,TR)
      CALL SCTRN( 1, 4, 1, 4,INST(1,   14),MR1,MR2,MRC)
      CALL SOUTP( 1, 4, 1, 4,INST(1,   15),MR1,MR2)
9999  WRITE(108,999)
999   FORMAT('0STOP INSTRUCTION ENCOUNTERED!')
      WRITE(108,998)
```

```
998   FORMAT(1H1)
      CALL EXIT
      END
```

```
 (01,04:01,04)   INPT(00,00)      MC (01)
DEBUGGING CHECK ON ARRAY INDEXES
IR= 1     JR= 4     IC= 1     JC= 4     II= 1
REGISTER CONNECTION
  A     Q     OP        A     Q     OP        A     Q     OP        A     Q     OP
 437     0   NOP       571   757   NOP       571   757   NOP       670   747   NOP
 722   277   NOP         3     3   NOP       570   757   NOP       570   759   NOP
 722   277   NOP       720   277   NOP         3     3               570   759
 710   377   NOP       720   278   NOP       720   278   NOP         0   167   NOP
DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
    00100000     00910010     00920020     00930030
    00901100     00111110     00921120     00931130
    00902200     00912210     00122220     00932230
    00903300     00913310     00923320     00133330
                  INPT(00,15)      MR (01)
DEBUGGING CHECK ON ARRAY INDEXES
IR= 1     JR= 4     IC= 1     JC= 4     II= 1
DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          16
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           1
DEBUGGING CHECK ON RTEMP DURING SINPT
        11.000000       12.000000       13.000000       14.000000
        21.000000       22.000000       23.000000       24.000000
        31.000000       32.000000       33.000000       34.000000
        41.000000       42.000000       43.000000          .000000
DEBUGGING CHECK OF MR1( 1) REGISTERS AFTER SINPT
        11.000000       12.000000       13.000000       14.000000
        21.000000       22.000000       23.000000       24.000000
        31.000000       32.000000       33.000000       34.000000
        41.000000       42.000000       43.000000          .000000
                  CTRN                MR1(01)          MR1(01)
DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
DEBUGGING CHECK ON CON(1,I,J) REGISTERS
```

```
          1          2          3          4
          1          2          3          4
          1          2          3          4
          1          2          3          4
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          1          1          1          1
          2          2          2          2
          3          3          3          3
          4          4          4          4
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         32
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          2
                    OUTP(00,15)     MR1(01)
 CONTENTS OF MR1 REGISTERS OF PLANE  1
     .1100000E 02    .2100000E 02    .3100000E 02    .4100000E 02
     .1200000E 02    .2200000E 02    .3200000E 02    .4200000E 02
     .1300000E 02    .2300000E 02    .3300000E 02    .4300000E 02
     .1400000E 02    .2400000E 02    .3400000E 02    .0000000E 00
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         48
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          3
 STOP INSTRUCTION ENCOUNTERED
1
```

APPENDIX E

A PARALLEL MATRIX MULTIPLICATION ALGORITHM (MMA) AS IMPLEMENTED ON SIM3DMIX

```
C   MATRIX MULTIPLICATION ALGORITHM
C   TIME OF EXECUTION IS PROPORTIONAL TO N
C   TIME OF EXECUTION IN SERIAL MACHINES IS PROPORTIONAL
C   TO THE CUBIC POWER OF N
(01:05)(01:05)    INPT          CR (01)
                  INPT          CR (02)
                  INPT          MR (01)
                  TRAN          CR (01)        AC (02)
F       DO 25 I=1,5
                  TRAN          MR (01)        AR (02)
                  CTRN          MR (02)        AR (02)
                  TRAN(00,00,16) MR (02)       CP                FMY
                  TRAN(15,00,16) MR (03)       CP                FAD
                  TRAN(00,15)   CP             AR (03)
                  OUTP          MR (03)
                  DTRN          MR1(03)        MR2(03)
                  TRAN          CR (02)        AC (01)
                  CTRN          MR (01)        AR (01)
F   25 CONTINUE
                  OUTP          MR (03)
                  STOP
030 040                 634 040         634 040         634 040
600 040                 030 144         634 144         634 144
634 144                 600 144         030 144         634 144
634 144                 634 144         600 144         030 144
634 144                 634 144         634 144         600 144
030 100                 634 100         634 100         634 100
600 100
300 400                 360 400         360 400         360 400
060 400                 300 410         360 410         360 410
360 410                 060 410         300 410         360 410
360 410                 360 410         060 410         300 410
360 410                 360 410         360 410         060 410
300 010                 360 010         360 010         360 010
060 010
11.       12.       13.       14.       15.       21.       22.       23.
24.       25.       31.       32.       33.       34.       35.       41.
42.       43.       44.       45.       51.       52.       53.       54.
55.
11.       21.       31.       41.       51.       12.       22.       32.
42.       52.       13.       23.       33.       43.       53.       14.
24.       34.       44.       54.       15.       25.       35.       45.
55.
 EOD
      INTEGER CR(10,16,16),TEMP(20),INST(20,200),MRC(16,16,16)
      INTEGER SCOUNT,PCOUNT,TR(16,16)
      REAL MR1(16,16,16),MR2(16,16,16)
      REAL CPA(16,16),CPQ(16,16)
```

```
      COMMON SCOUNT,PCOUNT
      SCOUNT=0
      PCOUNT=0
      DO 4 I=1,16
      DO 4 J=1,16
      DO 4 K=1,16
      MRC(I,J,K)=0
      MR1(I,J,K)=0.
      MR2(I,J,K)=0.
      CRA(J,K)=0.
      CRQ(J,K)=0.
4     CONTINUE
      DO 5 I=1,20
      DO 5 J=1,200
      INST(I,J)=0
5     CONTINUE
      DO 6 I=1,10
      DO 6 J=1,16
      DO 6 K=1,16
6     CR(I,J,K)=0
      MINDX=   44
      DO 13 J=1,MINDX
      READ(110,14)(INST(I,J),I=1,20)
14    FORMAT(20A4)
13    WRITE(108,15)(INST(I,J),I=1,20)
15    FORMAT(1X,20A4)
C   MATRIX MULTIPLICATION ALGORITHM
C   TIME OF EXECUTION IS PROPORTIONAL TO N
C   TIME OF EXECUTION IN SERIAL MACHINES IS PROPORTIONAL
C   TO THE CUBIC POWER OF N
      CALL SINPT( 1, 5, 1, 5,INST(1,     5),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 5, 1, 5,INST(1,     6),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 5, 1, 5,INST(1,     7),MR1,MR2,MRC,CR,TR)
      CALL STRAN( 1, 5, 1, 5,INST(1,     8),MR1,MR2,MRC,CRA,CRQ,CR)
      DO 25 I=1,5
      CALL STRAN( 1, 5, 1, 5,INST(1,    10),MR1,MR2,MRC,CRA,CRQ,CR)
      CALL SCTRN( 1, 5, 1, 5,INST(1,    11),MR1,MR2,MRC)
      CALL STRAN( 1, 5, 1, 5,INST(1,    12),MR1,MR2,MRC,CRA,CRQ,CR)
      CALL STRAN( 1, 5, 1, 5,INST(1,    13),MR1,MR2,MRC,CRA,CRQ,CR)
      CALL STRAN( 1, 5, 1, 5,INST(1,    14),MR1,MR2,MRC,CRA,CRQ,CR)
      CALL SOUTP( 1, 5, 1, 5,INST(1,    15),MR1,MR2)
      CALL SOTRN( 1, 5, 1, 5,INST(1,16),MR1,MR2,MRC)
      CALL STRAN( 1, 5, 1, 5,INST(1,    17),MR1,MR2,MRC,CRA,CRQ,CR)
      CALL SCTRN( 1, 5, 1, 5,INST(1,    18),MR1,MR2,MRC)
   25 CONTINUE
      CALL SOUTP( 1, 5, 1, 5,INST(1,   20),MR1,MR2)
9999  WRITE(108,999)
999   FORMAT('0STOP INSTRUCTION ENCOUNTERED!')
```

```
      WRITE(108,998)
998   FORMAT(1H1)
      CALL EXIT
      END
 (01:05/01:05)   INPT                    CR (01)
 DEBUGGING CHECK ON ARRAY INDEXES
IR= 1      JR= 5      IC= 1      JC= 5      II= 1
 REGISTER CONNECTION
   A     Q      OP          A     Q      OP          A     Q      OP          A     Q      OP
  30    40             634   40             634   40             634   40
 600    40              30  144             634  144             634  144
 634   144             600  144              30  144             634  144
 634   144             634  144             600  144              30  144
 634   144             634  144             634  144             600  144
  30   100             634  100             634  100             634  100
 600   100
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
    00100000     00900010     00900020     00900030     00900040
    00510000     00D10010     00D10020     00D10030     00D10040
    00520000     00D20010     00D20020     00D20030     00D20040
    00530000     00D30010     00D30020     00D30030     00D30040
    00540000     00D40010     00D40020     00D40030     00D40040
                   INPT                    CR (02)
 DEBUGGING CHECK ON ARRAY INDEXES
IR= 1      JR= 5      IC= 1      JC= 5      II= 2
 REGISTER CONNECTION
   A     Q      OP          A     Q      OP          A     Q      OP          A     Q      OP
 300   400             360   400             360   400             360   400
  60   400             300   410             360   410             360   410
 360   410              60   410             300   410             360   410
 360   410             360   410              60   410             300   410
 360   410             360   410             360   410              60   410
 300    10             360    10             360    10             360    10
  60    10
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
```

```
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
    00D01100     00D02110     00D03120     00D04130     00504140
    00D11200     00D12210     00D13220     00D14230     00514240
    00D21300     00D22310     00D23320     00D24330     00524340
    00D31400     00D32410     00D33420     00D34430     00534440
    00941400     00942410     00943420     00944430     00144440
                   INPT                    MR (01)
 DEBUGGING CHECK ON ARRAY INDEXES
IR= 1     JR= 5     IC= 1     JC= 5     II= 1
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         25

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          1
 DEBUGGING CHECK ON RTEMP DURING SINPT
        11.000000      12.000000      13.000000      14.000000      15.000000
        21.000000      22.000000      23.000000      24.000000      25.000000
        31.000000      32.000000      33.000000      34.000000      35.000000
        41.000000      42.000000      43.000000      44.000000      45.000000
        51.000000      52.000000      53.000000      54.000000      55.000000
 DEBUGGING CHECK OF MR1( 1) REGISTERS AFTER SINPT
        11.000000      12.000000      13.000000      14.000000      15.000000
        21.000000      22.000000      23.000000      24.000000      25.000000
        31.000000      32.000000      33.000000      34.000000      35.000000
        41.000000      42.000000      43.000000      44.000000      45.000000
        51.000000      52.000000      53.000000      54.000000      55.000000
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         50
```

```
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          2
 DEBUGGING CHECK OF MR2( 1) REGISTERS AFTER SINPT
       11.000000       21.000000       31.000000       41.000000       51.000000
       12.000000       22.000000       32.000000       42.000000       52.000000
       13.000000       23.000000       33.000000       43.000000       53.000000
       14.000000       24.000000       34.000000       44.000000       54.000000
       15.000000       25.000000       35.000000       45.000000       55.000000
                    TRAN                    OR (01)             MC (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         75

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          3
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN

    00100000    00900010    00900020    00900030    00900040

    00510000    00D10010    00D10020    00D10030    00D10040

    00520000    00D20010    00D20020    00D20030    00D20040

    00530000    00D30010    00D30020    00D30030    00D30040

    00540000    00D40010    00D40020    00D40030    00D40040
                    TRAN                    MR (01)             MR (02)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        100

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          4
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
                    CTRN                    MR (02)             MR (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        125
```

```
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          5
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         150

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          6
                      TRAN(00,00,16)  MR (02)          CP                 RMY
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         175

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          7
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         200

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          8
 DEBUGGING CHECK OF MASK MATRIX
          1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         225

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          9
                      TRAN(15,00,16)  MR (03)          CP                 FAD
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         250

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
```

```
        10
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        275

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        11
                      TRAN(00,15)      OP                MR (03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        300

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        12
                     OUTP              MR (03)
 CONTENTS OF MR1 REGISTERS OF PLANE  3
     .1210000E 03     .2310000E 03     .3410000E 03     .4510000E 03     .5610000E 03
     .2310000E 03     .4410000E 03     .6510000E 03     .8610000E 03     .1071000E 04
     .3410000E 03     .6510000E 03     .9610000E 03     .1271000E 04     .1581000E 04
     .4510000E 03     .8610000E 03     .1271000E 04     .1681000E 04     .2091000E 04
     .5610000E 03     .1071000E 04     .1581000E 04     .2091000E 04     .2601000E 04
 CONTENTS OF MR2 REGISTERS OF PLANE  3
     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00
     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00
     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00
     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00
     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00     .0000000E 00
                     DTRN              MR1(03)          MR2(03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
```

```
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        325

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          13
                    TRAN                GR (02)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        350

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          14
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00D01100    00D02110    00D03120    00D04130    00504140
    00D11200    00D12210    00D13220    00D14230    00514240
    00D21300    00D22310    00D23320    00D24330    00524340
    00D31400    00D32410    00D33420    00D34430    00534440
    00941400    00942410    00943420    00944430    00144440
                 CTRN               MR (01)          MR (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        375

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          15
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        400

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          16
                    TRAN                MR (01)          MR (02)
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        425

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          17
 DEBUGGING CHECK OF MASK MATRIX
```

```
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
                    CTRN                MR (02)          MR (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          450

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           18
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          475

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           19
                    TRAN(00,00,16)  MR (02)          CP               RMY
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          500

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           20
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          525

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           21
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          550

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           22
                    TRAN(15,00,16)  MR (03)          CP               RAD
 DEBUGGING CHECK OF MASK MATRIX
```

```
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         575

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          23
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         600

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          24
                    TRAN(00,15)      CP              MR (03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         625

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          25
                    OUTP             MR (03)
 CONTENTS OF MR1 REGISTERS OF PLANE  3
     .265000E 03     .495000E 03     .725000E 03     .955000E 03     .118500E 04
     .495000E 03     .925000E 03     .135500E 04     .178500E 04     .221500E 04
     .725000E 03     .135500E 04     .198500E 04     .261500E 04     .324500E 04
     .955000E 03     .178500E 04     .261500E 04     .344500E 04     .427500E 04
     .118500E 04     .221500E 04     .324500E 04     .427500E 04     .530500E 04
 CONTENTS OF MR2 REGISTERS OF PLANE  3
     .121000E 03     .231000E 03     .341000E 03     .451000E 03     .561000E 03
     .231000E 03     .441000E 03     .651000E 03     .861000E 03     .107100E 04
     .341000E 03     .651000E 03     .961000E 03     .127100E 04     .158100E 04
```

```
    .451000E 03     .861000E 03     .127100E 04     .168100E 04     .209100E 04
    .561000E 03     .107100E 04     .158100E 04     .209100E 04     .260100E 04
                    DTRN            MR1(03)         MR2(03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        650

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         26
                    TRAN            CR (02)         MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        675

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         27
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00D01100    00D02110    00D03120    00D04130    00504140
    00D11200    00D12210    00D13220    00D14230    00514240
    00D21300    00D22310    00D23320    00D24330    00524340
    00D31400    00D32410    00D33420    00D34430    00534440
    00941400    00942410    00943420    00944430    00144440
                CTRN            MR (01)         MR (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        700

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         28
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        725

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         29
                    TRAN            MR (01)         MR (02)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
```

```
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        750

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          30
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
                    CTRN                MR (02)          MR (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        775

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          31
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        800

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          32
                    TRAN(00,00,16)   MR (02)          CP                FMY
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        825

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          33
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        850

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          34
```

```
 DEBUGGING CHECK OF MASK MATRIX
    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       875

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
       35
                    TRAN(15,00,16)  MR (03)          CP                FAD
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       900

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
       36
 DEBUGGING CHECK OF MASK MATRIX
    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       925

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
       37
                    TRAN(00,15)     CP                MR (03)
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       950

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
       38
                    OUTP              MR (03)
 CONTENTS OF MR1 REGISTERS OF PLANE  3
    .434000E 03    .794000E 03    .115400E 04    .151400E 04    .187400E 04
    .794000E 03    .145400E 04    .211400E 04    .277400E 04    .343400E 04
    .115400E 04    .211400E 04    .307400E 04    .403400E 04    .499400E 04
    .151400E 04    .277400E 04    .403400E 04    .529400E 04    .655400E 04
```

```
    .187400E 04     .343400E 04     .499400E 04     .655400E 04     .811400E 04
 CONTENTS OF MR2 REGISTERS OF PLANE  3
    .265000E 03     .495000E 03     .725000E 03     .955000E 03     .118500E 04
    .495000E 03     .925000E 03     .135500E 04     .178500E 04     .221500E 04
    .725000E 03     .135500E 04     .198500E 04     .261500E 04     .324500E 04
    .955000E 03     .178500E 04     .261500E 04     .344500E 04     .427500E 04
    .118500E 04     .221500E 04     .324500E 04     .427500E 04     .530500E 04
                  DTRN                MR1(03)         MR2(03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       975

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        39
                  TRAN              CR (02)         MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1000

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        40
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00D01100     00D02110     00D03120     00D04130     00504140
    00D11200     00D12210     00D13220     00D14230     00514240
    00D21300     00D22310     00D23320     00D24330     00524340
    00D31400     00D32410     00D33420     00D34430     00534440
    00941400     00942410     00943420     00944430     00144440
                  CTRN              MR (01)         MR (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1025

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        41
```

```
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1050

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        42
                        TRAN                MR (01)          MR (02)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1075

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        43
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
                        CTRN                MR (02)          MR (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1100

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        44
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1125

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        45
                        TRAN(00,00,16)  MR (02)          CP               FMY
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1150

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        46
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
```

```
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        1175

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          47
 DEBUGGING CHECK OF MASK MATRIX
           1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        1200

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          48
                    TRAN(15,00,16)  MR (03)          CP                FAD
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        1225

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          49
 DEBUGGING CHECK OF MASK MATRIX
           1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        1250

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          50
                    TRAN(00,15)      CP               MR (03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        1275

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          51
                    OUTP             MR (03)
```

```
 CONTENTS OF MR1 REGISTERS OF PLANE  3
    .630000E 03     .113000E 04     .163000E 04     .213000E 04     .263000E 04
    .113000E 04     .203000E 04     .293000E 04     .383000E 04     .473000E 04
    .163000E 04     .293000E 04     .423000E 04     .553000E 04     .683000E 04
    .213000E 04     .383000E 04     .553000E 04     .723000E 04     .893000E 04
    .263000E 04     .473000E 04     .683000E 04     .893000E 04     .110300E 05
 CONTENTS OF MR2 REGISTERS OF PLANE  3
    .434000E 03     .794000E 03     .115400E 04     .151400E 04     .187400E 04
    .794000E 03     .145400E 04     .211400E 04     .277400E 04     .343400E 04
    .115400E 04     .211400E 04     .307400E 04     .403400E 04     .499400E 04
    .151400E 04     .277400E 04     .403400E 04     .529400E 04     .655400E 04
    .187400E 04     .343400E 04     .499400E 04     .655400E 04     .811400E 04
                      DTRN                MR1(03)          MR2(03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1300
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        52
                    TRAN                CR (02)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1325
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        53
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00D01100     00D02110     00D03120     00D04130     00504140
    00D11200     00D12210     00D13220     00D14230     00514240
    00D21300     00D22310     00D23320     00D24330     00524340
```

```
    00D31400     00D32410     00D33420     00D34430     00534440

    00941400     00942410     00943420     00944430     00144440
                   CTRN                    MR (01)              MR (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1350

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        54
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1375

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        55
                   TRAN                    MR (01)              MR (02)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1400

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        56
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
                   CTRN                    MR (02)              MR (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1425

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        57
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1450

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        58
                   TRAN(00,00,16)  MR (02)              CP                   FMY
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
```

```
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1475

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        59
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1500

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        60
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1525

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        61
                    TRAN(15,00,16)  MR (03)          CP                FAD
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
     1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1550

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        62
 DEBUGGING CHECK OF MASK MATRIX
     1
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1575

NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        63
                    TRAN(00,15)     CP                MR (03)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1
     1    1    1    1    1
```

```
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1600
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         64
                      OUTP                MR (03)
 CONTENTS OF MR1 REGISTERS OF PLANE  3
     .8550000E 03     .1505000E 04     .2155000E 04     .2805000E 04     .3455000E 04
     .1505000E 04     .2655000E 04     .3805000E 04     .4955000E 04     .6105000E 04
     .2155000E 04     .3805000E 04     .5455000E 04     .7105000E 04     .8755000E 04
     .2805000E 04     .4955000E 04     .7105000E 04     .9255000E 04     .1140500E 05
     .3455000E 04     .6105000E 04     .8755000E 04     .1140500E 05     .1405500E 05
 CONTENTS OF MR2 REGISTERS OF PLANE  3
     .6300000E 03     .1130000E 04     .1630000E 04     .2130000E 04     .2630000E 04
     .1130000E 04     .2030000E 04     .2930000E 04     .3830000E 04     .4730000E 04
     .1630000E 04     .2930000E 04     .4230000E 04     .5530000E 04     .6830000E 04
     .2130000E 04     .3830000E 04     .5530000E 04     .7230000E 04     .8930000E 04
     .2630000E 04     .4730000E 04     .6830000E 04     .8930000E 04     .1103000E 05
                      DTRN                MR1(03)          MR2(03)
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1625
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         65
                      TRAN                CR (02)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1650
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         66
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
     00D01100     00D02110     00D03120     00D04130     00504140
     00D11200     00D12210     00D13220     00D14230     00514240
     00D21300     00D22310     00D23320     00D24330     00524340
     00D31400     00D32410     00D33420     00D34430     00534440
     00941400     00942410     00943420     00944430     00144440
                      CTRN                MR (01)          MR (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1675
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         67
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
```

```
      1700
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        68
                    OUTP                    MR (03)
 CONTENTS OF MR1 REGISTERS OF PLANE  3
     .8550000E 03     .1505000E 04     .2155000E 04     .2805000E 04     .3455000E 04
     .1505000E 04     .2655000E 04     .3805000E 04     .4955000E 04     .6105000E 04
     .2155000E 04     .3805000E 04     .5455000E 04     .7105000E 04     .8755000E 04
     .2805000E 04     .4955000E 04     .7105000E 04     .9255000E 04     .1140500E 05
     .3455000E 04     .6105000E 04     .8755000E 04     .1140500E 05     .1405500E 05
 CONTENTS OF MR2 REGISTERS OF PLANE  3
     .8550000E 03     .1505000E 04     .2155000E 04     .2805000E 04     .3455000E 04
     .1505000E 04     .2655000E 04     .3805000E 04     .4955000E 04     .6105000E 04
     .2155000E 04     .3805000E 04     .5455000E 04     .7105000E 04     .8755000E 04
     .2805000E 04     .4955000E 04     .7105000E 04     .9255000E 04     .1140500E 05
     .3455000E 04     .6105000E 04     .8755000E 04     .1140500E 05     .1405500E 05
 STOP INSTRUCTION ENCOUNTERED
```

APPENDIX F

A PARALLEL MATRIX INVERSION ALGORITHM ACCORDING TO THE GAUSS-GORDAN ELIMINATION PRINCIPLE AS IMPLEMENTED ON SIM3DMIX

```
C   SIM3D-FORTRAN MIX (SIM3DMIX)
C   GAUSSIAN ELIMINATION ON A 3 BY 3
C   FIRST TRY
F       I=1
(01,03,01,07)    INPT(00,15)        MR (01)
                 INPT               CR (08)
(01,01,01,07)    INPT               CR (01)
(01,03,01,07)    INPT               CR (02)
(02,02,01,07)    INPT               CR (03)
(01,03,01,02)    INPT               CR (04)
(01,02,01,07)    INPT               CR (05)
(01,03,01,07)    INPT               CR (06)
(01,03,01,03)    INPT               CR (07)
(01,01,01,07)    TRAN               CR (01)        MC (01)                  415
                 CTRN               MR1(01)        MR2(01)
(01,03,01,07)    TRAN               CR (02)        MC (01)
                 CTRN               MR2(01)        MR2(01)
(01,01,01,07)    TRAN(00,00,16)     MR (01)        CP                FDV
                 TRAN(00,15)        CP             MR (01)
(01,03,01,07)    OUTP               MR1(01)
                 DTRN               MR1(01)        MR2(01)
                 TRAN(00,15)        MR (01)        MR (02)
(02,03,01,07)    TRAN               CR (03)        MC (02)
                 CTRN               MR1(02)        MR2(02)
(01,03,01,07)    TRAN               CR (08)        MC (02)
                 CTRN               MR1(02)        MR1(02)
(02,03,01,07)    TRAN(00,00,16)     MR (02)        CP                FMY
                 TRAN(15,00,16)     MR (01)        CP                FSB
                 TRAN(15,15,16)     MR (01)        CP                CPM
                 TRAN(00,15)        CP             MR (01)
(01,03,01,07)    OUTP               MR1(01)
F       I=I+1
F       IF(I.EQ.3)GO TO 115
F       IF(I.EQ.4)GO TO 215
(01,03,01,02)    TRAN               CR (04)        MC (01)
                 CTRN               MR1(01)        MR1(01)
(01,02,01,07)    TRAN               CR (05)        MC (01)
                 CTRN               MR1(01)        MR1(01)
                 BRNU               415
(01,03,01,07)    TRAN               CR (06)        MC (01)                  115
                 CTRN               MR1(01)        MR1(01)
(01,03,01,03)    TRAN               CR (07)        MC (01)
                 CTRN               MR1(01)        MR1(01)
                 BRNU               415
(01,03,01,07)    OUTP               MR1(01)                                 215
                 STOP
   2.       7.       4.       9.       1.       0.       0.       1.
   9.       6.       1.       0.       1.       0.       3.       8.
```

```
 5.      6.      0.      0.      1.
040 000 040 000 040 000 040 000 040 000
040 000 040 000 040 000 144 000 144 000
144 000 144 000 144 000 144 000 144 000
144 000 144 000 100 000 100 000 100 000
100 000 100 000 100 000 100 000 100 000
100 000
070 734 000 634 000 634 000 634 000 634
000 634 000 634 000 600 000 600 070 734
000 040 000 040 000 040 000 040 000 634
000 040 000 040 000 040 000 040 000 040
000 144 000 144 000 144 000 144 000 144
000 144 000 144 000 100 000 100 000 100
000 100 000 100 000 100 000 100 000 100
000 100
070 734 000 634 000 634 000 634 000 634
000 634 000 634 000 600 000 600 070 734
000 634 000 634 000 634 000 634 000 634
000 634 000 600
370 737 760 677 370 737 760 677 760 677
370 737 760 677
740 477 740 477 740 477 740 477 740 477
740 477 740 477 740 477 170 717 170 717
170 717 170 717 170 717
170 717
740 477 740 477 740 477 740 477 740 477
740 477 740 477 740 477 002 002 002 002
002 002 002 002 002 002 170 717 170 717
170 717 170 717 170 717
170 717
730 377 001 001 670 767 730 377 001 001
001 001 670 767
670 767
 EOD
```

```
      INTEGER CR(10,16,16),TEMP(20),INST(20,200),MRC(16,16,16)
      INTEGER SCOUNT,PCOUNT,TR(16,16)
      REAL MR1(16,16,16),MR2(16,16,16)
      REAL CPA(16,16),CPQ(16,16)
      COMMON SCOUNT,PCOUNT
      SCOUNT=0
      PCOUNT=0
      DO 4 I=1,16
      DO 4 J=1,16
      DO 4 K=1,16
      MRC(I,J,K)=0
      MR1(I,J,K)=0.
      MR2(I,J,K)=0.
      CPA(J,K)=0.
      CPQ(J,K)=0.
    4 CONTINUE
      DO 5 I=1,20
      DO 5 J=1,200
      INST(I,J)=0
    5 CONTINUE
      DO 6 I=1,10
      DO 6 J=1,16
      DO 6 K=1,16
    6 CR(I,J,K)=0
      MINDX=   83
      DO 13 J=1,MINDX
      READ(110,14)(INST(I,J),I=1,20)
   14 FORMAT(20A4)
   13 WRITE(108,15)(INST(I,J),I=1,20)
   15 FORMAT(1X,20A4)
C    SIM3D-FORTRAN MIX (SIM3DMIX)
C    GAUSSIAN ELIMINATION ON A 3 BY 3
C    FIRST TRY
      I=1
      CALL SINPT( 1, 3, 1, 7,INST(1,   5),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 3, 1, 7,INST(1,   6),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 1, 1, 7,INST(1,   7),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 3, 1, 7,INST(1,   8),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 2, 3, 1, 7,INST(1,   9),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 3, 1, 2,INST(1,  10),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 2, 1, 7,INST(1,  11),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 3, 1, 7,INST(1,  12),MR1,MR2,MRC,CR,TR)
      CALL SINPT( 1, 3, 1, 3,INST(1,  13),MR1,MR2,MRC,CR,TR)
  415 CALL STRAN( 1, 1, 1, 7,INST(1,  14),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 1, 1, 1, 7,INST(1,  15),MR1,MR2,MRC)
      CALL STRAN( 1, 3, 1, 7,INST(1,  16),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 1, 3, 1, 7,INST(1,  17),MR1,MR2,MRC)
      CALL STRAN( 1, 1, 1, 7,INST(1,  18),MR1,MR2,MRC,CPA,CPQ,CR)
```

```
      CALL STRAN( 1, 1, 1, 7,INST(1,  19),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SOUTP( 1, 3, 1, 7,INST(1,  20),MR1,MR2)
      CALL SDTRN( 1, 3, 1, 7,INST(1,21),MR1,MR2,MRC)
      CALL STRAN( 1, 3, 1, 7,INST(1,  22),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL STRAN( 2, 3, 1, 7,INST(1,  23),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 2, 3, 1, 7,INST(1,  24),MR1,MR2,MRC)
      CALL STRAN( 1, 3, 1, 7,INST(1,  25),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 1, 3, 1, 7,INST(1,  26),MR1,MR2,MRC)
      CALL STRAN( 2, 3, 1, 7,INST(1,  27),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL STRAN( 2, 3, 1, 7,INST(1,  28),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL STRAN( 2, 3, 1, 7,INST(1,  29),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL STRAN( 2, 3, 1, 7,INST(1,  30),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SOUTP( 1, 3, 1, 7,INST(1,  31),MR1,MR2)
      I=I+1
      IF(I.EQ.3)GO TO 115
      IF(I.EQ.4)GO TO 215
      CALL STRAN( 1, 3, 1, 2,INST(1,  35),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 1, 3, 1, 2,INST(1,  36),MR1,MR2,MRC)
      CALL STRAN( 1, 2, 1, 7,INST(1,  37),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 1, 2, 1, 7,INST(1,  38),MR1,MR2,MRC)
      CALL SBUN(INST(1,  39), 415S)
  115 CALL STRAN( 1, 3, 1, 7,INST(1,  40),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 1, 3, 1, 7,INST(1,  41),MR1,MR2,MRC)
      CALL STRAN( 1, 3, 1, 3,INST(1,  42),MR1,MR2,MRC,CPA,CPQ,CR)
      CALL SCTRN( 1, 3, 1, 3,INST(1,  43),MR1,MR2,MRC)
      CALL SBUN(INST(1,  44), 415S)
  215 CALL SOUTP( 1, 3, 1, 7,INST(1,  45),MR1,MR2)
 9999 WRITE(108,999)
  999 FORMAT(!0STOP INSTRUCTION ENCOUNTERED!)
      WRITE(108,998)
  998 FORMAT(1H1)
      CALL EXIT
      END
```

```
(01,03,01,07)   INPT(00,15)     MR (01)
                INPT            CR (08)
(01,01,01,07)   INPT            CR (01)
(01,03,01,07)   INPT            CR (02)
(02,03,01,07)   INPT            CR (03)
(01,03,01,02)   INPT            CR (04)
(01,02,01,07)   INPT            CR (05)
(01,03,01,07)   INPT            CR (06)
(01,03,01,03)   INPT            CR (07)
(01,01,01,07)   TRAN            CR (01)         MC (01)                 415
                CTRN            MR1(01)         MR2(01)
(01,03,01,07)   TRAN            CR (02)         MC (01)
                CTRN            MR2(01)         MR2(01)
(01,01,01,07)   TRAN(00,00,16)  MR (01)         CP              FDV
                TRAN(00,15)     CP              MR (01)
(01,03,01,07)   OUTP            MR1(01)
                CTRN            MR1(01)         MR2(01)
                TRAN(00,15)     MR (01)         MR (02)
(02,03,01,07)   TRAN            CR (03)         MC (02)
                CTRN            MR1(02)         MR2(02)
(01,03,01,07)   TRAN            CR (08)         MC (02)
                CTRN            MR1(02)         MR1(02)
(02,03,01,07)   TRAN(00,00,16)  MR (02)         CP              FMY
                TRAN(15,00,16)  MR (01)         CP              FSB
                TRAN(15,15,16)  MR (01)         CP              CPM
                TRAN(00,15)     CP              MR (01)
(01,03,01,07)   OUTP            MR1(01)
F       I=I+1
F       IF(I.EQ.3)GO TO 115
F       IF(I.EQ.4)GO TO 215
(01,03,01,02)   TRAN            CR (04)         MC (01)
                CTRN            MR1(01)         MR1(01)
(01,02,01,07)   TRAN            CR (05)         MC (01)
                CTRN            MR1(01)         MR1(01)
                BRNU            415
(01,03,01,07)   TRAN            CR (06)         MC (01)                 115
                CTRN            MR1(01)         MR1(01)
(01,03,01,03)   TRAN            CR (07)         MC (01)
                CTRN            MR1(01)         MR1(01)
                BRNU            415
(01,03,01,07)   OUTP            MR1(01)                                 215
                STOP
   2.      -7.      4.      9.      1.      0.      0.      1.
   9.      -6.      1.      0.      1.      0.      -3.     8.
   5.      6.       0.      0.      1.
040 000         040 000         040 000         040 000
040 000         040 000         040 000         144 000
144 000         144 000         144 000         144 000
```

```
 144 000          144 000          100 000          100 000
 100 000          100 000          100 000          100 000
 100 000
 070 734          000 634          000 634          000 634
 000 634          000 634          000 600
 000 040          000 040          000 040          000 040
 000 040          000 040          000 040          000 144
 000 144          000 144          000 144          000 144
 000 144          000 144          000 100          000 100
 000 100          000 100          000 100          000 100
 000 100
 070 734          000 634          000 634          000 634
 000 634          000 634          000 600          070 734
 000 634          000 634          000 634          000 634
 000 634          000 600
 370 737          760 677          370 737          760 677
 370 737          760 677
 740 477          740 477          740 477          740 477
 740 477          740 477          740 477          170 717
 170 717          170 717          170 717          170 717
 170 717          170 717
 740 477          740 477          740 477          740 477
 740 477          740 477          740 477          002 002
 002 002          002 002          002 002          002 002
 002 002          002 002          170 717          170 717
 170 717          170 717          170 717          170 717
 170 717
 730 377          001 001          670 767          730 377
 001 001          670 767          730 377          001 001
 670 767
  EOD
  (01 03 01 07)   INPT(00 15)    MR (01)
 DEBUGGING CHECK ON ARRAY INDEXES
IR= 1    JR= 3    IC= 1    JC= 7    II= 1
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          21
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           1
 DEBUGGING CHECK ON RTEMP DURING SINPT
         2.000000        -7.000000        4.000000        9.000000        1.000000
          .000000         .000000
         1.000000         9.000000        -6.000000        1.000000         .000000
        1.000000         .000000
        -3.000000         8.000000        5.000000        6.000000         .000000
```

```
    .0000000       1.000000
 DEBUGGING CHECK OF MR1( 1) REGISTERS AFTER SINPT
      2.000000      -7.000000       4.000000       9.000000       1.000000
      .000000        .000000
       1.000000      9.000000      -6.000000       1.000000        .000000
  1.000000            .000000
      -3.000000      8.000000       5.000000       6.000000        .000000
    .0000000       1.000000
                   INPT                CR (08)
 DEBUGGING CHECK ON ARRAY INDEXES
IR= 1    JR= 3    IC= 1    JC= 7    II= 8
 REGISTER CONNECTION
  A    Q    OP        A    Q    OP        A    Q    OP        A    Q    OP
 40    0              40    0              40    0              40    0
 40    0              40    0              40    0             144    0
144    0             144    0             144    0             144    0
144    0             144    0             100    0             100    0
100    0             100    0             100    0             100    0
100    0
 DEBUGGING CHECK OF MASK MATRIX
   1   1   1   1   1   1   1
   1   1   1   1   1   1   1
   1   1   1   1   1   1   1
 DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
   00100000    00101010    00102020    00103030    00104040    00105050    0010
6060
   00900100    00901110    00902120    00903130    00904140    00905150    0090
6160
   00900200    00901210    00902220    00903230    00904240    00905250    0090
6260
  (01 01 01 07)   INPT                CR (01)
 DEBUGGING CHECK ON ARRAY INDEXES
IR= 1    JR= 1    IC= 1    JC= 7    II= 1
 REGISTER CONNECTION
  A    Q    OP        A    Q    OP        A    Q    OP        A    Q    OP
 70  734              0  634              0  634              0  634
  0  634              0  634              0  600
 DEBUGGING CHECK OF MASK MATRIX
   1   1   1   1   1   1   1
 DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
   00400000    00401000    00402000    00403000    00404000    00405000    0040
6000
  (01 03 01 07)   INPT                CR (02)
 DEBUGGING CHECK ON ARRAY INDEXES
IR= 1    JR= 3    IC= 1    JC= 7    II= 2
 REGISTER CONNECTION
  A    Q    OP        A    Q    OP        A    Q    OP        A    Q    OP
  0   40              0   40              0   40              0   40
```

```
 0  40                0  40                0  40                0  144
 0  144               0  144               0  144               0  144
 0  144               0  144               0  100               0  100
 0  100               0  100               0  100               0  100
 0  100
DEBUGGING CHECK OF MASK MATRIX
   1   1   1   1   1   1   1
   1   1   1   1   1   1   1
   1   1   1   1   1   1   1
DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
   00100000    00101010    00102020    00103030    00104040    00105050    0010
6060
   00510000    00511010    00512020    00513030    00514040    00515050    0051
6060
   00520000    00521010    00522020    00523030    00524040    00525050    0052
6060
 (02,03,01,07)   INPT              CR (03)
DEBUGGING CHECK ON ARRAY INDEXES
IR= 2    JR= 3    IC= 1    JC= 7    II= 3
REGISTER CONNECTION
  A    Q    OP        A    Q    OP        A    Q    OP        A    Q    OP
 70  734               0  634               0  634               0  634
  0  634               0  634               0  600              70  734
  0  634               0  634               0  634               0  634
  0  634               0  600
DEBUGGING CHECK OF MASK MATRIX
   1   1   1   1   1   1   1
   1   1   1   1   1   1   1
DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
   00410100    00411100    00412100    00413100    00414100    00415100    0041
6100
   00420200    00421200    00422200    00423200    00424200    00425200    0042
6200
 (01,03,01,02)   INPT              CR (04)
DEBUGGING CHECK ON ARRAY INDEXES
IR= 1    JR= 3    IC= 1    JC= 2    II= 4
REGISTER CONNECTION
  A    Q    OP        A    Q    OP        A    Q    OP        A    Q    OP
370  737               760  677              370  737               760  677
370  737               760  677
DEBUGGING CHECK OF MASK MATRIX
   1   1
   1   1
   1   1
DEBUGGING CHECK ON TR REGISTER GENERATED BY PACK
   00901000    00900010
   00911100    00910110
   00921200    00920210
```

```
 (01 02 01 07)   INPT                CR (05)
DEBUGGING CHECK ON ARRAY INDEXES
IR= 1     JR= 2     IC= 1     JC= 7     II= 5
REGISTER CONNECTION
  A     Q     OP          A     Q     OP          A     Q     OP          A     Q     OP
740   477                 740   477               740   477               740   477
740   477                 740   477               740   477               170   717
170   717                 170   717               170   717               170   717
170   717                 170   717
DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
DEBUGGING CHECK ON TR REGISTER GENERATED BY RACK
    00910000    00911010    00912020    00913030    00914040    00915050    0091
6060
    00900100    00901110    00902120    00903130    00904140    00905150    0090
6160
 (01 03 01 07)   INPT                CR (06)
DEBUGGING CHECK ON ARRAY INDEXES
IR= 1     JR= 3     IC= 1     JC= 7     II= 6
REGISTER CONNECTION
  A     Q     OP          A     Q     OP          A     Q     OP          A     Q     OP
740   477                 740   477               740   477               740   477
740   477                 740   477               740   477                 2     2
  2     2                   2     2                 2     2                 2     2
  2     2                   2     2               170   717               170   717
170   717                 170   717               170   717               170   717
170   717
DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
DEBUGGING CHECK ON TR REGISTER GENERATED BY RACK
    00920000    00921010    00922020    00923030    00924040    00925050    0092
6060
    00110100    00111110    00112120    00113130    00114140    00115150    0011
6160
    00900200    00901210    00902220    00903230    00904240    00905250    0090
6260
 (01 03 01 03)   INPT                CR (07)
DEBUGGING CHECK ON ARRAY INDEXES
IR= 1     JR= 3     IC= 1     JC= 3     II= 7
REGISTER CONNECTION
  A     Q     OP          A     Q     OP          A     Q     OP          A     Q     OP
730   377                   1     1               670   767               730   377
  1     1               670   767               730   377                 1     1
670   767
DEBUGGING CHECK OF MASK MATRIX
```

```
     1    1    1
     1    1    1
     1    1    1
 DEBUGGING CHECK ON TR REGISTER GENERATED BY RACK
    00902000     00101010     00900020
    00912100     00111110     00910120
    00922200     00121210     00920220
  (01,01,01,07)    TRAN              CR (01)          MC (01)                415
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
           28
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           2
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00400000     00401000     00402000     00403000     00404000     00405000     0040
6000
                    CTRN                MR1(01)          MR2(01)
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1    1    1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
         1         1         1         1         1         1         1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
         1         1         1         1         1         1         1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
           35
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           3
  (01,03,01,07)    TRAN              CR (02)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
           56
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
           4
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00100000     00101010     00102020     00103030     00104040     00105050     0010
6060
    00510000     00511010     00512020     00513030     00514040     00515050     0051
6060
    00520000     00521010     00522020     00523030     00524040     00525050     0052
6060
                    CTRN                MR2(01)          MR2(01)
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1    1    1
    1    1    1    1    1    1    1
    1    1    1    1    1    1    1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
         1         1         1         1         1         1         1
         1         1         1         1         1         1         1
         1         1         1         1         1         1         1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
```

```
          1         2         3         4         5         6         7
          1         2         3         4         5         6         7
          1         2         3         4         5         6         7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         77
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          5
  (01 01 01 07)   TRAN(00 00 16)  MR (01)          CP                PDV
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         84
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          6
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         91
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          7
 DEBUGGING CHECK OF MASK MATRIX
    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
         98
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          8
                   TRAN(00 15)      CP                MR (01)
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        105
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          9
  (01 03 01 07)   OUTP              MR1(01)
 CONTENTS OF MR1 REGISTERS OF PLANE 1
   .100000E 01   .350000E 01   .200000E 01   .450000E 01   .500000E 00   .
0000000E 00   .000000E 00
   .100000E 01   .900000E 01   .600000E 01   .100000E 01   .000000E 00   .
1000000E 01   .000000E 00
   .300000E 01   .800000E 01   .500000E 01   .600000E 01   .000000E 00   .
0000000E 00   .100000E 01
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        126
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         10
                   DTRN            MR1(01)          MR2(01)
 DEBUGGING CHECK OF MASK MATRIX
    1    1    1    1    1    1    1
```

```
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          147
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          11
                      TRAN(00,15)      MR (01)          MR (02)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          168
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          12
   (02,03,01,07)   TRAN             CR (03)          MC (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          182
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          13
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
     00410100     00411100     00412100     00413100     00414100     00415100     0041
6100
     00420200     00421200     00422200     00423200     00424200     00425200     0042
6200
                      CTRN             MR1(02)          MR2(02)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
          2         2         2         2         2         2         2
          3         3         3         3         3         3         3
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          1         1         1         1         1         1         1
          1         1         1         1         1         1         1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          196
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          14
   (01,03,01,07)   TRAN             CR (08)          MC (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          217
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          15
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
     00100000     00101010     00102020     00103030     00104040     00105050     0010
6060
     00900100     00901110     00902120     00903130     00904140     00905150     0090
```

```
6160
     00900200     00901210     00902220     00903230     00904240     00905250     0090
6260
                       CTRN                 MR1(02)              MR1(02)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
          1              1              1              1              1              1
          1              1              1              1              1              1
          1              1              1              1              1              1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          1              2              3              4              5              6              7
          1              2              3              4              5              6              7
          1              2              3              4              5              6              7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          238
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          16
  (02,03,01,07)    TRAN(00,00,16)  MR (02)          CP                FMY
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          252
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          17
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          266
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          18
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          280
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          19
                  TRAN(15,00,16)  MR (01)          CP                FSB
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          294
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
```

```
          20
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          308
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          21
                        TRAN(15,15,16)  MR (01)          CP                 CRM
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          322
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          22
                        TRAN(00,15)     CP               MR (01)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
     1    1    1    1    1    1    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          336
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          23
  (01,03,01,07)   OUTP             MR1(01)
 CONTENTS OF MR1 REGISTERS OF PLANE  1
     .1000000E 01   -.3500000E 01    .2000000E 01    .4500000E 01    .5000000E 00   .
.0000000E 00    .0000000E 00
     .0000000E 00    .1250000E 02   -.8000000E 01   -.3500000E 01   -.5000000E 00   .
.1000000E 01    .0000000E 00
     .0000000E 00   -.2500000E 01    .1100000E 02    .1950000E 02    .1500000E 01   .
.0000000E 00    .1000000E 01
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          357
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          24
  (01,03,01,02)   TRAN             CR (04)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          363
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          25
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
     00901000    00900010
     00911100    00910110
     00921200    00920210
                 CTRN             MR1(01)          MR1(01)
 DEBUGGING CHECK OF MASK MATRIX
     1    1
     1    1
     1    1
```

```
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
          1          1
          2          2
          3          3
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          2          1
          2          1
          2          1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        369
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         26
  (01,02,01,07)   TRAN              CR (05)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        383
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         27
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00910000    00911010    00912020    00913030    00914040    00915050    0091
6060
    00900100    00901110    00902120    00903130    00904140    00905150    0090
6160
                  CTRN              MR1(01)          MR1(01)
 DEBUGGING CHECK OF MASK MATRIX
   1   1   1   1   1   1   1
   1   1   1   1   1   1   1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
          2          2          2          2          2
          1          1          1          1          1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          1          2          3          4          5          6          7
          1          2          3          4          5          6          7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        397
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         28
                  BRNU              415
  (01,01,01,07)   TRAN              CR (01)          MC (01)          415
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        404
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         29
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00400000    00401000    00402000    00403000    00404000    00405000    0040
6000
                  CTRN              MR1(01)          MR2(01)
 DEBUGGING CHECK OF MASK MATRIX
   1   1   1   1   1   1   1
```

```
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
        1        1        1        1        1        1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
        1        1        1        1        1        1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       411
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        30
  (01:03,01:07)   TRAN              CR (02)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       432
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        31
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
   00100000    00101010    00102020    00103030    00104040    00105050    0010
6060
   00510000    00511010    00512020    00513030    00514040    00515050    0051
6060
   00520000    00521010    00522020    00523030    00524040    00525050    0052
6060
                 CTRN              MR2(01)          MR2(01)
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
        1        1        1        1        1        1
        1        1        1        1        1        1
        1        1        1        1        1        1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
        1        2        3        4        5        6        7
        1        2        3        4        5        6        7
        1        2        3        4        5        6        7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       453
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        32
  (01:01,01:07)   TRAN(00,00,16)  MR (01)          CP               FDV
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       460
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        33
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       467
```

```
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          34
  DEBUGGING CHECK OF MASK MATRIX
       1
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          474
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          35
                        TRAN(00,15)      CP                MR (01)
  DEBUGGING CHECK OF MASK MATRIX
     1   1   1   1   1   1   1
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          481
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          36
   (01,03,01,07)    OUTP              MR1(01)
 CONTENTS OF MR1 REGISTERS OF PLANE  1
     .100000E 01     .000000E 00    -.640000E 00    -.280000E 00    -.400000E-01     .
.8000002E-01     .000000E 00
    -.350000E 01     .100000E 01     .200000E 01     .450000E 01     .500000E 00     .
.0000000E 00     .000000E 00
    -.250000E 01     .000000E 00     .110000E 02     .195000E 02     .150000E 01     .
.0000000E 00     .100000E 01
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          502
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          37
                        DTRN              MR1(01)           MR2(01)
  DEBUGGING CHECK OF MASK MATRIX
     1   1   1   1   1   1   1
     1   1   1   1   1   1   1
     1   1   1   1   1   1   1
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          523
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          38
                        TRAN(00,15)      MR (01)           MR (02)
  DEBUGGING CHECK OF MASK MATRIX
     1   1   1   1   1   1   1
     1   1   1   1   1   1   1
     1   1   1   1   1   1   1
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          544
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          39
   (02,03,01,07)    TRAN              CR (03)           MC (02)
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          558
```

```
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
      40
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00410100    00411100    00412100    00413100    00414100    00415100    0041
6100
    00420200    00421200    00422200    00423200    00424200    00425200    0042
6200
                  CTRN              MR1(02)          MR2(02)
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
        2       2       2       2       2       2       2
        3       3       3       3       3       3       3
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
        1       1       1       1       1       1       1
        1       1       1       1       1       1       1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      572
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
      41
  (01:03 01:07)   TRAN              CR (08)          MC (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      593
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
      42
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00100000    00101010    00102020    00103030    00104040    00105050    0010
6060
    00900100    00901110    00902120    00903130    00904140    00905150    0090
6160
    00900200    00901210    00902220    00903230    00904240    00905250    0090
6260
                  CTRN              MR1(02)          MR1(02)
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
        1       1       1       1       1       1       1
        1       1       1       1       1       1       1
        1       1       1       1       1       1       1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
        1       2       3       4       5       6       7
        1       2       3       4       5       6       7
        1       2       3       4       5       6       7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      614
```

```
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        43
  (02,03,01,07)    TRAN(00,00,16)  MR (02)          CP              FMY
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        628
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        44
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        642
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        45
 DEBUGGING CHECK OF MASK MATRIX
    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        656
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        46
                  TRAN(15,00,16)  MR (01)          CP              FSB
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        670
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        47
 DEBUGGING CHECK OF MASK MATRIX
    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        684
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        48
                  TRAN(15,15,16)  MR (01)          CP              GPM
 DEBUGGING CHECK OF MASK MATRIX
    1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        698
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        49
                  TRAN(00,15)     CP              MR (01)
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
```

```
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        712
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         50
   (01 03 01 07)   OUTP                MR1(01)
 CONTENTS OF MR1 REGISTERS OF PLANE 1
    .100000E 01    .000000E 00   -.640000E 00   -.280000E 00   -.400000E-01    .
.800000E-01    .000000E 00
    .000000E 00    .100000E 01   -.240000E 00    .352000E 01    .360000E 00    .
.280000E 00    .000000E 00
    .000000E 00    .000000E 00    .940000E 01    .188000E 02    .140000E 01    .
.200000E 00    .100000E 01
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        733
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         51
   (01 03 01 07)   TRAN                CR (06)          MC (01)                  115
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        754
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         52
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00920000    00921010    00922020    00923030    00924040    00925050    0092
.6060
    00110100    00111110    00112120    00113130    00114140    00115150    0011
.6160
    00900200    00901210    00902220    00903230    00904240    00905250    0090
.6260
                  CTRN              MR1(01)          MR1(01)
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
        3       3       3       3       3       3       3
        2       2       2       2       2       2       2
        1       1       1       1       1       1       1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
        1       2       3       4       5       6       7
        1       2       3       4       5       6       7
        1       2       3       4       5       6       7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        775
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         53
   (01 03 01 03)   TRAN                CR (07)          MC (01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        784
```

```
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        54
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00902000     00101010     00900020
    00912100     00111110     00910120
    00922200     00121210     00920220
                   CTRN              MR1(01)          MR1(01)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1
     1    1    1
     1    1    1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
          1         1         1
          2         2         2
          3         3         3
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          3         2         1
          3         2         1
          3         2         1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        793
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        55
                   BRNU              415
  (01,01,01,07)    TRAN              CR(01)           MC(01)           415
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        800
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        56
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00400000     00401000     00402000     00403000     00404000     00405000     0040
6000
                   CTRN              MR1(01)          MR2(01)
 DEBUGGING CHECK OF MASK MATRIX
     1    1    1    1    1    1    1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
          1         1         1         1         1         1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          1         1         1         1         1         1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        807
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        57
  (01,03,01,07)    TRAN              CR(02)           MC(01)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        828
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        58
```

```
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00100000    00101010    00102020    00103030    00104040    00105050    0010
6060
    00510000    00511010    00512020    00513030    00514040    00515050    0051
6060
    00520000    00521010    00522020    00523030    00524040    00525050    0052
6060
                    CTRN            MR2(01)         MR2(01)
 DEBUGGING CHECK OF MASK MATRIX
     1   1   1   1   1   1   1
     1   1   1   1   1   1   1
     1   1   1   1   1   1   1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
         1       1       1       1       1       1
         1       1       1       1       1       1
         1       1       1       1       1       1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
         1       2       3       4       5       6       7
         1       2       3       4       5       6       7
         1       2       3       4       5       6       7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       849
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        59
  (01,01,01,07)   TRAN(00,00,16)  MR(01)          CP              FDV
 DEBUGGING CHECK OF MASK MATRIX
     1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       856
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        60
 DEBUGGING CHECK OF MASK MATRIX
     1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       863
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        61
 DEBUGGING CHECK OF MASK MATRIX
     1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       870
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
        62
                    TRAN(00,15)     CP              MR(01)
 DEBUGGING CHECK OF MASK MATRIX
     1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       877
```

```
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          63
   (01:03:01:07)    OUTP                MR1(01)
  CONTENTS OF MR1 REGISTERS OF PLANE 1
     .100000E 01     .000000E 00     .000000E 00     .200000E 01     .148936E 00     .
212766E-01     .106383E 00
    -.240000E 00     .100000E 01     .000000E 00     .352000E 01     .360000E 00     .
280000E 00     .000000E 00
    -.640000E 00     .000000E 00     .100000E 01    -.280000E 00    -.400000E-01     .
800000E-01     .000000E 00
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          898
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          64
                     DTRN                MR1(01)          MR2(01)
  DEBUGGING CHECK OF MASK MATRIX
      1    1    1    1    1    1    1
      1    1    1    1    1    1    1
      1    1    1    1    1    1    1
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          919
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          65
                     TRAN(00,15)         MR (01)          MR (02)
  DEBUGGING CHECK OF MASK MATRIX
      1    1    1    1    1    1    1
      1    1    1    1    1    1    1
      1    1    1    1    1    1    1
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          940
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          66
   (02:03:01:07)    TRAN                CR (03)          MC (02)
  NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
          954
 NO OF EXECUTION CYCLES OF THE 3-D MACHINE
          67
  DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
     00410100        00411100        00412100        00413100        00414100        00415100        0041
6100
     00420200        00421200        00422200        00423200        00424200        00425200        0042
6200
                     CTRN                MR1(02)          MR2(02)
  DEBUGGING CHECK OF MASK MATRIX
      1    1    1    1    1    1    1
      1    1    1    1    1    1    1
  DEBUGGING CHECK ON CON(I,J) REGISTERS
          2         2         2         2         2         2         2
```

```
          3          3          3          3          3          3          3
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          1          1          1          1          1          1          1
          1          1          1          1          1          1          1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        968
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         68
  (01,03,01,07)    TRAN              CR (08)          MC (02)
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
        989
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         69
 DEBUGGING CHECK ON HEX CONTENTS OF MRC REGISTER IN SUB STRAN
    00100000    00101010    00102020    00103030    00104040    00105050    0010
6060
    00900100    00901110    00902120    00903130    00904140    00905150    0090
6160
    00900200    00901210    00902220    00903230    00904240    00905250    0090
6260
                   CTRN              MR1(02)          MR1(02)
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 DEBUGGING CHECK ON CON(1,I,J) REGISTERS
          1          1          1          1          1          1          1
          1          1          1          1          1          1          1
          1          1          1          1          1          1          1
 DEBUGGING CHECK ON CON(2,I,J) REGISTERS
          1          2          3          4          5          6          7
          1          2          3          4          5          6          7
          1          2          3          4          5          6          7
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1010
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         70
  (02,03,01,07)    TRAN(00,00,16)   MR (02)          CP               FMY
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1024
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         71
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
```

```
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1038
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         72
 DEBUGGING CHECK OF MASK MATRIX
      1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1052
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         73
                    TRAN(15,00,16)  MR (01)          CP                FSB
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1066
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         74
 DEBUGGING CHECK OF MASK MATRIX
      1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1080
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         75
                    TRAN(15,15,16)  MR (01)          CP                CPM
 DEBUGGING CHECK OF MASK MATRIX
      1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1094
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         76
                    TRAN(00,15)     CP               MR (01)
 DEBUGGING CHECK OF MASK MATRIX
    1   1   1   1   1   1   1
    1   1   1   1   1   1   1
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
       1108
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
         77
  (01,03,01,07)   OUTP              MR1(01)
CONTENTS OF MR1 REGISTERS OF PLANE  1
    .100000E 01     .000000E 00     .000000E 00     .200000E 01     .148936E 00     .
212766E-01     .106383E 00
    .000000E 00     .100000E 01     .000000E 00     .400000E 01     .395745E 00     .
285106E 00     .255319E-01
    .000000E 00     .000000E 00     .100000E 01     .999998E 00     .553191E-01     .
936170E-01     .680851E-01
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
```

```
      1129
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
      78
 (01 03 01 07)   OUTP              MR1(01)                                  215
 CONTENTS OF MR1 REGISTERS OF PLANE  1
    .100000E 01    .000000E 00    .000000E 00    .200000E 01    .148936E 00    .
212766E-01    .106383E 00
    .000000E 00    .100000E 01    .000000E 00    .400000E 01    .395745E 00    .
2851O6E 00    .255319E-01
    .000000E 00    .000000E 00    .100000E 01    .999998E 00    .553191E-01    .
936170E-01    .680851E-01
 NO OF EXECUTION CYCLES OF THE SERIAL MACHINE
      1150
NO OF EXECUTION CYCLES OF THE 3-D MACHINE
      79
 STOP INSTRUCTION ENCOUNTERED
```

APPENDIX G

3-D MACHINE'S COST ESTIMATES

This appendix is included to give a preliminary idea of the cost involved in building a prototype of the 3-D machine. The criterion of cost estimation used herein is undoubtedly a simplistic one but it has been used by hardware designers for a long time. The ultimate hardware realization technique used might require a cost estimate revision. The cost estimation here is based on the number of gates needed in a certain subunit. Even if gates are not used as such, an equivalent to the flip-flop which is assumed (e.g. a gate count equivalent to the flip-flop which makes a single bit of memory is used to estimate the cost of the memory). In the analysis a cost estimate of 1¢/gate is assumed, which is more conservative than most estimates used in up-to-date publications. In addition it has been assumed by intuition that the hardware cost comprises only 10% of the total cost of an operative computer system.

Gate count per cell of the MC

AND gates	1504
S-R Flip-flops	160
NOT gates	160

Gate count of a MC of N_i x N_j x N_k cells

Assume that an S-R glip-flop is equivalent to 4 NAND gates.

Gates equivalent to the MC = $N_i x N_j x N_k x (1504+4x160+160)$

$= (N_i x N_j x N_k) x 2304$

Gate count of a CP of N_i x N_j cells

Assume that each cell of the CP has 11 registers of 32 bits each. Each register's bit is an S-R flip-flop with four extra gates/bit to control the register, i.e. each bit of the register is equivalent to 8 gates. Also assume 512 words of read only memory for each microprogrammed Arithmetic/logic unit (ALU). Each word has 32 bits, with 6 gates/bit.

Gates equivalent to the registers = $(N_i x N_j)$ x 11 x 32 x 8

$= (N_i x N_j)$ x 2816

Gates equivalent to the reao only memories = nx512x32x6

= nx98304

n varies from 1 to $(N_k x N_j)$ depending on the number of ALU's.

Gates equivalent to the CP = $(N_i N_j) x 2816+nx98304$

Gate count of an IOP of N_i x N_j cells

Assume 2 registers per cell, with 32 bits each. IOP registers are of the same type of registers used in the CP. Gate equivalent

to the registers = $N_i x N_j x 2 x 32 x 8$

= $N_i x N_j x 512$

Gates equivalent to the control registers = $mx(N_i+N)x10$

= $10mx(N_i+N_j)$

Assuming that there are m IOP processors: m varies between 1 and $(N_i x N_j)$. Also, assume for each processor a HCSR and a VCSR; these registers have N_i bits for the VCSR and N_j bits for the HCSR with 10 gates/bit for the registers.

Gates equivalent to the IOP - $(N_i x N_j)x512+10mx(N_i+N_j)$

Gate count of a CR of N_i x N_j x N_1 cells

There are 3 registers/cell of CR, each register has 32 bits. Assume 5 gates/bit of the register.

Gates equivalent to the CR = $N_i x N_j x N_1 x 3 x 32 x 5$

= $(N_i x N_j x N_1)x480$

Gate count of the 3-D machine

Gates equivalent to the 3-D machine = $(N_i x N_j x N_k x 2304)$ +

Gate count of the 3-D Machine

Gates equivalent to the 3-D machine = $(N_i x N_j x N_k x 2304)$ +

$(N_i x N_j x 2816 + nx98304)$ + $(N_i N_j x 512 + mx10x(N_i+N_j)$ +

$(N_i x N_j x N_\ell x 480)$

Assume: $N_i=N_j$, and $N_\ell=N_k/2$

Gates equivalent to the 3-D machine = $N_i^2 x N_k x 2304 + N_i^2 x 2816 +$
$n x 98304 + N_i^2 x 512 + 20 x m x N_i + N_i^2 x N_k x 240$
$= N_i^2 x N_k x 2544 + N_i^2 x 3328 + N_i x m x 20 + n x 98304$

Assume: $N_k = \alpha N_i$, $m = \beta N_i^2$, $n = \gamma N_i^2$

Gates equivelant to the 3-D machine = $\alpha x N_i^3 x 2544 + N_i^2 x 3328 +$
$N_i^3 x \beta x 20 + N_i^2 x \gamma x 98304$
$N_i^3 x (\alpha x 2544 + \beta x 20) + N_i^2 x (3328 + \gamma x 98304)$

For $N_i = 16$, $\alpha = 1$, $(\beta = \gamma = 1/64)$

Gates equivalent to the 3-D machine = $N_i^3 x (2544 + 20/64) +$
$N_i^2 x (3328 + 98304/64)$
$= 4096 x (2544 + 20/64) + 256 x (3328 + 1536)$
$= 256 x (40,704 + 5) + 256 x (4864)$
$= 10,421,504 + 1,245,184$
$= 11,666,688$ gates

In this gate count, the N_i^3 term is dominated by the storage requirements of the MC and the CR while the N_i^2 term is dominated by the processing requirements in the CP and the IOP. It is to be noticed that the storage requirement is around 10 times the processing control requirement. This is for the case of a 3-D machine having 4 ALU's and 4 IOP processors.

For: $N_i = 16$, $(\alpha = 1, \beta = \gamma = 1/256)$

Gates equivalent to the 3-D machine =

$n_i^3 x(2544 + 20/256) + N_i^2 x(3328 _ 98304/256)$

$= N_i x(651,264+20) + (851,968 + 98,304)$

$- 16x651,284 + 950,272$

$= 10,420,544 + 950,272$

$= 11,370,816$ gates

The above calculation is for the case of a 3-D machine having a single ALU and a single IOP processor.

For: $N_i = 16$, $(\alpha = \beta = \gamma 1)$

Gates count equivalent to the 3-D machine =

$N_i^3 x(2544 + 20) + N_i^2 x(3328 + 98304)$

$= (4,096x2,564) + (256x101,632)$

$= 10,502,144 + 26,017,792$

$= 36,519,836$ gates

This is the case of a full parallel machine with 256 ALU's and 256 IOP processors.

Dollars Estimate

For: $N_i = 16$, $(\alpha = 1$, $\beta = \gamma = 1/64)$, 1 ¢/gate

Gates cost = 11,666,688x(1/100) = \$116,666.88

Machine cost = 116,666.88 x 10 = \$1,166,688.88

The last figure is a cost estimate of an operative 3-D computer with 4 ALU's and 4 IOP processors.

APPENDIX H

NON-CONDUCTIVE DATA TRANSFER TECHNOLOGIES

There are many potentially useful techniques for non-conductive transfer; however, all familiar schemes suffer from one or more serious shortcomings. The most common of these are: (1) The requirement that an energy source be present at any cell to be read, which not only makes the machine expensive to build and maintain but also creates a tremendous heat dissipation problem. (2) A potential conflict between intraplane (conductive) transfer and interplane (non-conductive) transfer.

Figure H.1 is a schematic of the laser non-conductive data transfer mechanism. It is non-ideal in that edge connections are necessary.

A liquid crystal can change from a transparent object to a light reflector (blocking the laser beam for writing into the cell) through changing the bias voltage. A photo detector feeds back the information concerning the contents of the source cell being read to control the transmission of inhibition of the write signal to the destination cell. To achieve non-destructive reading capability, a specific wave length is used for reading while a different wave length is used for writing (8) e.g. for NaF crystals the write wave length is 505 nm, while the read wave length is 203 nm).

Heterojunction lasers (34) are very promising as a non-conductive means of data transfer except that they are still experimental devices and have never been mass produced. One of the most attractive features

of heterojunction lasers is the fact that they can operate at room temperature. They have their own shortcomings such as the relatively polychromatic nature of their emission compared to semiconductor lasers, and the low efficiency of emission compared to semiconductors. These shortcomings can be very well tolerated without degrading the performance of the 3-D machine since light transmission in the 3-D machine is limited to a short distance (no more than 5 feet).

LED - fiber optic technology has reached a highly developed (15) stage in which bundles of fibers can be manufactured and stacked together to be connected between a source and a destination which might require bending the bundle without much loss of the light transmitted and with a very minimal amount of cross-talk.

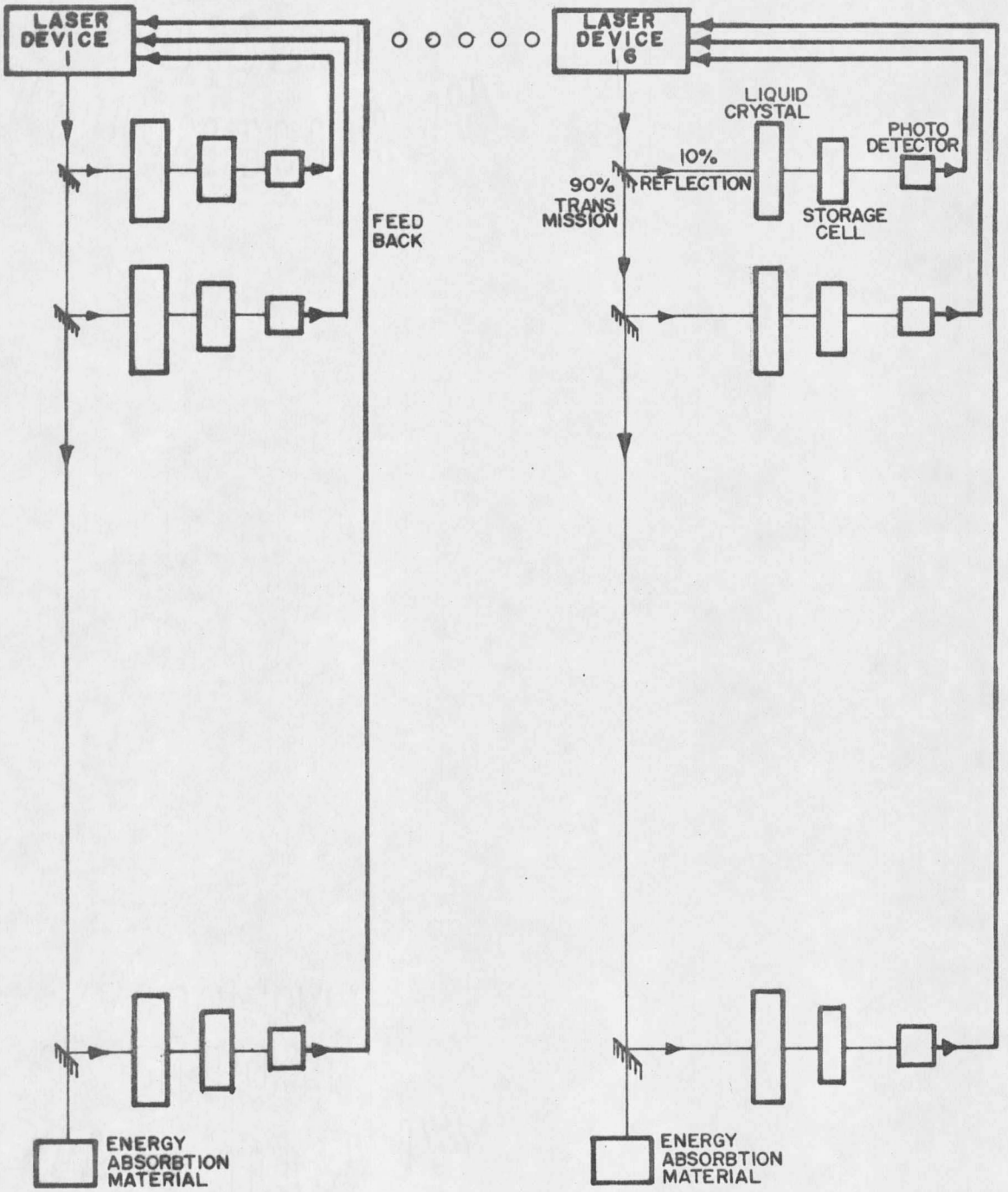


Figure H.1a Non-conductive Transfer System Set Up.

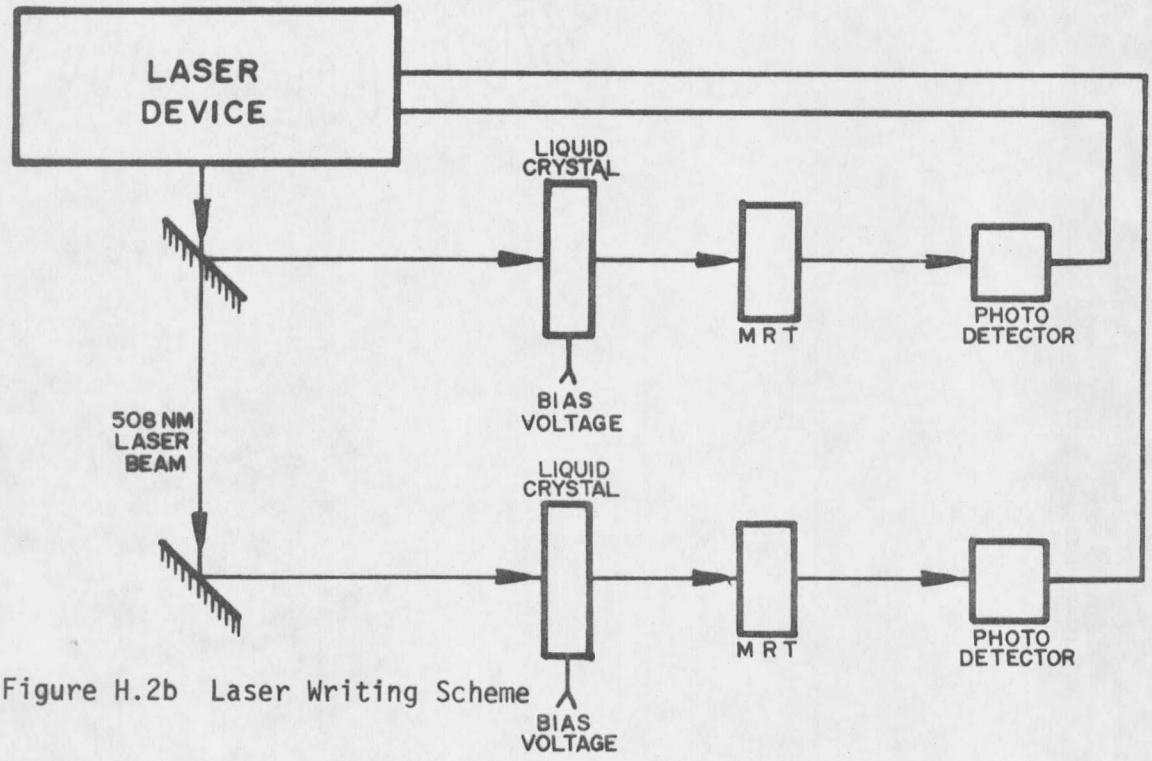


Figure H.2b Laser Writing Scheme

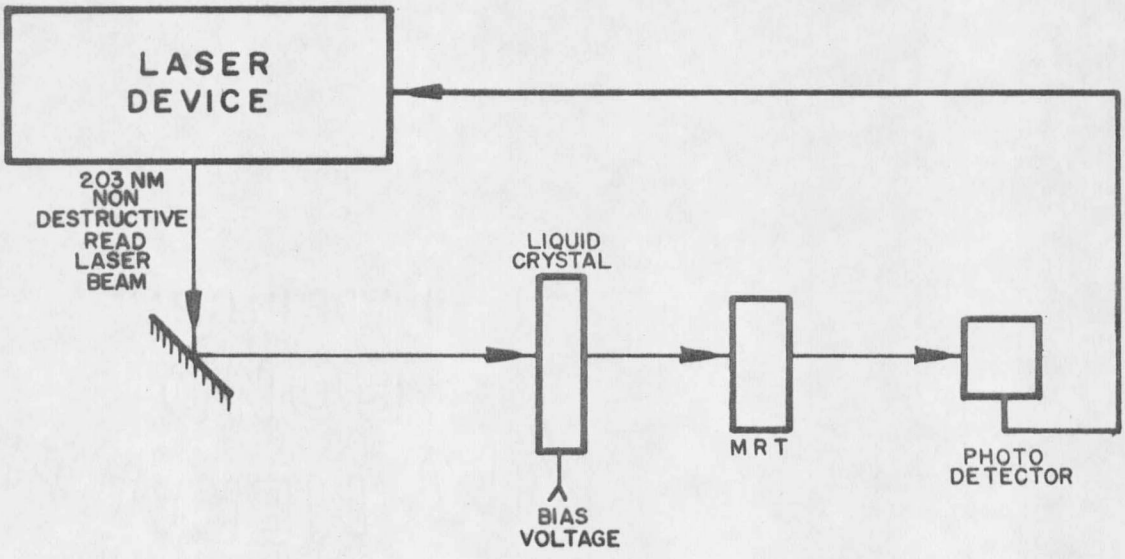


Figure H.2c Laser Reading Scheme

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Avizienix, A. "Design of Fault Tolerant Computers," AFIPS Proceedings, Fall Joint Computer Conference, vol. 31, 1967, pp. 733-744.

2. Fall, J. R., et al. "On the Use of SOLOMON Parallel-Processing Computer," AFIPS Proceedings, Fall Joint Computer Conference, 1962, pp. 137-146.

3. Ball, M., Hardie, F. H. "Self-Repair in a TMR Computer," Computer Design, February 1969, pp. 54-57.

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

5. Bell, C. G., Newell, A. Computer Structures: Reading and Examples, McGraw-Hill Book Company, 1971.

6. Bouknight, W. J., et al. "The ILLIAC IV System," Proceedings of the IEEE, vol. 60, April 1972, pp. 369-388.

7. Boyle, W. S., Smith, G. E. "Charge Couple Semiconductor Devices," The Bell System Technical Journal, April 1970, pp. 587-592.

8. Burt, J. V. "An Experimental, High Density Optical Memory," Report R-557, University of Illinois, Urbana, Illinois, May 1972.

9. Cannon, L. E. "A Cellular Computer to Implement the Lakman Filter Algorithm," Ph.D. Thesis, Montana State University Bozeman, Montana, August 1969.

10. Chandy, K. M., et al. "A Framework for Analyzing Hardware-Software Trade-Offs in Fault Tolerant Computing Systems," AFIPS Proceedings, Fall Joint Computer Conference, vol. 41, Part I, November 1972, pp. 55-65.

11. Chandy, K. M., Ramamoorthy, C. V. "Optimal Rollback," IEEE Transactions on Computers, vol. 21, no. 6, June 1972, pp. 546-556.

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

13. Crane, B. A. et al. "PEPE Computer Architecture," Sixth Annual IEEE Computer Society International, San Francisco, California, September 1972, pp. 57-64.

14. Crane, B. A., Githens, J. A. "Bulk Processing in Distributed Logic Memory," IEEE Transactions on Computers, vol. 14, April 1965, pp. 186-196.

15. Doering, M. D., "Fiber Optics Today," Radio Electronics, August 1972, pp. 33-37.

16. Garnahan, B., et al. Applied Numerical Methods, John Wiley Inc., New York, 1969.

17. Gonzalez, R. A. "A Multi-Layer Iterative Circuit Computer," IEEE Transactions on Computers, vol. 12, December 1963, pp. 781-790.

18. Gregory, J., McReynolds, R. "The SOLOMON Computer," IEEE Transactions on Computers, vol. 16, December 1964, pp. 774-781.

19. Habermann, A. N. "Prevention of System Deadlocks," Comm. ACM, vol. 12, no. 7, July 1969.

20. Hamming, R. W. "Error Detecting and Error Correcting Codes," Bell System Technical J., Vol. 29, pp. 147-160, April 1950.

21. Hellerman, H. Digital Computer System Principles, McGraw-Hill Book Company, New York, 1967.

22. Hobbs, L. C. (Editor) Parallel Processor Systems, Technologies, and Applications, Spartan Books, New York, 1970.

23. Hodges, D. A. "Alternative Component Technologies for Advanded Memory Systems," COMPUTER, IEEE Computer Society, vol. 6, no. 8, August 1973, pp. 34-37.

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

25. Kuck, D. J. "ILLIAC IV Software and Application Programming," IEEE Transactions on Computers, vol. 17, no. 8, August 1968, pp. 758-770.

26. Lee, C. Y. Paul, M. C. "A Content Addressable Distributed Logic Memory with Application Programming," IEEE Transactions on Computers, vol. 17, no. 8, August 1968, pp. 758-770.

27. LeFebre, D. A. "Optical Approaches to Mass Data Storage," Data & Communications Design, May-June 1972, pp. 10-12.

28. Mathur, F. P., Avizienix, A. "Reliability Analysis and Architecture of a Hybrid-Redundant Digital System: Generalized Triple Module Redundancy with Self-Repair,: AFIPS Proceedings, Spring Joint Computer Conference, vol. 36, May 1970.

29. Matick, R. E. "Review of Current Proposed Technologies for Mass Storage Systems," Proceedings of the IEEE, vol. 60, no. 3, March 1972, pp. 266-289.

30. McClusky, E. J. "Introduction to the Theory of Switching Circuits", McGraw-Hill Book Company, New York, 1965.

31. Minnick, R. C. "A Survey of Microcellular Research," Journal of ACM, vol. 14, no. 2, April 1967, pp. 203-241.

32. Mulder, M. C. "The Design of a Matrix-Oriented - Cellular Computer Capable of Fault-Tolerant Operation," Ph.D. Thesis, Montana State University, Bozeman, Montana August 1970.

33. Nuese, C. J., et al. "The Future for LED's," IEEE Spectrum, vol. 9, May 1972, pp. 28-38.

34. Panish, M. B. "Heterostructure Injection Lasers," Bell Labs, Record, November 1971, pp. 298-304.

35. Pease, M. C. "An adaption of the Fast Fourier Transform for Parallel Processing," Journal of ACM, vol. 15, no. 2, April 1968, pp. 252-264.

36. Pierce, W. H. "Failure-Tolerant Computer Design," Academic Press, New York, 1965.

37. Pierre, D. A. "A Nongradient Minimization Algorithm having Parallel Structure, with Implication for An Array Computer," Computer and Electrical Engineering, vol. 1. pp. 3-21, 1973.

38. Rosin, R. F. "Contemporary Concepts of Microprogramming and Emulation," Computer Surveys, pp. 197-212.

39. Ross, M. "Visible and Near-IR Laser Modulation," Electro-Technology, February 1969, pp. 21-28.

40. Rudberg, D. A., "A Microcellular Parallel Processor for the Discrete Kalman Filter," Rocky Mountain Section of the Astronautical Society Conference, Denver, Colorado, July 15-16, 1968.

41. Shore, J. E. "Second Thoughts on Parallel Processing," Naval Research Lab. Report 7346, Washington, D. C., December 1971.

42. Shore, J. E., Collins, R. L. "Another Associative Processor," Naval Research Lab. Report 7348, Washington, D. C., December 1971.

43. Slotnick, D. L. "Unconventional Systems," AFIPS Proceedings, Spring Joint Computer Conference, 1967, pp. 477-481.

44. Slotnick, D. L., et al. "The SOLOMON Computer," AFIPS Proceedings, Fall Joint Computer Conference, 1962, pp. 97-107.

45. Unger, S. H. "A Computer Oriented Towards Spatial Problems" IRE Proceedings, vol. 36, October 1958, pp. 1744-1750.

46. Von Neuman, J. "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," Ann. of Math. Studies Press, Princeton, New Jersey, 1965, pp. 43-49.

47. West, G. P. "The Best Approach to a Large Computing Capability," AFIPS Proceedings, Spring Joint Computer Conference, 1967 pp. 467-469.

48. White, G. S. "Coded Decimal Number System for Digital Computers," Proceedings of IRE, pp. 1450-1457, October 1953.

49. Wilkes, M. V. "Microprogramming," AFIPS Proceedings, The Eastern Joint Computer Conference, 1958, pp. 18-20.

50. Wilkes, M. V. "The Growth of Interest in Microprogramming: A Literature Survey," Computing Surveys, vol. 1, no. 3, September 1969, pp. 139-145.

51. Wilson, D. E. "The PEPE Support Software System," Sixth Annual IEEE Computer Society International Conference, San Francisco, California, September 1972, pp. 61-64.

52. _____. Extended FORTRAN IV Reference Manual for XDS Sigma 5/7 Computers, XDS Publication 90-09-56D, April 1970.

53. _____. Microprogramming Guide for Hewlett-Packard Model 2100 Computer, Hewlett-Packard Company, Cupertino, California, 1971.

54. _____. SYMBOL/MET ASYMBOL Reference Manual for Sigma 5/7 Computers, XDS Publication 90-09-52D, July 1970.

55. _____. "Unpublished Discussion on Advances in LSI Technology," AFIPS, Fall Joint Computer Conference, November 1972.

56. _____. XDS Sigma 7 Computer Reference Manual, XDS Publication 90-09-50H, May 1970.

57. _____. Xerox Universal Time-Sharing (UTS) Sigma 6/7/9 Computers Batch Processing Reference Manual, XDS Publication 90-17-64B, June 1972.

MONTANA STATE UNIVERSITY LIBRARIES
3 1762 10010787 7

D378
H1945 Hanna, Aziz Hanna
cop.2 Register-transfer-level design and software simulation of a parallel processor

NAME AND ADDRESS
[redacted] JAIN SUNIL KUMAR 1123 ROSKIE 2487

[redacted]

D378
H1945
Cop. 2