



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-art 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

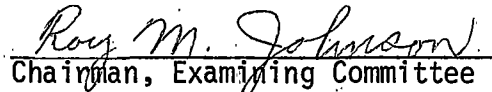
DOCTOR OF PHILOSOPHY

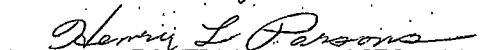
in

Electrical Engineering

Approved:

  
Head, Major Department

  
Chairman, Examining Committee

  
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

	<u>Page</u>
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 accom- plished 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 in- tegrated 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
0	Cell contents are interplane trans- ferred using a rectangular path.....	124
[x]	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)<sup>1</sup> (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. Multipoint 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

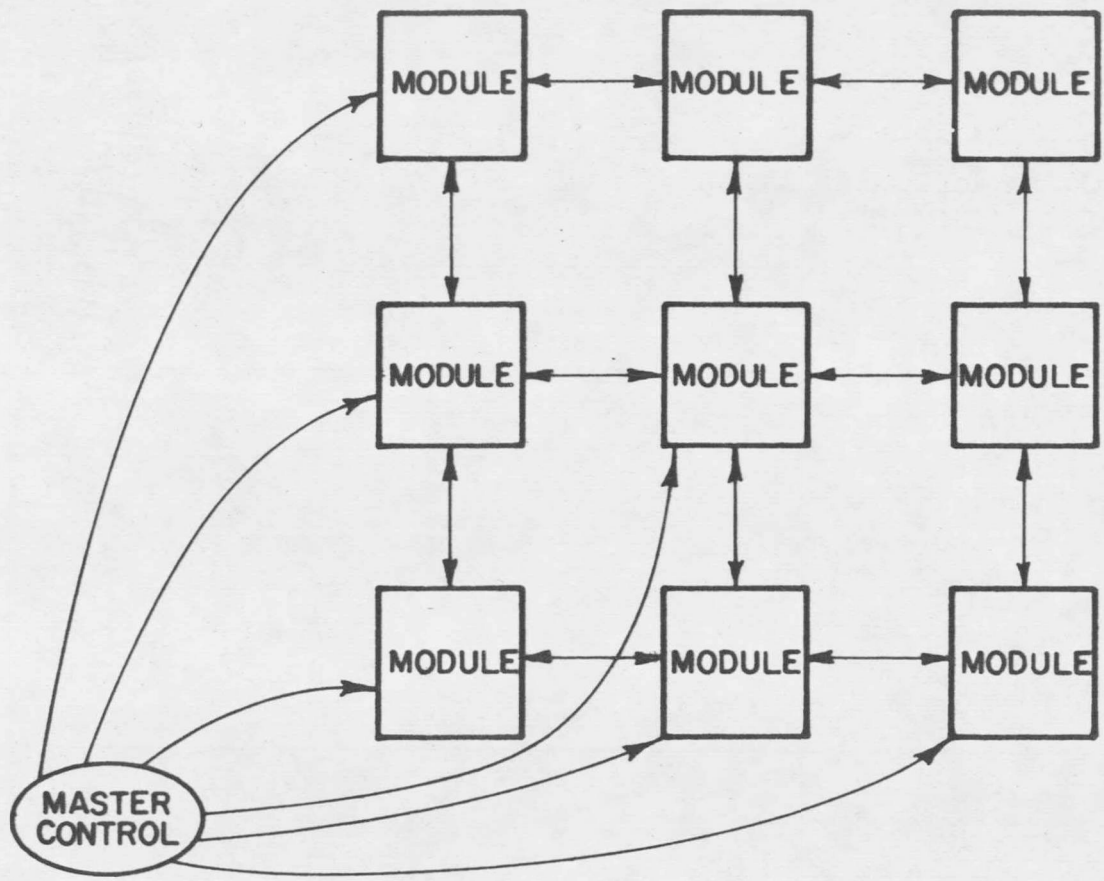


Figure 1.1 The Unger Spatially-Oriented Computer.

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

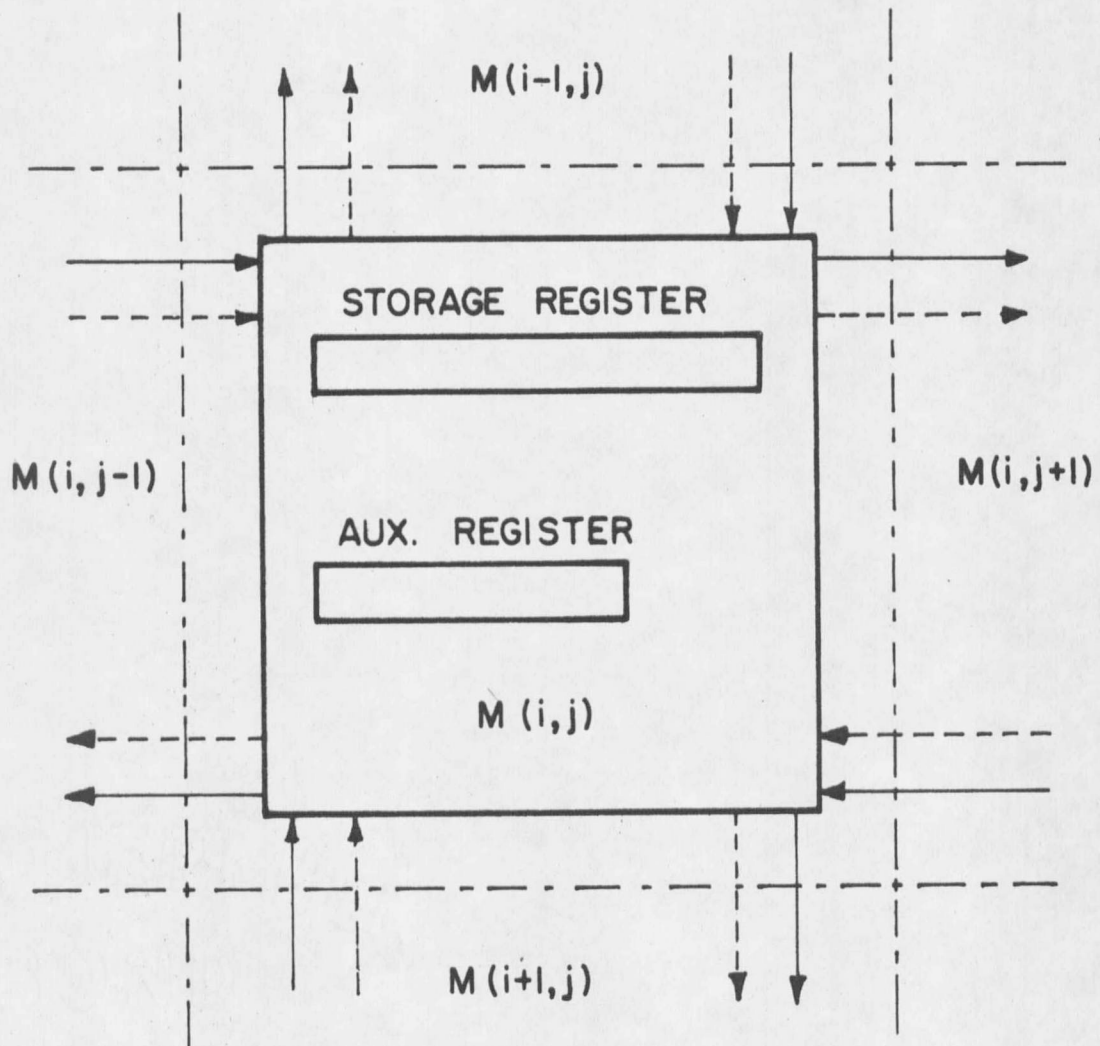


Figure 1.2 A Module of Holland Machine.

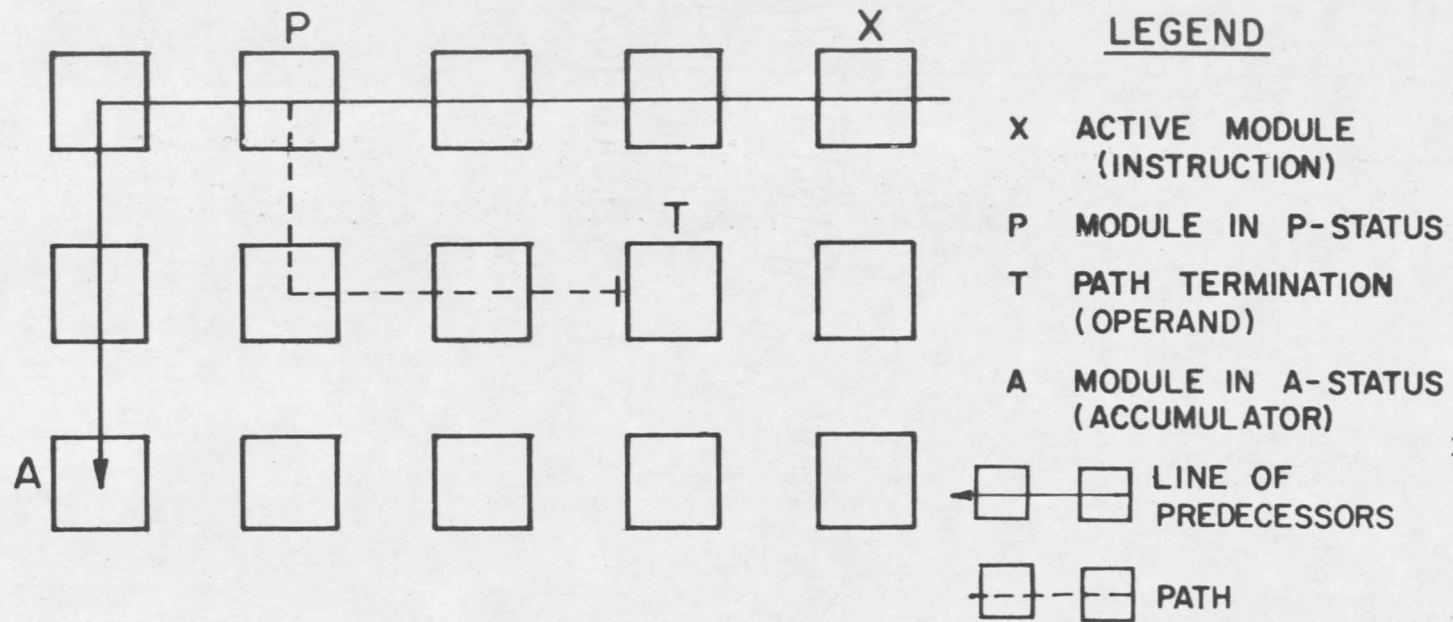


Figure 1.3 Modules Used in the Execution of an Instruction.

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:

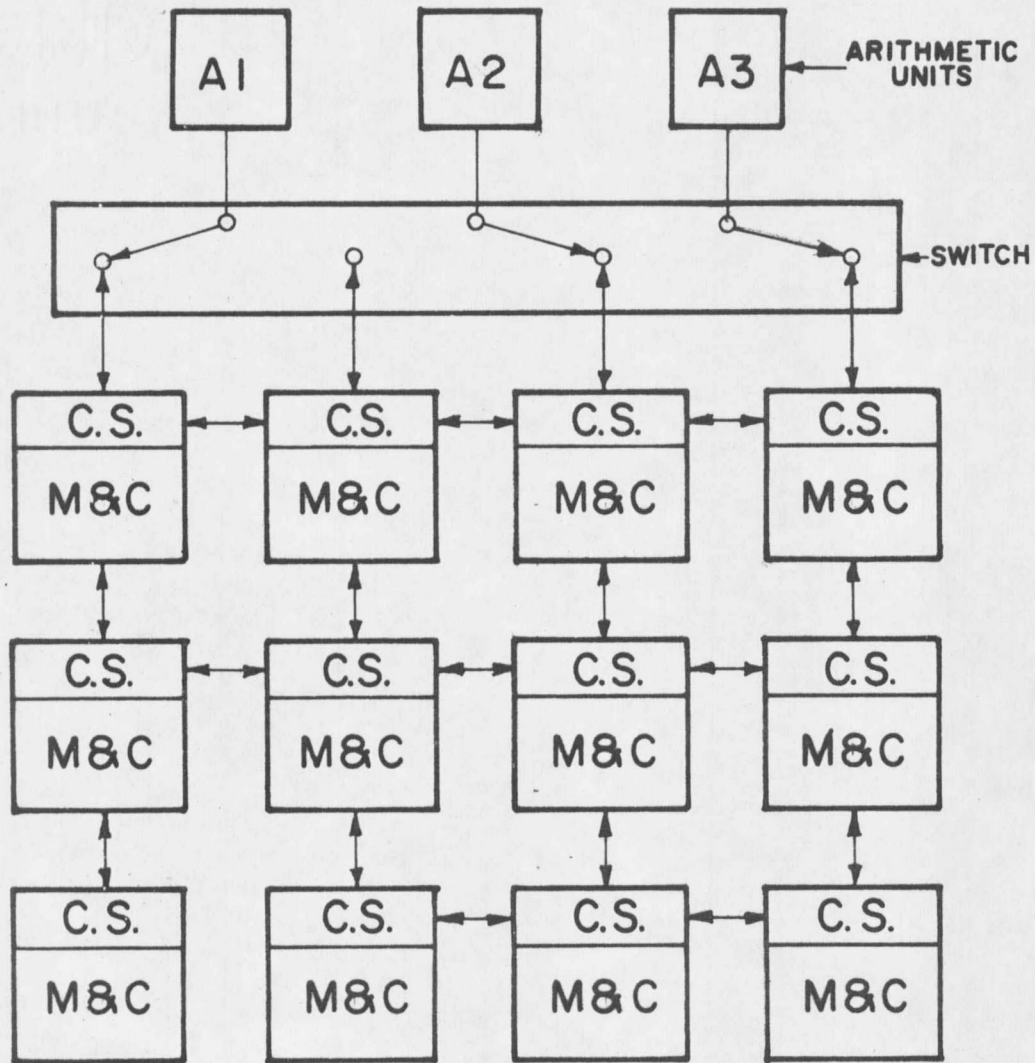
1. A parallel network of computing elements.
2. Sequencing of instructions which are arranged spatially throughout the array of modules with an arbitrary number of sequences being executed at any given time.
3. The method of data accessing.

The Comfort computer is fundamentally 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



LEGEND

C.S. = COMMUNICATION SECTION  
 M&C = MEMORY AND CONTROL

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.

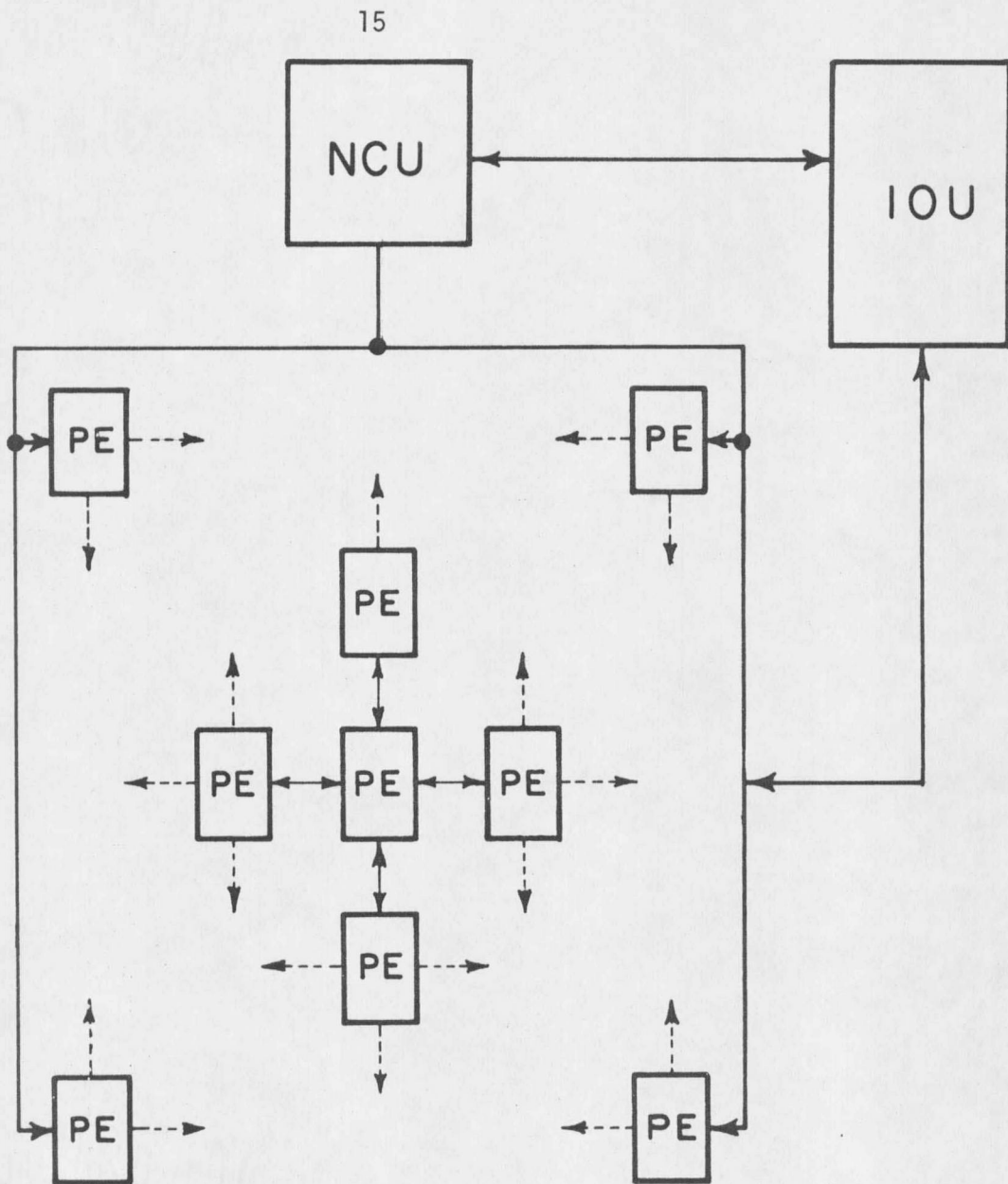


Figure 1.5 The Basic Organization of the SOLOMON Computer.

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 pluggable 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

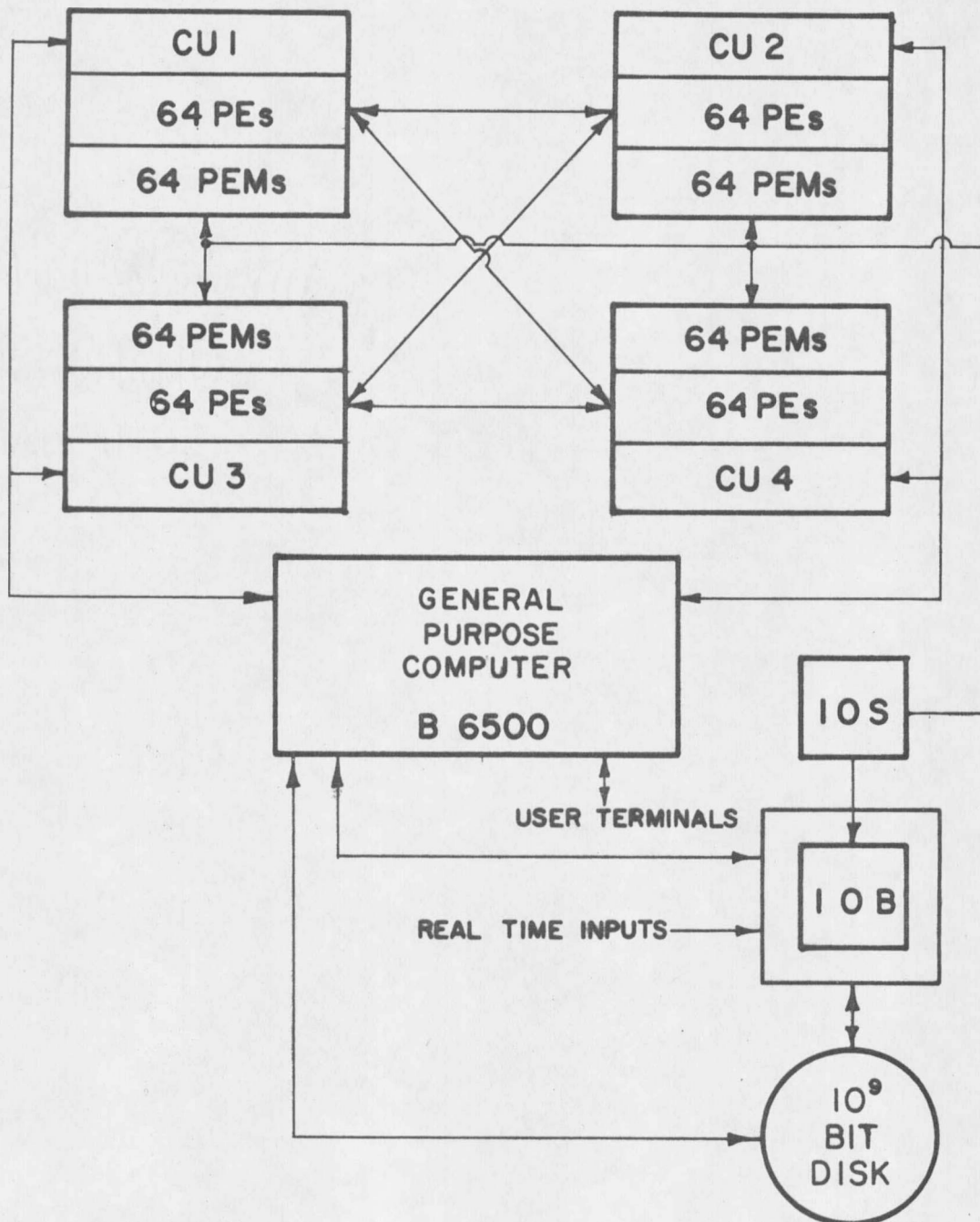


Figure 1.6 The General Organization of the 4 quadrant ILLIAC IV Computer.

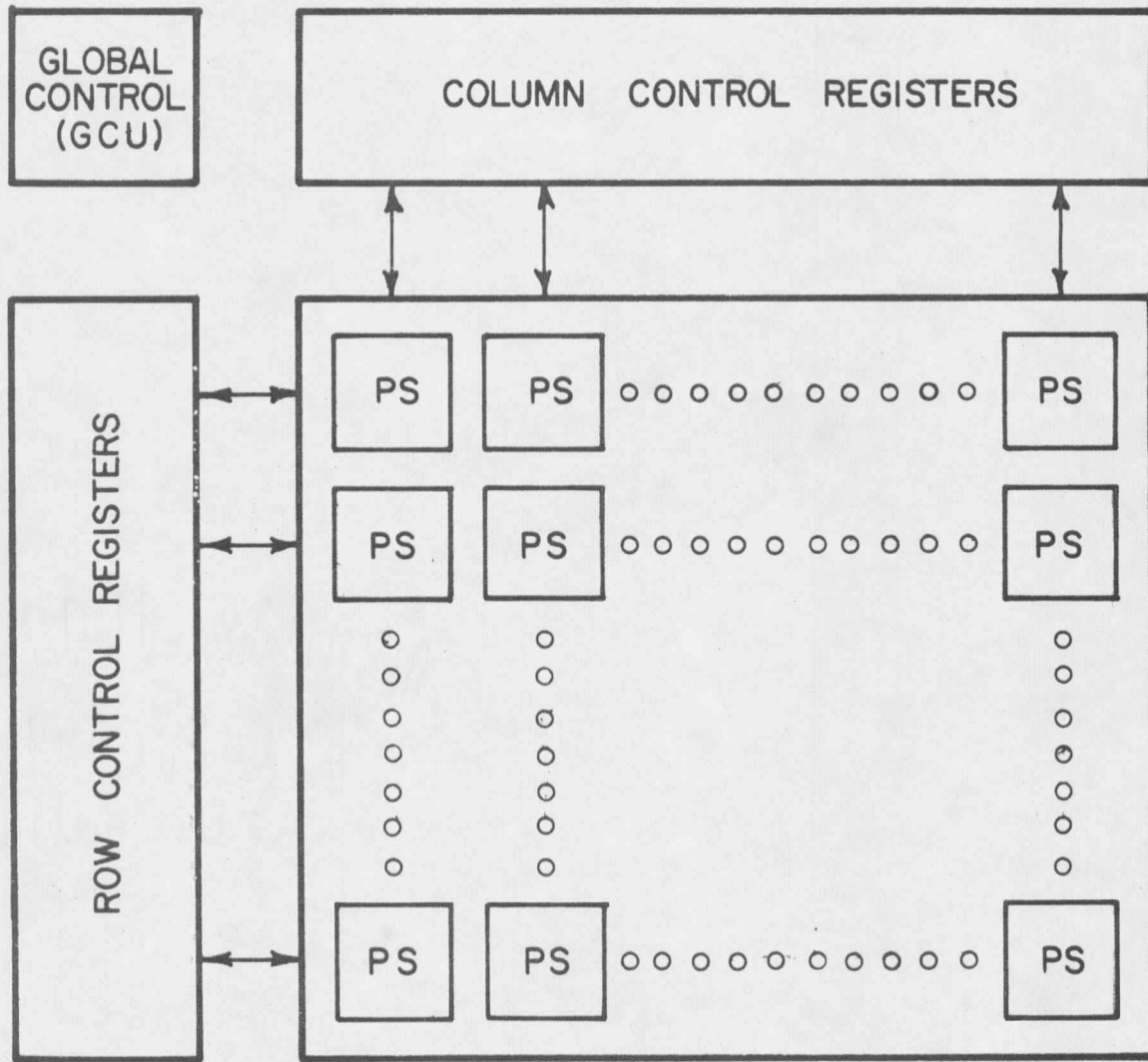
of some modern techniques such as thin magnetic film memories and LSI logic. The goal of the ILLIAC IV project was to build a computer capable of executing on the order of  $10^9$  instructions/sec, but due to subcontractor problems (6), some technological changes were necessary. This led to cost escalation, reduction of execution rate to  $2 \times 10^8$  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  $i, j^{\text{th}}$  element of the array is stored in the  $i, j^{\text{th}}$  cell of the processing array. See Figure 1.7.



LEGEND

GCU = Global Control Unit

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 reconfigurable 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

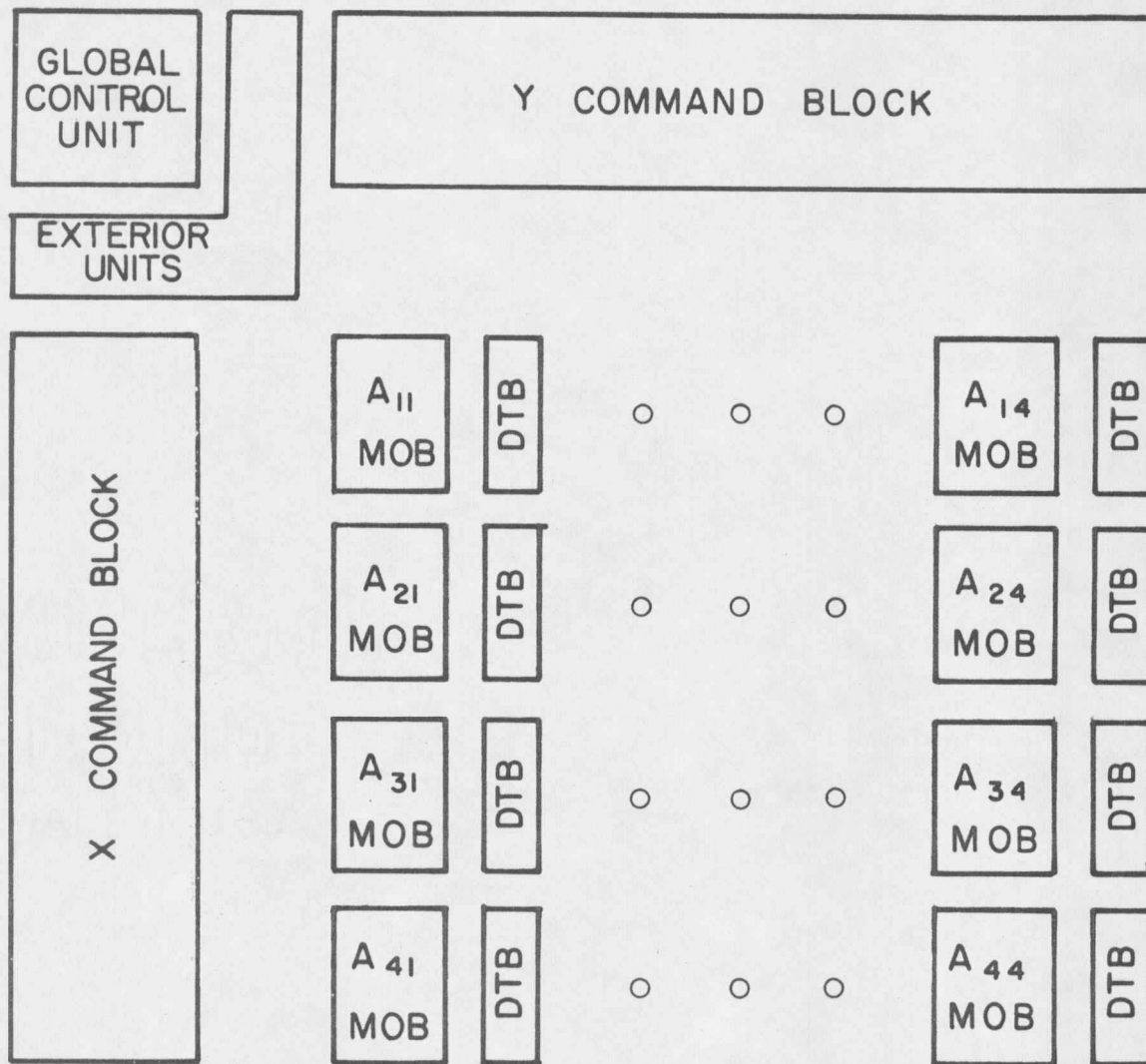


Figure 1.8 General Organization of Mulder's Computer

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  $N \times N$  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.

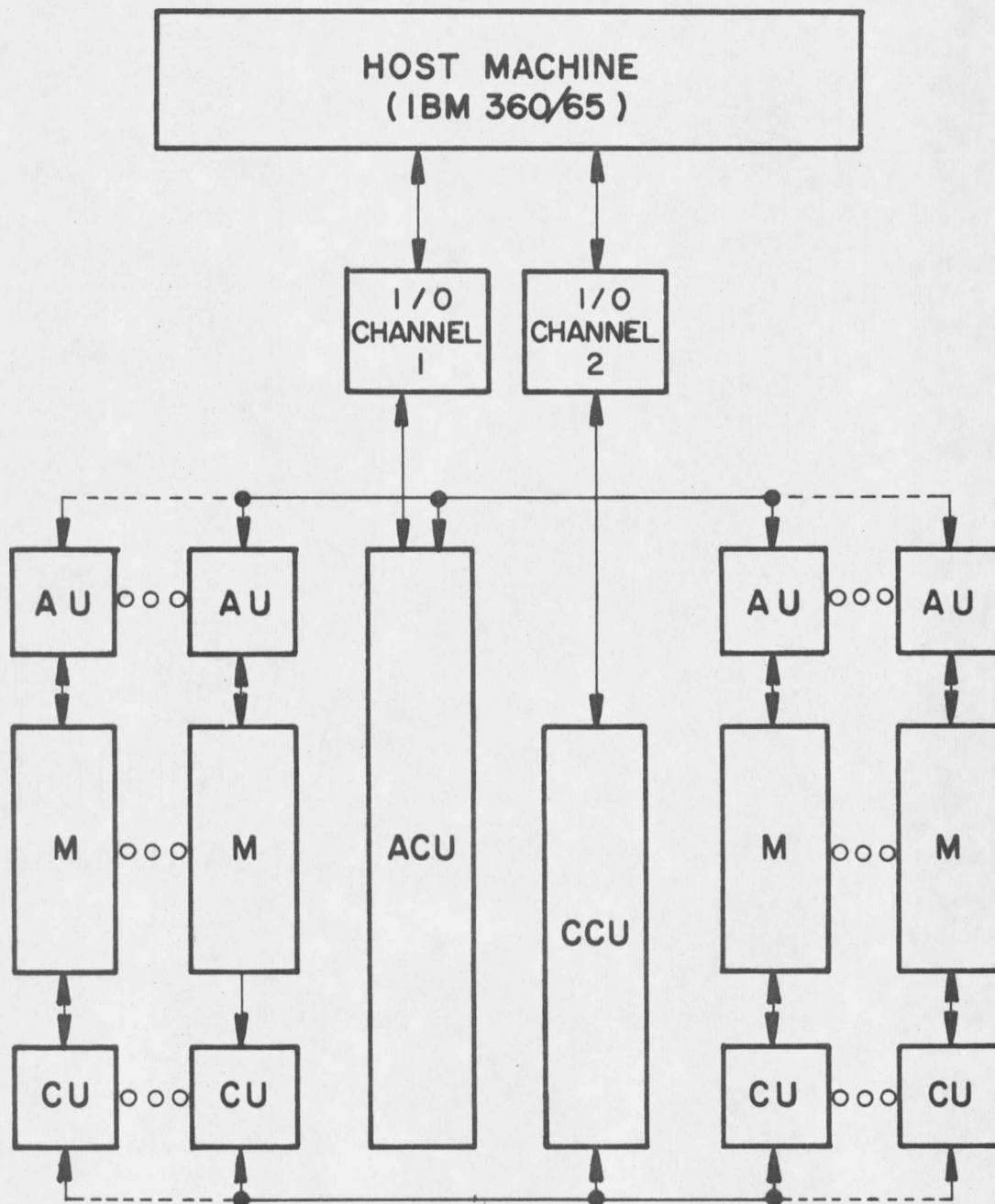


Figure 1.9 General Organization of the PEPE Computer.

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 achieved) 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 cells 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.

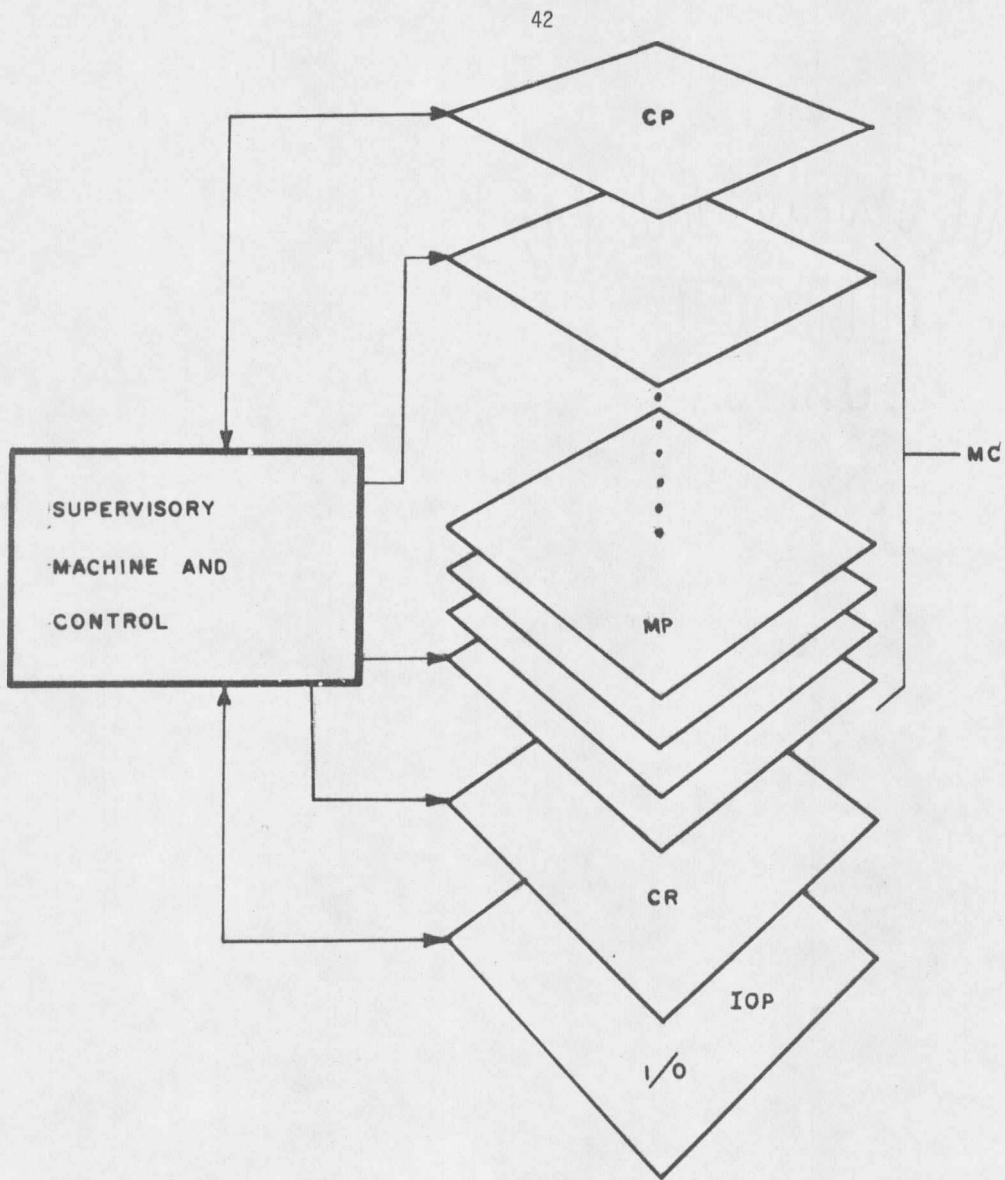


Figure 2.1 Physical Relation of Major 3-D Machine Subsystems.

## 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

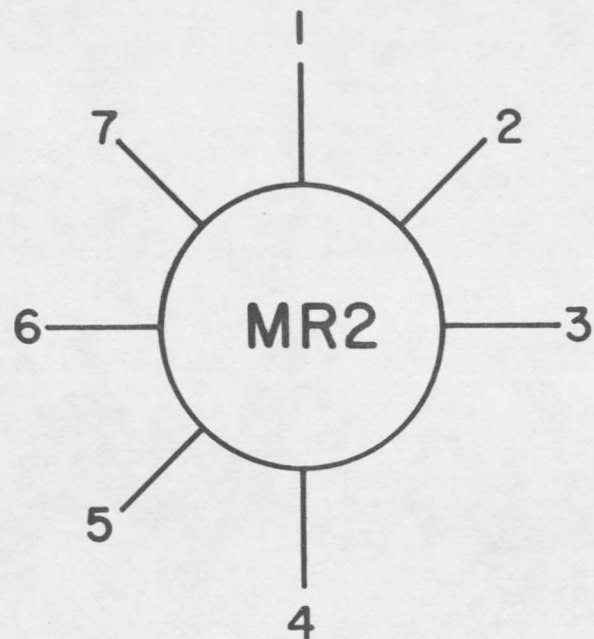
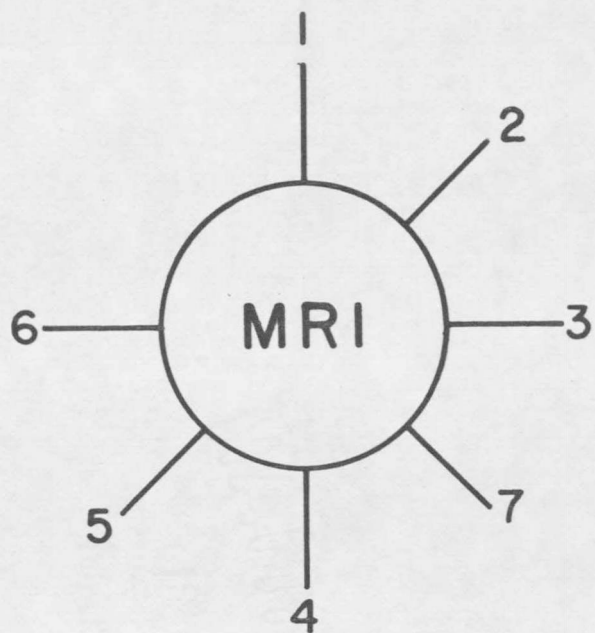
structure for the machine, providing geometrically useful data transfer paths that can be highly effective in prearranging data for certain types of arithmetic and logical operations. Data transfer in the 3-D machine is innovative, not only in the fact that the data transfer mechanism is a structure 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.



48

Figure 2.2 Interplane transfer possibilities for MR1 and MR2.

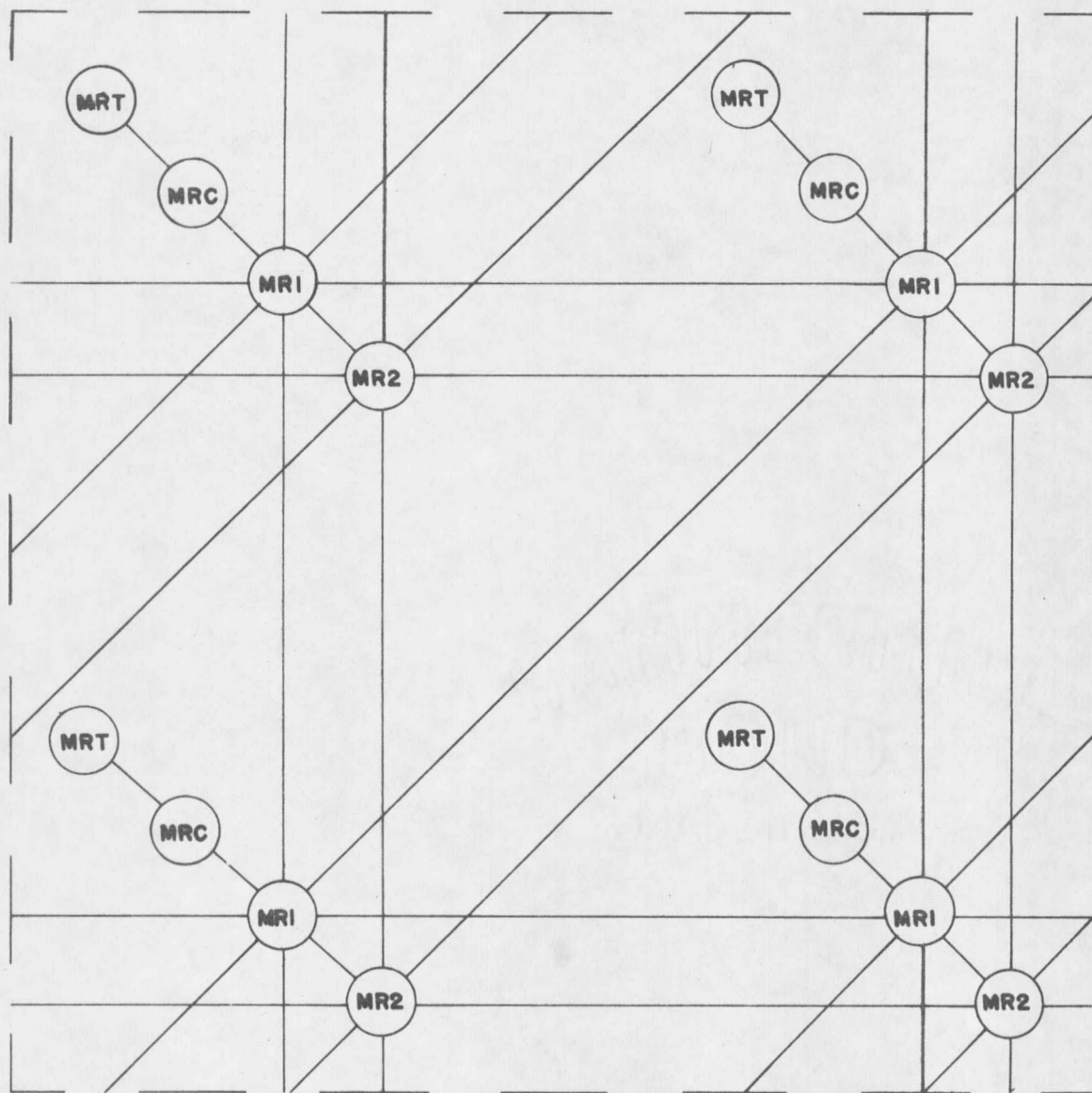


Figure 2.3 A 2x2 Memory Plane.

### 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 instruction 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 is 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







































































































































































































































































































































































































































































































































































































