

https://dspace.org/

Ramrod: an experimental multi-microprocessor

Rabinowitz, Alan Errol

2015-07-02
http://hdl.handle.net/10539/18045

Downloaded from a DSpace repository. https://dspace.org/

R A M R O D : A N E X P E R I M E N T A L M U L T I - M I C R O P R O C E S S O R

Alan Errol Rabinowitz

A Thesis Submitted to the Faculty of Engineering
University of the Witwatersrand, Johannesburg,
for the Degree of Doctor of Philosophy.

Johannesburg 1982.

This thesis is dedicated to the everlasting
memory of my loving mother
Anne Rabinowitz B.T.h.
Who departed on 22 Adar 5711.

DECLARATION

I hereby declare that this thesis is my own work and that it has not previously been submitted for a degree at any other university.

Alan Errol Rabinowitz

23rd Day of December , 1982

ABSTRACT

The computer architect of the 80's faces an apparently intractable dilemma: Computer manufacturers have to contend with the soaring costs incurred in producing custom-made chips, and would prefer to use commercially-available, state-of-the art, large-scale integrated circuits. Product users, however demand highly reliable, realistically- priced systems which are nevertheless flexible enough to meet changing needs.

It is generally accepted that to be reliable and flexible a system should be conceptually simple and inherently fault-tolerant. Further, accepting the necessity for maintenance, it becomes clear that the architecture should be totally modular both for hardware and for software.

This thesis is an attempt to reconcile these seemingly conflicting demands. An architecture is proposed, based on the frequently-used principle of closely-coupled multiprocessors, which avoides the pitfalls of over-complexity and too-heavy software dependance.

The proposed system is inherenty simple, making use of a single, high-speed time-division multiplexed bus to provide for communication between processors and memory. Software complexity is reduced by adopting a distributed, hardware-oriented operating system. Simplicity is enhanced by the use of a unified memory structure, whereby the user may freely allocate local or global memory, or a mixture of both.

Of importance is the use throughout of commercially-available, large- scale integrated ci.cuits. This is particularly relevant as the work was undertaken in isolation from the centres of research into custom-made microelectronics.

The author has developed the proposed system to prototype level. The prototype has been subjected to a series of performance evaluation tests , and the results obtained prove the viability of the technique adopted, and demonstrate its promise for the future.

The proposed system is inherenty simple, making use of a single, high-speed time-division multiplexed bus to provide for communication between processors and memory. Software complexity is reduced by adopting a distributed, hardware-oriented operating system. Simplicity is enhanced by the use of a unified memory structure, whereby the user may freely allocate local or global memory, or a mixture of both.

Of importance is the use throughout of commercially-available, large-scale integrated circuits. This is particularly relevant as the work was undertaken in isolation from the centres of research into custom-made microelectronics.

The author has developed the proposed system to prototype level. The prototype has been subjected to a series of performance evaluation tests, and the results obtained prove the viability of the technique adopted, and demonstrate its promise for the future.

ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to the following people without whose help this thesis would not have been written.

Professor M.G. Rodd, his supervisor , Head of the Department of Electrical Engineering, University of the Witwatersrand, for his guidance, enthusiasm, interest and unselfish help in the writing of this thesis and advice on solution of technical problems and for the opportunity to be able to 'mirror' his thoughts.

Susan, his wife, for her patience, tolerance , enthusiasm and encouragement throughout his 'career' as a student and her help in preparing the diagrams.

Riva Rachel, Aviva Esther and Yona Mordechai, his children, who helped provide a reason for completing the work

Ralph and Anne Rabinowitz, his parents, for their material and moral support.

The technicians of the Department of Electrical Engineering for their contribution to the technical work and 'repairs' done to the project.

Gus Finucci and Johann Lambrechs, his colleagues, for their help through tight spots

Sue Rodd for the task of making the thesis legible and intelligable.

Finally the author wishes to thank the University of the Witwatersrand for the use of equipment , the CSIR for providing a grant for purchasing components, and Perseus for providing the author with a Research Fellowship.

GLOSSARY OF TERMS AND ABBREVIATIONS

CCU	Computer Control Unit
EPROM	Electrically Programmable Read Only Memory
TTL	Transistor-Transistor Logic
CPA	Central Processor Array
TDM	Time-Division Multiplex
CPU	Central Processing Unit
ECL	Emitter Coupled Logic
ALU	Arithmetic and Logic Unit
RAM	Random Access Memory
MOS	Metal Oxide Silicon
IC	Integrated Circuit
DC	Direct Current
VDU	Visual Display Unit
ns	nano-seconds
ma	milli-amps 1/1000 amps
V	Volts
nano	10**-9
milli	10**-3
micro	10**-6
Kilo	10**3
mbytes	mega bytes
mega	10**6
VLSI	Very Large Scale Integrated

CONTENTS

PAGE

ABSTRACT.....(iii)

ACKNOWLEDGEMENTS..... (v)

GLOSSARY OF TERMS AND ABBREVIATIONS.....(vii)

LIST OF GRAPHS, TABLES AND FIGURES.....(xii)

1 INTRODUCTION..... 1-1

1.1 The Growing Demand for Computing Power..... 1-1

1.2 Distributed Control..... 1-3

1.3 The Influence of Technology on Architecture..... 1-15

1.4 Software..... 1-18

1.5 Multiprocessors..... 1-20

1.6 Ramrod: a Multiprocessor Architecture 1-26

1.7 Conclusion..... 1-27

2 MULTIPROCESSORS..... 2-1

2.1 Multiprocessor Structures..... 2-1

2.2 Interconection Strategies..... 2-4

2.3 Shared Memory..... 2-13

2.4 Time-Division Multiplexed Bus..... 2-14

2.5 Supervisor Control..... 2-17

2.6 An Overview of Ramrod..... 2-17

2.7 Conclusion 2-20

3 A REAL-TIME OPERATING SYSTEM FOR Ramrod..... 3-1

CONTENTS

PAGE

3.1 The Role of an Operating System..... 3-1
3.2 Multiprocessor Operating Systems.... 3-3
3.3 The use of the Operating System in Ramrod..... 3-5
3.4 Basic Structure of the Operating System..... 3-7
3.5 Inter-Task Communications..... 3-12
3.6 User Task to Operating System Communication..... 3-16
3.7 Conclusion..... 3-16
4 HARDWARE STRUCTURE..... 4-1
4.1 System Overview..... 4-2
4.2 Basic Structure of Ramrod..... 4-6
4.3 Physical Construction..... 4-32
4.4 Conclusion..... 4-35
5 IMPLEMENTATION OF THE OPERATING SYSTEM..... 5-1
5.1 Operating System Kernel..... 5-2
5.2 Local Operating System..... 5-5
5.3 Conclusion..... 5-8
6 SOFTWARE STRUCTURE..... 6-1
6.1 Data Flow Approach..... 6-2
6.2 Task Definition..... 6-6
6.3 Inter-Task Communication..... 6-8

CONTENTS

		PAGE
	6.4 Conclusion	6-15
7	EVALUATION OF SYSTEM	7-1
	7.1 Practical Limitations	7-2
	7.2 Factors Influencing the Relative Comparison	7-4
	7.3 Program used in Relative Comparison.	7-6
	7.4 Results	7-8
	7.5 Conclusion	7-12
8	CONCLUSION	8-1
	8.1 Uniqueness of Ramrod	8-2
	8.2 Commercial Viability of Ramrod	8-4
	8.3 Critical Analysis of Ramrod	8-5
	8.4 Future Enhancements	8-6
	8.5 Conclusion	8-7

CONTENTS PAGE

APPENDICES.....

A Emitter Coupled Logic..... A-1

B Microprogramming Bit-Slice Technology. B-1

C The Modelling of a Circular Bus..... C-1

D Currently available Multiprocessors... D-1

E Input/Output interfacing..... E-1

F The Exoslice Development System..... F-1

G The Circuitry..... G-1

H Marketing Costs of Ramrod H-1

I High Level Description of Programs.... I-1

J Reliability..... J-1

REFERENCES..... R-1

Index to Figures, Tables and Circuit Diagrams

Figures:

No.	Title	PAGE
2.1	Typical Multiprocessor	2-3
2.2	Shared-bus	2-5
2.3	Crossbar switch	2-9
2.4	Multiport Memory	2-12
2.5	System Diagram	2-19
3.1	Slave Processor to memory segement Pairing	3-11
4.1	Ramrod block Diagram	4-3
4.2	Timing on the Shared-Bus	4-5
4.3	Circular Construction	4-14
4.4	Microinstruction format	4-23
4.5	Master Controller	4-31
4.6	View of Ramrod	4-34
6.1	Data flow Instructions	6-5
7.1	Operating System Sequence	7-7
7.2	Execution Sequence	7-9
A-1	ECL Structure	A-4
A-2	Series Gating	A-4
A-3	Collector Dotting	A-5
A-4	10804 latch	A-5
B-1	Conventional and Microprogrammed Computers	B-3
B-2	Typical Microprogrammed Computer	B-5

No.	Title	PAGE
C-1	Thevenin Equivalent of Driving gate	C-5
C-2	Capacitor Resistor network	C-15
D-1	Cyba-M	D-2
D-2	Siemens 4004/220, 230	D-4
D-3	Siemens 201	D-6
D-4	Cmmp.	D-8
D-5	The Banyan Multi-Microcompter System	D-10
D-6	The Intel 432 System	D-13
F-1	Microinstruction execution steps	F-7
J-1	Serial reliability	J-5
J-2	Parallel Reliability	J-5
J-3	Composite Reliability	J-6
J-4	Ramrod's Reliability	J-6

Circuit Diagrams

No.	Title	
G-1	Microprocessor Module	G-2
G-2	ECL latch Module	G-4
G-3	Memory Module	G-6
G-4	Control Board	G-8
G-5(a)	Central Processor Array	G-10
G-5(b)	Input/Output	G-11
G-6(a)	Computer Control Unit	G-13
G-6(b)	Pipeline Registers	G-14

Graph:

No.	Title	PAGE
1	Failure Rate of Components	1-8
2	Average task delay time as a function of task Characteristics	1-21
C-1	Comparison between predicted and observed results	C-7
C-2	Comparison of rising edges for various termination resistors.	C-9
C-3	Comparison of falling edges for various termination resistors.	C-10
C-4	Comparison of the Voltage cross-section on the bus at various times for various termination resistors.	C-12

Tables

No.	Title	
1.1	Cost/Performance ratio	1-4
1.2	Reliability of Components	1-6
7.1	Execution Times	7-11

CHAPTER 1

INTRODUCTION

"An I directed my heart to know wisdom, and to know madness and folly ,but I have perceived that this also is a torture of the spirit. For where there is much wisdom there is much vexation, and he that increaseth knowledge increaseth pain" [Eccelesiastes i 17,18].

This thesis proposes a technique for interconnecting a large number of microprocessors to form a simple, inexpensive but efficient computer system. The system is inherently modular thus enhancing reliability, maintainability, and testability.

1.1 The Growing Demand for Computing Power

In order to cope with the rapid advance of technology and the ever-increasing demands of society, particularly in respect of automation, there is a need for the provision of more computing power at lower cost. One need only to look at fields such as those mentioned below to see that the 'supercomputer' is very much in demand.

Short-range weather forecasting requires very accurate and highly complex weather modelling. Computer assisted tomography (CAT), which involves high-speed signal processing and imaging, as well as the modelling of organs such as the heart, needs advanced equipment for computing at speeds approaching 100 million floating point operations (megaflops) per second [SLA]. Nuclear fusion researchers could use a computer 100 times faster than any existing machine for modelling the plasma instabilities of proposed fusion power generators [SUG 80].

One of the world's most complex undertakings in the past two decades has been the USA Department of Defense (DOD) Ballistic Missile Program. A critical part of the large research and developement investment in this program has been the effort to develop data processing hardware and software technologies to meet the computational challenges of this complex problem. The Ballistic Missile Program needs a computing system that will deliver a throughput of hundreds of megaflops per second, with a high degree of confidence that correct execution will occur. This challenges even the most advance technologists. [DAV 80]

The computer engineer, who takes on himself the burden of designing such a machine faces a great challenge. He must bear in mind that a computer is ultimately designed for the end user, and it is the user's evaluation that counts, as it is he who will be in the most intimate relationship with the computer. The computer, therefore, has to be user acceptable in terms of reliability, maintainability and safety.

1.2 Distributed Control

Amongst the many criteria which determine the choice of a particular design, is that of overall cost. A feel for this criteria may be established in table 1.1. In this Sugarman compares the processor cost/performance for a particular sample problem requiring 83000 flops for each iteration. It can be seen that the AP-120B peripheral array processor is 7 times as cost-effective as its nearest rival the CRAY-1 supercomputer [SUG 80].

Machine	Mflops	$/flop	installation cost in $M
AP-120B	5.9	.03	.15
CRAY-1	38.4	.21	8
STAR-100	16.8	.48	8
VAX 11/780	.26	.77	.2
CDC 7600	3.3	.91	3
ILLIAC IV	9.1	1.1	10
CDC 6600	.63	1.59	1
IBM 370/165	.87	2.3	2

TABLE 1. 1 COST/PERFORMANCE RATIO [SUG 80]

(A MegaFlop(Mflop) is a million floating point operations.)

Of interest from the above comparisions, is the observation which may be made that parallel processors, which are cheaper than supercomputers, can be used in situations such as those mentioned previously.

1.2.1 Reliability

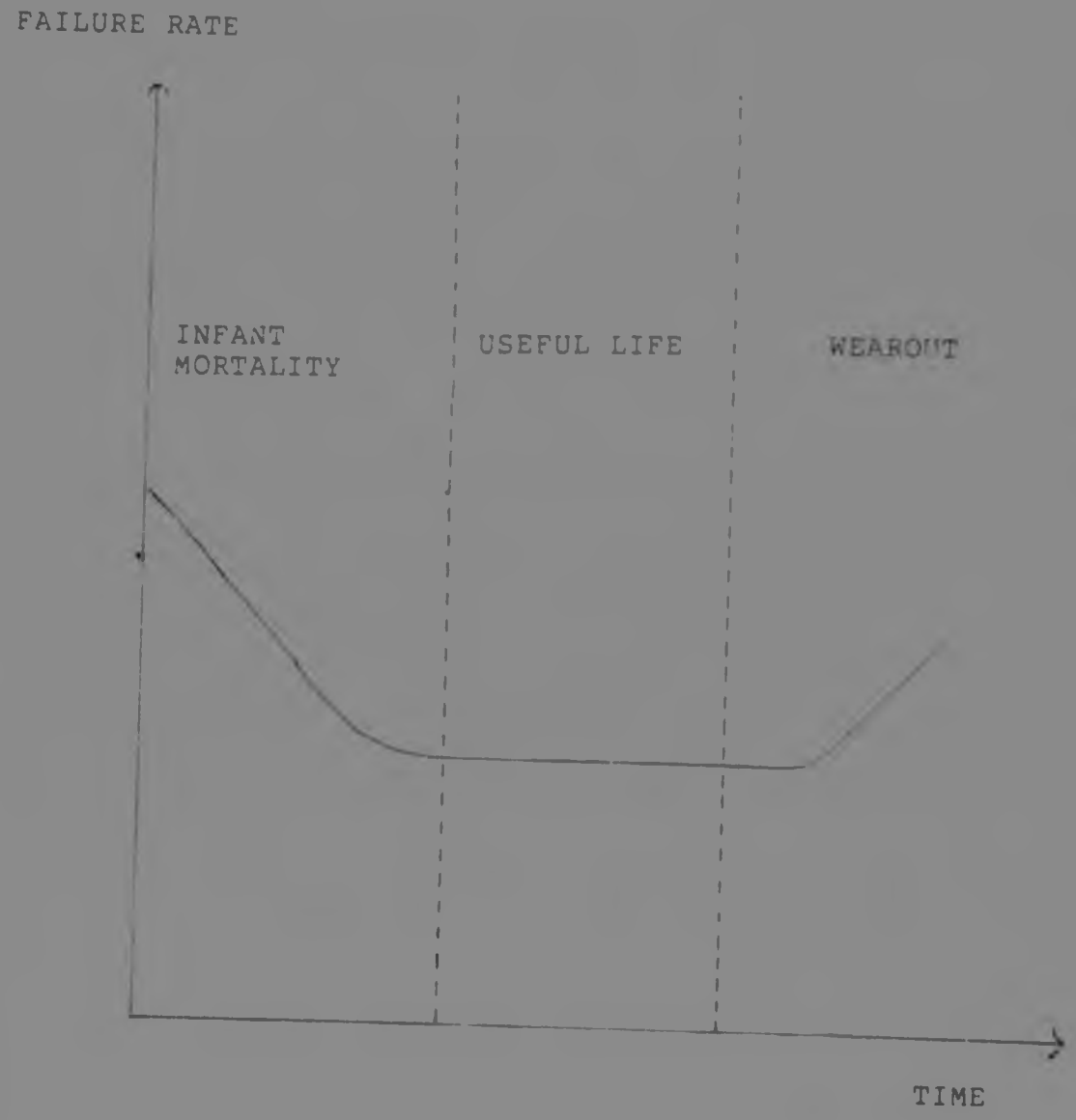
An essential trade-off to be considered in computer design is the complexity of the computer versus the power (or throughput). It is common knowledge that the more powerful a computer is, the more complex it becomes [SUG 81]. It is also common knowledge that complex electronics, unless highly integrated, becomes increasingly unreliable and costly. This is easy to explain:- From table 1.2 it can be seen (a) that the reliability of a computer board decreases 40-fold when compared to tne reliability of a single integrated circuit because of the increase in components and complexity, and (b) that the reliability decreases dramatically as the number of components per system increases.

Component	Order of Magnitude of Fits		
Transistor	10	to	100
LSI component	100	to	1000
Solder connection	2	to	20
Switch (per contact)	30	to	300
Plug connection (per contact)	30	to	300
1 Board computer (25 chips)	4000	to	40000

Table 1. 2 Reliability of Components [KOP 81]

(1 FIT = Failure in 10**9 hours i. e. 115000 years)

Graph 1 shows the characteristic curve for an electronic device. Early failures, such as infant mortality or burn in failures, occur at a high initial rate which decreases when the weak units have died out. The useful life period, which is the most important period because it is the key to reliability prediction, is followed by the wearout period. Wearout failure results from degradation of the strength of a device and exposure to the environment [DOY].



GRAPH 1 FAILURE RATE OF COMPONENTS

The society in which we live is becoming very safety-conscious, and increasingly dependent on computers. Therefore a computer which has the function of, say, controlling production machines or on which we rely for the handling of critical data, has to be extremely reliable. One need only look at what happened at Three Mile Island. The nuclear reactor at Three Mile Island was controlled by hardwired logic and many small computers. There was no central controlling facility nor was there communication between the distributed control points. At the moment of crisis the operators were faced with many indicator lights and perhaps, if there had been an interconnection of the control points, the near-disaster might not have occured. For this to be available, clearly a highly reliable computing system is vital. NASA, too, has an aircraft energy effiency research program needing ultra-reliable computers that would counteract faults automatically. The aircraft flies very close to the limits of stability and therefore a computer with a fast response time is needed (rather than a human) to control it. The probability of a computer failure during a flight must be less than the probability of mechanical structure failure during the same period. Thus an ultra-reliable and fast computer is needed.

Reliability is normally defined as the "probability that a system will function within the specified limits for at least a specified time under specific environmental conditions" [KOP 81]. It is concerned with all the parts of the system (hardware, software, printed circuit boards, etc.), their interaction, the interconnection mechanisms between the various parts and finally, naturally, depends on the mechanical construction.

In striving to achieve a high degree of software reliability, problems such as software validation are encountered. Present techniques are inadequate for evaluating the reliability of software, and perhaps the only way of checking software is by exhaustive testing [LAM]. Bernhard maintains that system validation problems are primarily related to software, and that no guidelines exist for determining software reliability [BERD]. Making the software simple and well defined can help in solving these problems, but the programmer can never claim with total certainty that his program is error-free (see 1.4).

The reliability of a computer system requires a thorough investigation. Reliability involves both software and hardware and it was decided to limit the study of reliability in this thesis primarily to that of hardware (appendix J). Software reliability is dealt with on the "keep it simple and well- defined" precept.

One of the most critical factors influencing the reliability of a computer system is the interconnection structure of the system. This is because, although the reliability of the individual components can be maximized, the overall reliability of the system will be related to the component interconnections, which are not usually duplicated and are inherently unreliable (being mainly mechanical in nature).

In practice, there are various techniques available for attaining a high degree of reliability. These essentially achieve reliability through either the use of inherently highly-reliable components or through the introduction of redundancy. (Redundancy here implies that the system contains more resources than are absolutely necessary for normal operation). According to Toong [TOO] highly-reliable systems are "systems with a structure independent of any critical resource that has a relatively high failure rate."

The cost of increasing the reliability of components achieved during manufacture, is very high. Therefore fault-tolerance is normally adopted on the premise that it is more economical to build redundant systems than to strive for extreme component reliability. A fault-tolerant computer system is one which can survive multiple faults that would normally bring a conventional computer to a halt [STI].

Of importance in a redundant system is the ability to detect an error. Error detection presupposes that the result of a step in a process can be related to an acceptance criterion. In a system with redundancy, additional resources typically are used to form an error detection module which may be separated from the actual active processing modules [KOP 81].

A key issue in fault-tolerance design is the size of the unit that is to be replaced in the event of a failure - often termed the Smallest Replaceable Unit (SRU). The SRU is generally visualised as the unit which is removed by the service engineer once he has localised a fault to a particular unit, which is then replaced by a identical one. It is also clear that an SRU must be testable - specifically this requires it to have well defined interfaces[KOP 81].The SRU could be a resistor or transistor at one extreme or a complete board at the other. Since the costs of electronic components are steadily decreasing it becomes economically

feasible to think in terms of a complete board as the SRU.

From the above it may be concluded that a well-structured computer should therefore have inherent fault-tolerance built into its architecture, by having redundant components. By adopting such a design, however, it would seem that reliability is achieved at the expense of simplicity. This thesis discusses an architecture for a computer system that is reliable, partially fault-tolerant and (of importance) simple and well-structured.

1.2.2 Maintainability

The user of a system is primarily concerned with the availability of the system for his use. Availability is a function of the Mean Time Between Failure (MTBF) and the Mean Time To Repair (MTTR). As a failure occurs, the faulty module is replaced by the service engineer and the user can then carry on operating the machine as if nothing had happened. Provided the principle of fault-tolerance is adopted, however,during the diagnosis and repair time the user will simply experience a slight drop in performance.

There is clearly a trade off between maintainability and reliability - both being linked to the availability of the system (see appendix J). Maintainability, which can be defined as the probability of repair in a given time, implies that the system must be modular. If the SRU is extracted for repair, the system must be able to tolerate this removal and recover once the module is re-inserted.

A module is characterised by the function it performs. It is essentially a 'black box' which transforms a set of inputs to a set of outputs. In designing systems using modules the designer assumes that other modules, except the one on which he is working, work to specification. Testing is done in a similar easy fashion.

Of importance too, is the practical realisation that once a computer system has been installed, the user inevitably needs to increase its capacity! Enhancement of a computer system can be achieved much more readily in a well-structured, modular design.

1.3 The Influence of Technology on Architecture

The advance of technology is sometimes too rapid for the system designer, in that by the time his design is functional there may be newer and more powerful components available which might more easily accomplish his required tasks. This problem is never more apparent than in the world of electronics, and particularly, the digital area where the pace of technological innovation is staggering.

The Electronic designer has three approaches available to him when utilizing state-of-the-art digital hardware. These may be summarized as follows:-

1. The use of Custom Designed Integrated Circuits. The engineer designs highly complicated integrated circuits from the transistor junction level - normally using the support of a Computer Aided Design system (CAD). These components are then fabricated especially to meet the required function. Clearly cost is a problem unless volume is high (typically > 10000 units).

2. The use of Readily available VLSI. The engineer attempts to utilise integrated circuits which have already been manufactured and which perform specific functions.

3. The use of Semi-Custom Logic (e.g. a Logic Array). In this technique the integrated circuit manufacturer produces a chip which is complete from the semiconductor point-of-view, but which lacks the final interconnection of the various logic functions that are performed by the semiconductor junctions. Thus the designer of a circuit typically has two to three thousand logic gates available for his design. Using CAD techniques, he then creates a system using only the types of components available on the particular array chip in which he is interested. Once again, using CAD facilities, the designer optimises the interconnection of the gates to give himself a system which meets his requirements. The final interconnection of the components (the metalisation process) is relatively cheap, and the approach is is cost-effective for a medium level of production [ROD 82].

In addition to the points mentioned above, a fundamental premise in design is that a designer should strive to utilise state-of-the-art technology; this, of course is in itself a situation requiring much thought. A case in point is the Josephson Junction. Conference papers continue to be delivered on this technology but the scientific world still waits for a commercial computer based on Josephson Junctions.

Josephson devices, which are based on superconductivity and tunnelling, are very attractive for ultra-high-performance computers. They are extremely fast-switching (<10 pico seconds) have extremely low power dissipation (< 500 nano watts per circuit). However, they have to operate at near the Absolute Zero temperature (-270 deg C), so that they can function according to the specifications. This temperature requirement causes undue environmental complexity as well as additional costs for refrigeration, and inconvenience of system debugging and servicing [ANA 80].

Therefore even though Josephson Junctions are undoubtedly superior in most aspects to any other logic family available, there is a natural reluctance amongst computer designers not to use this technology until it has been proven and tested.

An important factor which has to be considered is the local situation. As the work for this thesis took place in relative isolation from the centres where electronic technological advances are normally made, the decision was made to design a completely modular system based on locally available technology. This ruled out the use of Custom designed circuits, as well as that of logic arrays - this latter industry being still in its infancy in South

Africa[NOV]. However the majority of leading Integrated Circuit Producers are represented in the country, and thus the bulk of commercially available components could be considered.

Finally from a maintenance point-of-view the approach adopted appears to have much merit. One has always to ensure that the local maintenance personnel can cope with the technology they are servicing; also that replacement components are readily available.

1.4 Software

The complicated aspect of software reliability has not been dealt with in detail, as it is beyond the immediate scope of this particular investigation.

However, a few general guidelines which should be adhered to in attempting to produce reliable software have formed the basis of all software developed in this project. These are as follows:-

1. The specification of the program should be kept simple and accurate and must be well documented to allow non technical persons to understand the software

2. The software must be well designed with clear meaningful documentation,in order to reduce effort in testing and maintenance.

3. The software must preferably be organised in a tree structure, in order to make reading and understanding easier.

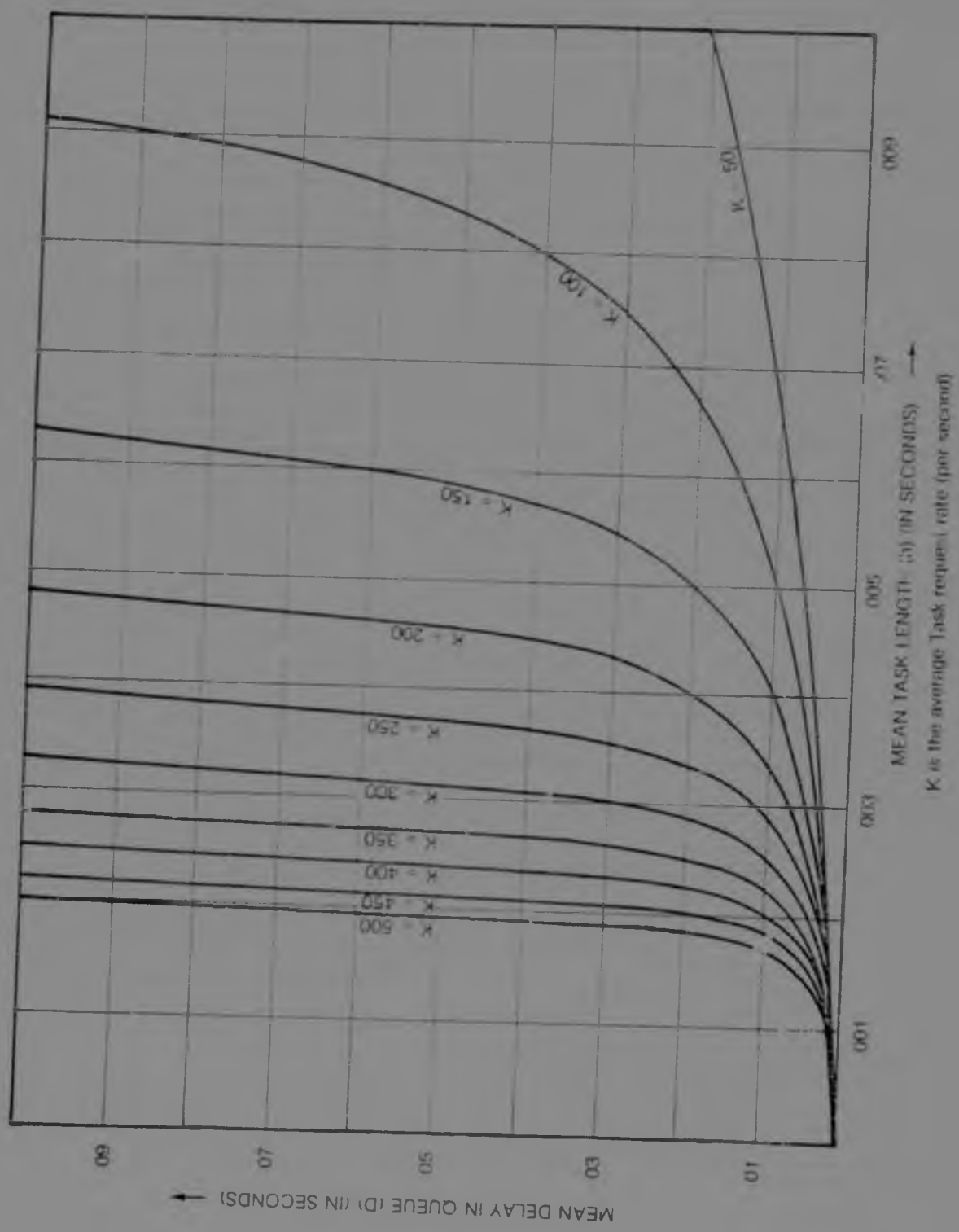
4. The software should be written in a modular structure with loose coupling between modules, so that any module can be extended, replaced or removed without affecting the other modules.

5. The software should be designed in a top-down fashion, which first describes the problem in a very high-level way, and then proceeds to give lower levels of description until the level is reached which contains definitions of indivisable functions [LAM].

Because most of the operating system has been implemented in hardware, and only a minimal amount of software (designed using the above principles) is required to complement the operating system, it is felt that this approach ensures a relatively high degree of software reliability. However it must be emphasised again that this aspect of reliability was not considered in detail in this thesis.

1.5 Multiprocessors

Rodd has applied the theory of traffic movement through a telephone exchange to the analysis of the performance of multitasking industrial control computers and has produced a graph (Graph 2) which mirrors the expression derived for the mean delay experienced by a task in a queue which can form in a multitasking computer system [ROD 76]. This was done in order to predict the performance of the system. As can be seen, the delay time in the queue increases as the average request rate increases. There are clearly various ways to increase throughput of a system as may be deduced from these curves:



GRAPH 2 AVERAGE TASK DELAY TIME AS A FUNCTION OF TASK CHARACTERISTICS [ROD]

1. Make the task length shorter, i.e. simplify tasks or increase speed of processing

2. Decrease the average request rate, i.e. reduce the demands made on the processor

3. Make the computer faster, i.e. increase overall operating speed (as well as achieving (a) above).

4. Increase the number of processors, i.e to decentralize the processing.

Powerful, large-word-size mainframe computers and supercomputers have made high-data rate processing feasible, but these systems are not economical for laboratory environments, data acquisition, process plants or reduction applications. Minicomputers on the other hand are economical, but are technically unsatisfactory because of their limited computing speed and smaller fixed-point formats [ALE 81]. Therefore the microprocessor, which is cheap, and which can be interconnected to form a powerful multiprocessor computer, can fill the gap left by the two other computer systems.

Decentralisation of a computer system implies that there is a distribution of intelligence (i.e. processors). As has been seen in section 1.1, parallel processors (of which the multiprocessor is one type) compare very well with 'supercomputers' on a cost/performance ratio. There is less reliance on a centralised facility, and processors can be added on a more flexible basis and in smaller increments.

Multiprocessors inherently rely heavily on parallelism to enhance throughput and computation. With such a hardware structure many elementary data routing and processing functions can be implemented concurrently, improving total processing speeds by 10 to 100 times over typical minicomputers.

A multiprocessor architecture increases productivity through parallel processing, and maximises the likelihood that a processor will be available when it is requested. The system can generally be tailored to user requirements in a more flexible manner than can a centralised facility, because each processor in the system can be used to perform a separate function.

A multiprocessor computer should also be inherently modular and therefore the cost of increasing its processing capability is smaller than that incurred when expanding a large computer. Redundancy at a hardware level is naturally easier in a multicomputer than in a monolithic central facility as extra modules (which are added on to take an active or passive part in the system) can take over the execution of a task in the event of a processor failure.

As has been pointed out in appendix J, a multiprocessor system that has redundant units is ideally more reliable than a uniprocessor system. An additional factor to consider when designing a redundant modular system is that the system should 'gracefully degrade'. This idea is illustrated in the following example. An on-line airline booking system is a distributed computer with user terminals in each booking office and with a centralised data base. The failure of any terminal should not inhibit other users from accessing the common data base. This is usually referred to as "graceful degradation" in that failures will accumulatively affect the overall system performance but not cause immediate and total system failure.

In such a system reconfiguration is, however, necessary when a permanent error, like a processor failure, occurs. At the conception of a redundant system it has therefore to be decided at what level redundancy is to be implemented - at system level, subsystem level or at a component level (as in the to discussion of SRU above). Therefore it is logical to make the SRU (i.e. a complete board) the redundant component as well.

From the previous sections it may be seen that the choice of components of a multiprocessor is critical. As mentioned in 1.2.1 the SRU should be a complete circuit board. A microprocessor computer board will provide a convenient basis for reconfiguration after an error and should therefore be the SRU.

It can therefore be concluded that a multiprocessor computer is a simpler alternative to a bigger computer in most applications.

1.6 Ramrod: A Multiprocessor Architecture

Ramrod, as the multiprocessor structure developed in this thesis has been named, was designed using a master-slave approach as it was felt that there was a need to provide for supervision of the slave processors with respect to their intercommunication, execution of tasks and probable failure. This is of particular importance in an experimental system which Ramrod essentially is.

For this reason it was concluded that the master had to be more sophisticated and more powerful than the actual processors. Therefore the master was designed using bit-slice technology and the instruction set was custom built to suit the application (see 4.2), whilst the slave processors were selected to be simple, single-board computers. Using bit-slice technology for the master implies that the designer has complete control over the architecture and many features are therefore included to provide this with properties inherent in operating systems.

1.7 Conclusion

Many inexpensive and relatively powerful single-board computers are currently available on the market and can therefore form the SRU of the multiprocessor system. In the event of a processor failure, the faulty processor board can be replaced by a working one, and redundancy achieved at the same level. The multiprocessor system can have redundant idle CPU boards ready to take over should a processor failure occur.

This approach to architecture is currently receiving much attention: a leading German Computer Architect Wolfgang Giloi maintains that "The distributed multiprocessor system is the only known architectural form that can satisfy high cost effectiveness, modular extensibility, fault tolerance and simplification of software production and maintenance simultaneously" [GIL BEHR].

Any multitasking computer system has its activities co-ordinated via an operating system. In the case of a multiprocessor the operating system may itself be distributed with a part of its functions performed by the master processor and other parts by the various slave processors. This should result in a highly efficient computer system as there is only partial reliance on the master processor, and each processor shares in the execution

of the operating system [TRAKH]. Of interest is the implication that the various component parts of the operating system can themselves be executed truly in parallel!

As will be shown in the next chapter, the preferred interconnection strategy for a multiprocessor is a shared bus in which the processors access common memory. As will be shown this is an optimal solution despite claims that a shared bus has serious bandwidth limitations.

Ramrod has such a shared-bus structure with a wide bandwidth, this having been achieved by a technique that appears to be novel. Ramrod has been designed, built and tested. The prototype, although suffering from certain timing problems, has been succesfully evaluated and the methods used are shown to be viable. The result is a multiprocessor system which makes use of commercial well-understood computing elements and which is reliable, modular and easy to maintain.

CHAPTER 2

MULTIPROCESSORS AND AN INTRODUCTION TO RAMROD

"Two are better than one because they will have a good reward for their toil. For if they fail the one will lift up his fellow, but woe to the single one that falleth for he hath no companion to lift him up " [Eccl iv 9,10]

Before dealing with the actual scheme adopted in Ramrod this chapter will provide a general background to multiprocessors and the various possible strategies which may be used. The interconnection philosophy of Ramrod will then be discussed in this light.

2.1 Multiprocessor Structures

A multiprocessor typically has the following attributes:

1. The system contains two or more processors of comparable capabilities.

2. All processors share access to common memory, but may have local memory.

3. All processors share access to Input/Output channels, control units and peripheral devices.

c. The entire system is ideally controlled by a single operating system. [ENS 74]

Innerent in this definition is the concept of a multiprocessor system as a so-called 'tightly coupled' distributed computer. This implies that the various processors in the system are in close proximity to each other and have access to a common memory and common Input/Output system.

A typical multiprocessor will take the form shown in Figure 2.1. Processors (P1-Pn) are connected to Memory Elements (M1-Mn) or other peripheral devices. Thus communication between the processors and resources (mem, I/O, peripherals) is an essential component of such a structure. The path is often referred to as the Processor/Memory switch.

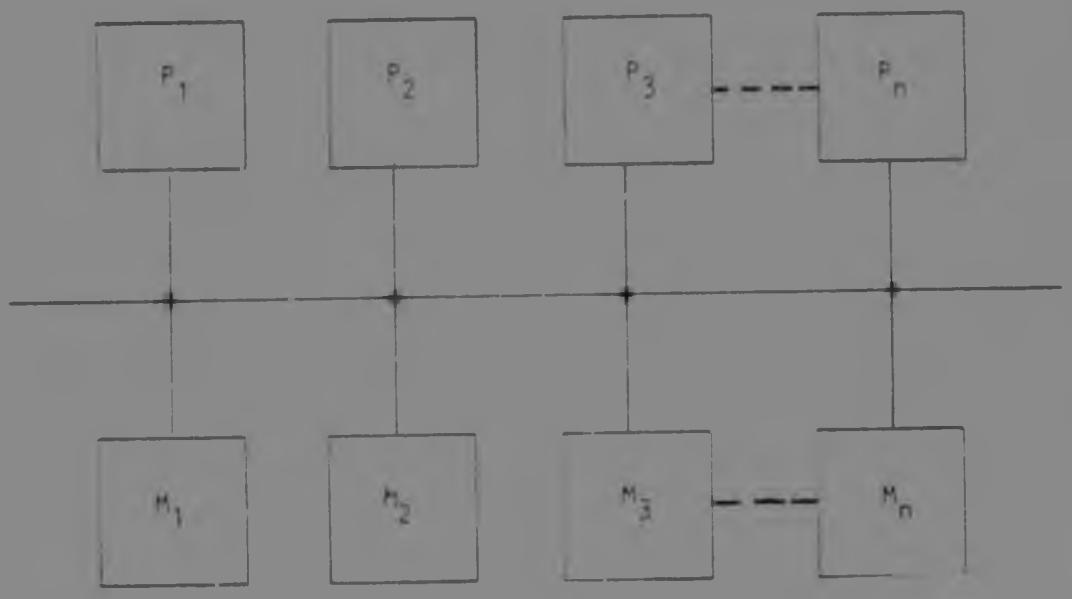


FIGURE 2.1 TYPICAL MULTIPROCESSOR

The following sections provide more detailed descriptions of the major interconnection technologies and their advantages and disadvantages. The shared bus, the cross bar switch and the multiport memory are compared in terms of cost, reliability, system throughput and transfer capacity. Discussions of systems using these architectures are to be found in appenndix D.

2.2 Interconnection Strategies

2.2.1 Shared Bus

The simplest switch for a multiprocessor system is a common bus connecting the units as shown in Figure 2.2.

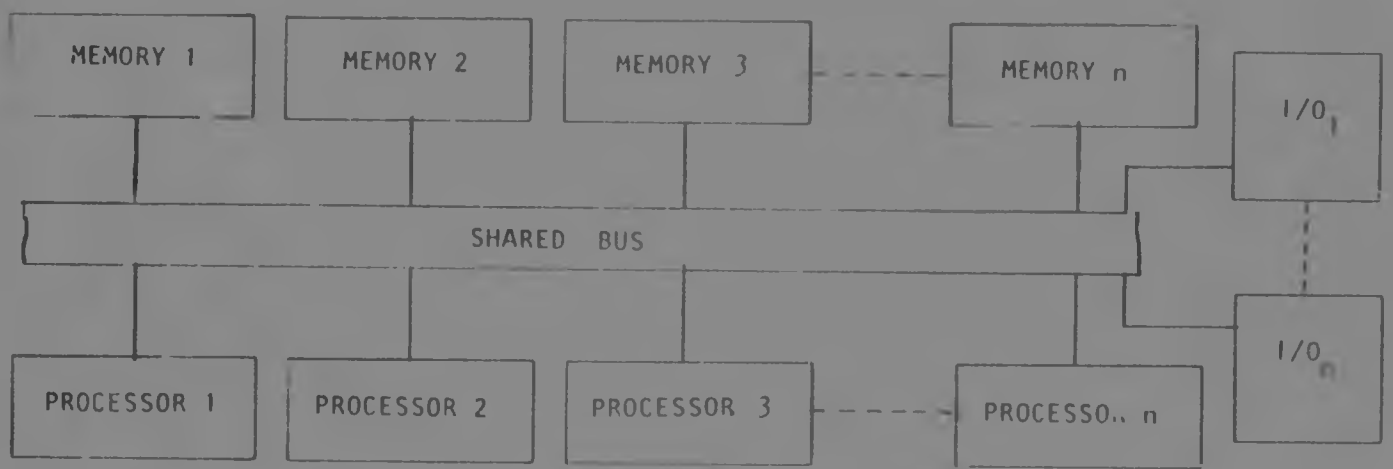


FIGURE 2.2 SHARED BUS

The shared bus can be centrally polled, i.e. the processors only transmit when selected by the controller. Bus contention is avoided by using schemes such as :

1. Fixed priorities, which allow processors with a higher priority to gain access to the bus if another lower priority processor presently has access.

2. First-in-first-out : the processor which first made the request is granted access to the bus

3. Daisy chaining: The processors are asked in turn whether they have made a request, and only then can a processor be given access to the bus.

On the other hand, the bus may be interrupt driven by the processors, which request bus usage. This scheme allows random usage of the bus. However an interrupt system can cause problems, if, during one processor's control of the bus, a second processor requests the bus, access can be granted to this second processor, and the first's data is lost. On the other hand, should all interrupts be disabled during a bus access then the requesting processors will have to wait for access and processor idle time is increased.

The bus can be a Time-Division Multiplexed (TDM) bus, where each processor is allocated a time slot, or it can be Frequency-Division Multiplexed (FDM), in which each processor has a particular transmit/receive frequency.

The shared bus is simple to design and construct, but has bandwidth limitations inasmuch as the number of active or passive units connected to it is limited [WEl 81]. This reduction in bandwidth results because when more units are connected to the bus, the bus is simply unable to keep up with the increase in communication which accompanies the addition of units [ZOC]. As there is only one path for all data transfers, the total transfer rate within the system is limited by the speed of access of devices onto the bus and the actual bus bandwidth.

The shared bus is usually connected to a common memory (indeed so are the other strategies) and therefore memory contention is also an obvious problem in that there is only one bus and one access to the memory connected on the other side, so there is a likelihood that two or more processors will try to access the same area of memory simultaneously. This problem can, however, be overcome by dividing the memory into segments, and allowing only one processor at a time to access a segment.

This scheme is the least costly in terms of the hardware used, and is also the least complex in terms of components as the bus can be totally passive. Modification is achieved simply by physically adding or removing functional units. However, a single bus system is naturally unreliable in that if the bus fails then a total system failure occurs.

2.2.2 Cross Bar Switch

The cross bar switch as shown in figure 2.3 has separate paths from the processors to each memory and I/O unit. The functional units (processors, memories and I/O) need not be concerned with the bus interface as the switch contains all the necessary logic.

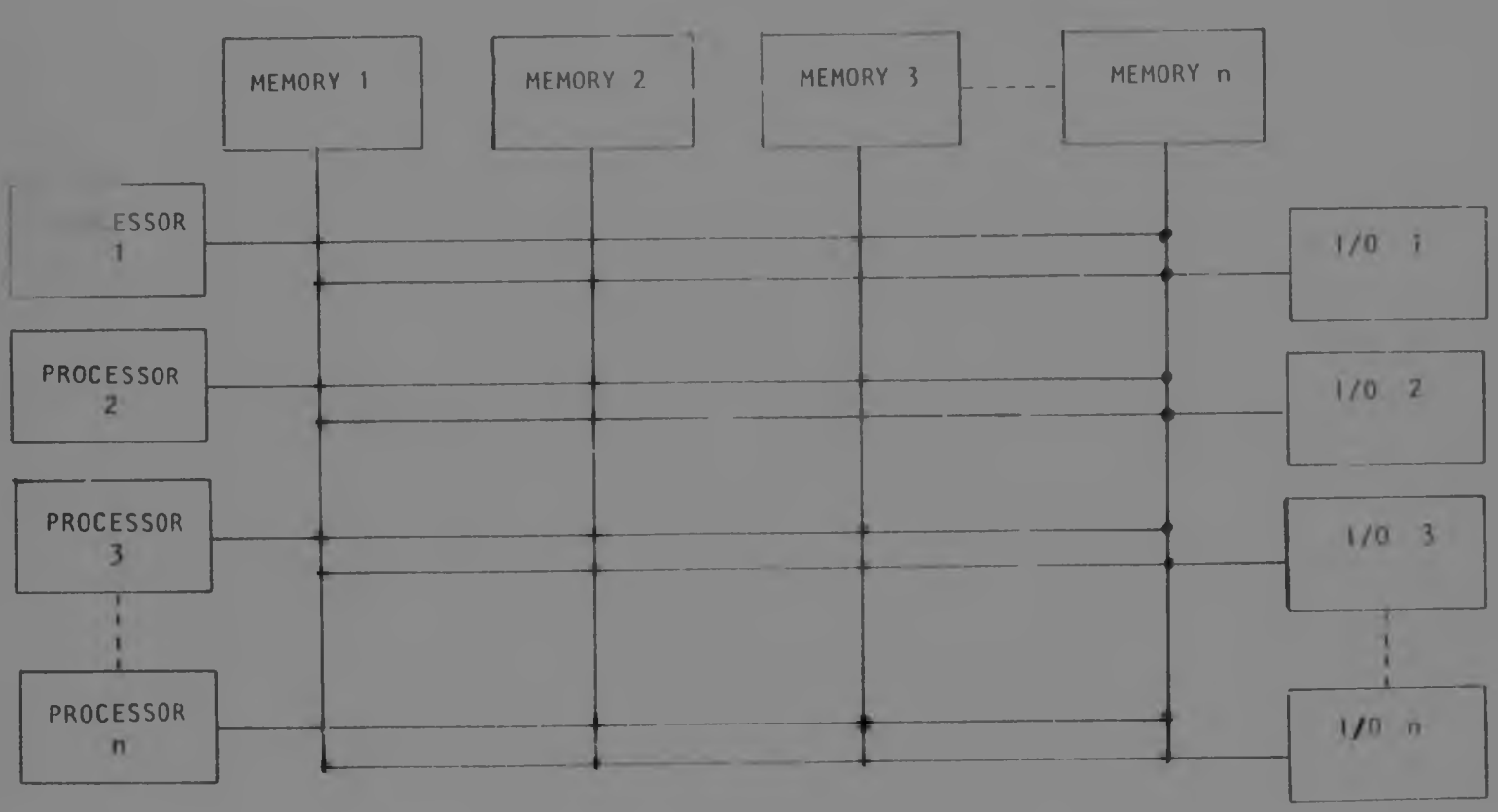


FIGURE 2.3 CROSSBAR SWITCH

This is the most complex interconnection scheme because the number of connections is necessarily large and because of the extra logic needed in the switch. The complexity grows exponentially as the number of units becomes large. Functional units, however, are simple and inexpensive because they do not need the extra logic to drive the interface and the potential exists for a high data transfer rate, since there is a separate path available to each unit.

Reliability is reasonable and can be improved by redundancy of the units. System efficiency is high because simultaneous transfers between processors and memory units can be accomplished.

Clearly the switching elements are the major drawback to such a scheme, but it must be pointed out that Intel is about to produce an LSI circuit with a large number of cross-bar switches for their new range of multi-processors [ENS 74]. This will obviously reduce the cost factor as well as the complexity discussed above.

2.2.3 Multiport Memory

In a multiport memory system the control, switching and priority arbitration are concentrated at the interface to the passive units, and not in the switch as in the cross bar scheme. Figure 2.4 shows that each processor has a separate port and bus connecting it to each memory and I/O unit.

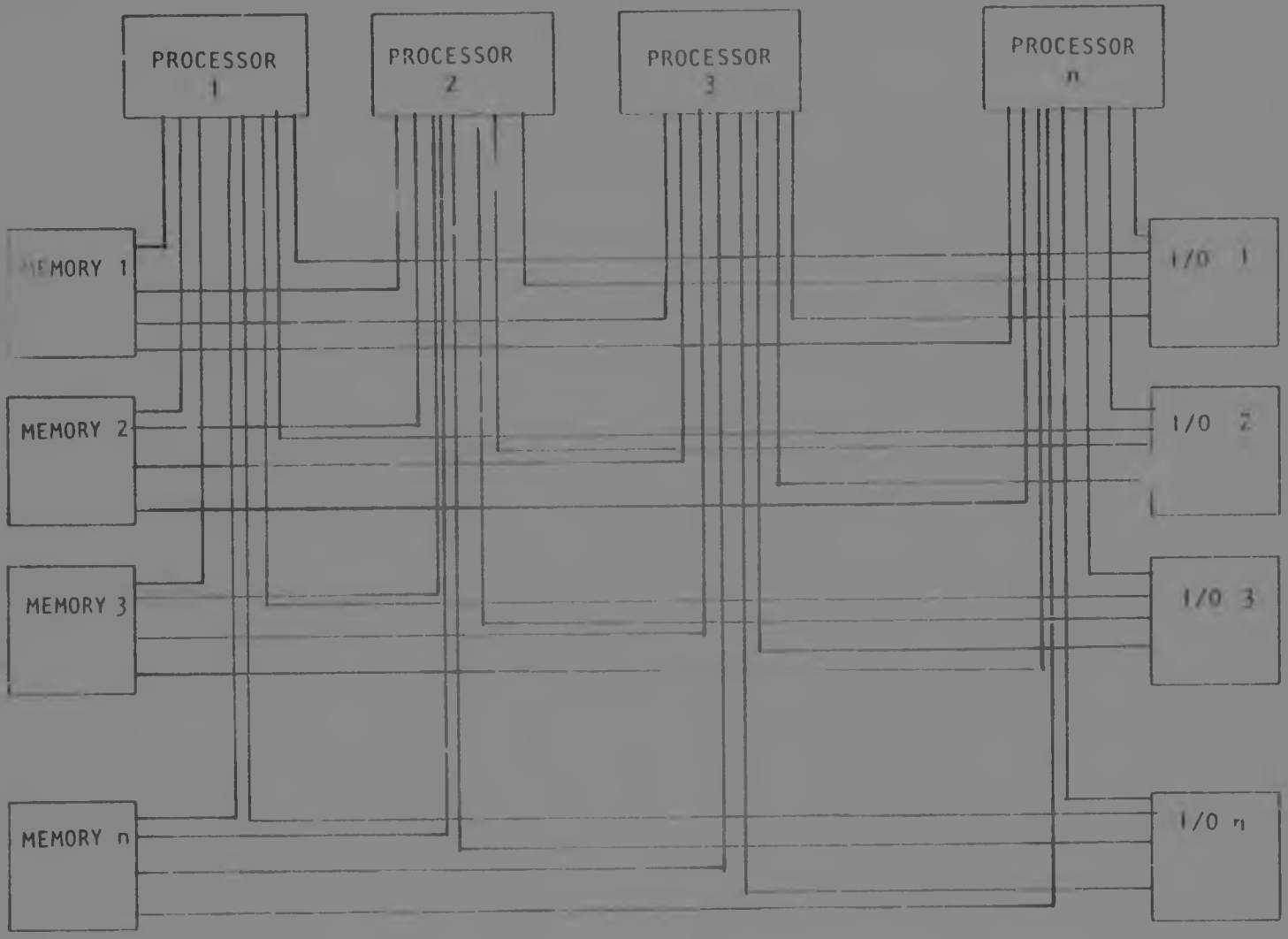


FIGURE 2.4 MULTIPORT MEMORY

This approach is the most expensive, since multiport memories are costly and each memory has to have contention logic built in, in order to arbitrate between processors competing for the resource. High data transfer rates can be achieved, but expandability is difficult as more logic is required to increase the number of memory ports, or to share the existing ports amongst all the processors.

This system has its use in a system which has a limited number of processors, but clearly becomes unwieldy as soon as the number gets large.

2.3 Shared Memory

All of the interconnection strategies, previously mentioned, usually use shared memory which provides for a means of interaction between microprocessors. This interaction can be enhanced if there is no distinction between local memory of a single processor and global memory of the multiprocessor system. This is clearly a unique feature which has the advantage of being able to incorporate a truly distributed data base, since processors can address all the memory simultaneously. However care must be taken to ensure security of data.

2.4 Time-Division Multiplexed Bus

The Time Division Multiplexed(TDM) shared bus offers the best capability of all the interconnections discussed as it is simple, cheap, easy to implement and is a passive interconnection. The apparent limitations of the TDM bus are:

1. Memory Contention
2. Reduced Bandwidth
3. Bus Contention

Fortunately these can largely be overcome as outlined below.

2.4.1 Memory Contention

A shared bus, as mentioned previously, is usually associated with common memory and therefore memory contention can occur. A system that allows the programmer to use a range of program addresses which may be different from the range of physical memories available (known as virtual memory addressing) may circumvent memory contention.

2.4.2 Bandwidth Limitations

Bandwidth limitations can be overcome by using high-speed technology to achieve a high data communication rate on the shared bus. In addition if the speed of access to the bus is increased then the overall bandwidth should also be increased.

2.4.3 Bus Contention

Examining the simple time-shared bus, it is found that contention occurs when several processors are making heavy use of the bus and when there are no mechanisms to resolve this contention (and cause processor idle time). Therefore a model of the shared bus can be made as a master/slave relationship (where each slave runs a single user task and the masters provide the requested service) in order to consider the problem of contention.

Let the slave request rate = 1/L in secs and slaves only process when serviced by a master.

Let

- N = number of slaves
- N_{avg} = average number of slaves
- M = number of masters
- M_{avg} = average number of masters

P = L/U, and Pi = probability of i slaves in queue.

N_{avg} = average number of total busy processors

W_{avg} = average waiting time in queue.

S_m = 1/U expected service time of requests

W_m = expected waiting time in queue

It can be shown [TOO 78] that

$$N_{avg} = \frac{M}{P}\left(1 - \sum_{i=0}^{M-i} \left\{\frac{M-i}{M}\right\} P_i\right) = \frac{M_{avg}}{P}$$

$$W_{avg} = W_m + S_m = \frac{N - N_{avg}}{N_{avg} \cdot L}$$

This simply states that the average slave waiting time increases as more slaves become idle $\{N-N_{avg}\}$ while waiting for service from masters.

The same conclusion can be reached by examining the bus utilization factor, which is the fraction of the time that a particular processor will make use of the data bus during an instruction cycle.

This master/slave approach reinforces the need for some sort of control to supervise the allocation of memory to processors and the allocation of time slots to processors for execution of tasks. If this time slot (or bus utilization factor) is reduced then a significant increase in system throughput is achieved.

2.5 Supervisor Control

The key to the success of a master/slave multiprocessor system lies in the system management methodolgy selected. Software has been shown to be less reliable than hardware [KOP 81] and a large program can never really be proved correct. Rodd [ROD 76] has shown in his investigation that the implementation of an operating system kernel in hardware has much promise. The use of bit-slice technology offers the designer a chance to design the architecture of the master processor to suit his needs. For this thesis, therefore, a bit-slice master controller was designed to contain several features of the kernel of an operating system. This will be discussed at a later stage.

2.6 An Overview of Ramrod

Based on the information previously discussed, Ramrod whose simplified diagram is shown in figure 2.5, was designed with the following features:

1. Distributed Operating System

2. TDM shared bus

3. Master-slave processors

4. Tightly-coupled slave processors

5. Common - Shared memory, with no distinction between global and local memory.

6. Intelligent Input/Output Control

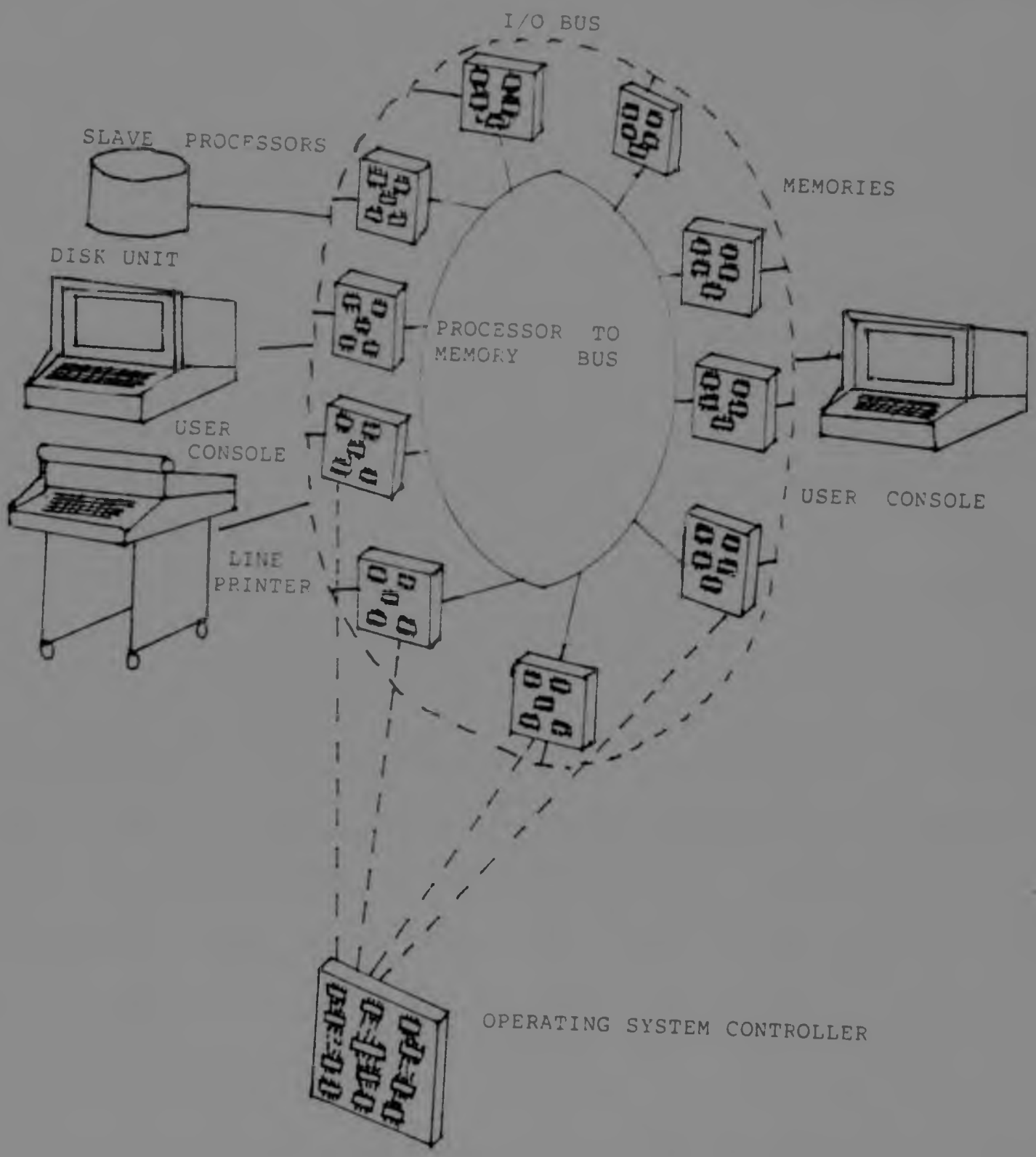


FIGURE 2.5 SYSTEM DIAGRAM

2.7 Conclusion

The multicomputer system with distributed control is probably the only architectural form that has the potential to satisfy all major architectural goals such as cost effectiveness, modular extensibility, fault-tolerance, and decomposition of software complexibility [GIL BEHR]. The multiprocessor is an interconection of uniprocessors and it is this interconnection scheme which forms the basis of this thesis.

Nemrod has been built to prototype level. It is controlled by a bit-slice processor which has the function of a system supervisor. There are presently 5 slave processors each of which are designed around the 8086 microprocessor. The memory modules, which initially consist of 256 byte segments, are interfaced to the processors via a time-shared common bus, which is implemented in Emitter Coupled Logic (ECL).

The I/O section has not been implemented in the prototype but is the subject of a parallel developement (appendix H). The following chapter describes the hardware structure whereas chapter 4 discusses the software in more detail.

CHAPTER 3

A REAL-TIME OPERATING SYSTEM FOR RAMROD

"Then did I see in the whole work of GOD, that a man is not able to find out the work that is done under the sun, inasmuch as though a man were able to toil to seek for it he would not find it, and even if he were wise to think to know it, he would yet not be able to find it" [Eccl viii 17].

This chapter discusses aspects of real-time operating systems which must be considered when providing the supervisory control required by Ramrod. In order to meet the objectives of speed and reliability it was decided to place as much as possible of the operating system in hardware rather than in software. Finally in order to meet the criterion of reliability, it was decided to distribute the operating system as far as possible throughout Ramrod.

3.1 The Role of an Operating System

In general an operating system has the prime function of transforming raw hardware into a system more amenable to its users! In addition, it should make the best possible use of available hardware so as to be generally more cost-effecive.

An operating system has to be able to:

1. Provide maximum system reliability with a minimum of operator interventio .

2. Exclude the user from details of implementation i.e make the system appear to the user as simple as possible.

3. Give the impression that the user has the sole use of the computer.

In general then a real-time, multi-user operating system should be able to:

1. Perform input/output either to a peripheral and/or to a user

2. Dispatch tasks to processors according to some predefined algorithm

3. Perform multitasking, i.e. allow many tasks to be executed concurrently

4. Supervise communications between tasks and/or the operating system

5. Recognize and service interrupts

3.2 Multiprocessor Operating Systems

In the system developed in this thesis, Ramrod, in which there are many slave processors available for executing tasks, the operating system has an extra function to perform in scheduling the slave processors for execution of tasks. Once a task is dispatched to a free slave processor then the slave processor still has to be initiated. This is a true multitasking or multiprocessing environment and tasks can be said to be running concurrently. It must be noted that concurrency in a parallel processor computer is true concurrency as processors can execute tasks absolutely simultaneously, whereas there is only 'apparent' concurrency in a uniprocessor computer (since the so-called concurrent tasks are actually being processed serially).

In order to have a near-linear increase in processing power in relation to an increase in processors, the master controller (in which the kernel of the operating system resides) should preferably have a cycle time faster than that of a single processor. This ensures the ability to control the slave processors as well as execute the kernel of the operating system.

An advantage of having some of the operating system removed from the processing environment is that the master can be faster than the slaves and thus have more effective control over the system. Secondly, a purpose-built hardware structure should be able to accommodate and execute the function of an operating system better than a general microprocessor. This is largely due to the fact that the actual functions performed by an operating system are relatively simple and require little data manipulation. The functions do, however, require to be executed as fast as possible, in order not to degrade the performance of the actual processors.

In order to have enough power to control many processors on the one hand and to contain an operating system on the other hand, bit-slice architecture (which is very fast and is microprogrammable, see app B), offers the opportunity of designing a purpose-built powerful operating system processor, as mentioned in 2.4.

An advantage of having some of the operating system removed from the processing environment is that the master can be faster than the slaves and thus have more effective control over the system. Secondly, a purpose-built hardware structure should be able to accommodate and execute the function of an operating system better than a general microprocessor. This is largely due to the fact that the actual functions performed by an operating system are relatively simple and require little data manipulation. The functions do, however, require to be executed as fast as possible, in order not to degrade the performance of the actual processors.

In order to have enough power to control many processors on the one hand and to contain an operating system on the other hand, bit-slice architecture (which is very fast and is microprogrammable, see app B), offers the opportunity of designing a purpose-built powerful operating system processor, as mentioned in 2.4.

3.3 The use of the Operating System in Ramrod

The Ramrod operating system has certain essential functions to perform. These are summarized as follows:-

1. Each processor in the Ramrod multiprocessor structure must be allocated a task to execute, and these have to be loaded into the common memory from an external source. A segment of memory must be assigned to each task, so that each processor can execute a task independently of other processors. A task is considered to be an activity which provides a function such as Input, Output or it may be an execution of a program, or segment of a program [LIST].

2. From 1 above it may be seen that each processor in Ramrod has to be allocated a time slice in order that it may access memory on the other side of the common bus. In addition, the particular memory segment selected has to be enabled. A processor must be able to address any or all of the memory segments in order that the system can be said to contain a virtual memory. To achieve this,

therefore, some sort of intelligent control is needed

3. A list of information pertaining to the location of defined tasks in memory, processors scheduled to run tasks, and the status of tasks must be monitored so that the operating system knows the configuration of the system at any point in time. An interface to the system user is also required in order that such system information may be accessed, as well as providing an overall ability to communicate with the system and its component parts.

4. In the event of a processor failing, the task which it had been executing must be redispatched to the next available working processor.

5. Some user-defined tasks will require the ability to communicate with others and this must be supervised in order that security of information may be assured. One processor may also need to draw on the results produced by another processor.

In view of the above more than dedicated logic is needed for the total control of the system. Therefore, an intelligent master controller must be created which, in essence, contains some basic features of the kernel of an operating system, and may indeed implement these facilities in hardware (see 4.2.).

As has been pointed out in chapter 1, distribution of the hardware improves system reliability and similarly distribution of the operating system will improve software reliability. Thus it was decided to distribute the operating system as much as possible. The major effect of this is to provide for limited operation in the event of the failure of a particular section of the system.

3.4 Basic Structure of the Operating System.

It has successfully been shown that a hardware-based operating system implemented using bit slice technology can indeed work with a high degree of efficiency [ROD 76].

As was demonstrated in Rodi's work this approach has the advantage of a high operating speed and requires a relatively simple hardware structure. This simplicity aids in the debugging of the hardware/software structure. The core of an operating system is the executive or the nucleus, and it is this that will be implemented in the bit-slice master processor. The nucleus concerns itself with memory management, input/output, task dispatching, processor scheduling, inter-task communication and interrupts.

The operating system proposed for Ramrod also has the highly desirable property of being partially distributed. This increases the reliability of the system as a whole, because once a slave processor is executing a task, it needs no assistance from the master until inter-task communication is wanted or the task has terminated. If the master fails, the slave can still carry on executing until one of the above two terminating conditions occur. This feature is important in view of the strategic role of the operating system construction. Clearly as appendix J shows, from the reliability point-of-view this is a weak point and the failure of the master should not cause a total system collapse. Therefore an effort should be made to distribute the operating system

wherever possible.

Thus routines related to the function of the Input/Output module are implemented in the slave processors while the rest of the kernel of the operating system is incorporated in the master bit-slice processor.

Another important consideration in the design of the operating system is the control over memory usage. Memory management is concerned with loading tasks into memory, ensuring that there is place for the task to reside and finally removing completed tasks. This can be combined in Ramrod with task dispatching since the memory is common to all processors, and therefore a particular memory segment can be assigned to any processor. A Task Control Block (TCB) table is kept to inform the nucleus where the task resides the state of the task and to which processor it has been dispatched. Thus dynamic rescheduling is achieved by allowing another free slave processor access to the segment of memory.

Pairing a memory segment and a slave processor is accomplished by selection, by the master, of a segment of memory simultaneously with the selection of a processor for access on the TDM common bus. A modulo n counter (where n = number of processors) generates consecutive addresses for reading a fast Read/Write memory (RAM), whose output selects or deselects processors and memory segments. The master controller has the ability to rewrite this fast RAM, thus allowing any combination of processor-memory communication. This is illustrated in figure 3.1, which shows that the master controller determines which devices are allowed access onto the TDM bus.

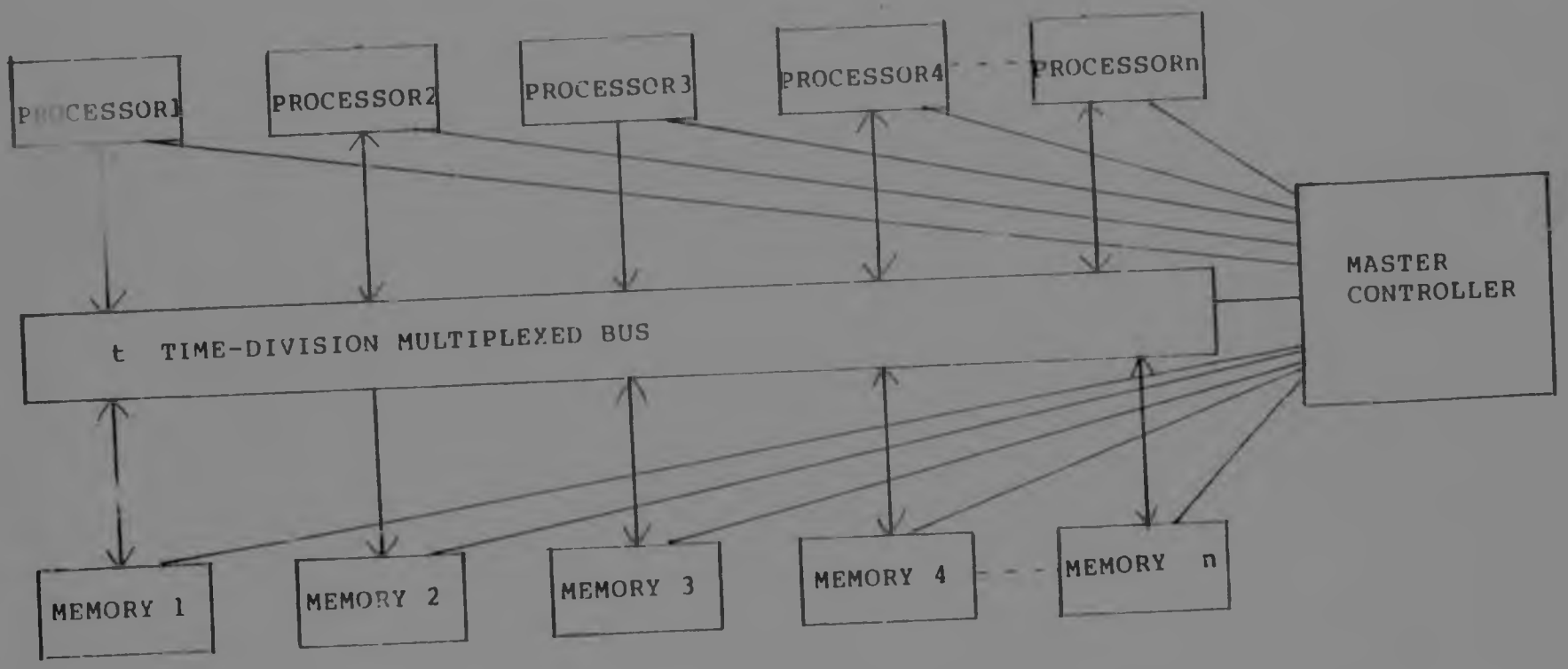


FIGURE 3.1 SLAVE PROCESSOR TO MEMORY SEGMENT PAIRING

3.5 Inter-Process Communications

In any system in which several tasks are being executed in parallel the situation will always occur when processors require to exchange data. Inter-process communication can be defined, in this context, as message passing between processes. For example, in a simple arithmetic calculation Z=(X+Y)*(X-Y), one task could do the addition, a second task the subtraction and a third task the multiplication. The first two tasks have to pass their data to the third task, in order that the multiplication can occur. Thus the third task waits until it receives messages from task 1 and task 2. Inter-process communication (IPC) has to be co-ordinated by the operating system which must know, amongst other things, who the partners to the message are, so that processes can cooperate correctly in the manipulation of data. Whilst a more detailed discussion of communications appears in Chapter 6, the discussion which follows outlines how Inter Process Communication (IPC) is presently implemented in Ramrod.

Of importance in the consideration of IPC methodology is the danger of deadlocks. This arises because resources are usually allocated to processors on the basis of their availability without any predetermined allocation algorithm. Deadlock can be explained as in

the following example: A user task is granted the printer for outputing data and then requests the card reader to read in data; Another user task is using the card reader and then requests the printer so that it can output results. If these resources can only be used by one process at a time, and neither process will release the resource it holds, then deadlock occurs. In order to prevent deadlock, (or "deadly embrace") Dijkstra proposed the semaphore[DIJ], as 'a non negative integer, which apart from initialisation of its value, can be acted upon only by the operations Wait and Signal' [LIS]. The Wait and Signal functions can be summarized as follows;

Wait(s) : when s>0, decrement s
Signal(s): increment s

Thus a resource (printer etc) can only be allocated to one process. This approach is widely used and could be implemented in Ramrod (see section 6.3).

Another more practical solution is, however, possible. If a program is partioned into tasks, which are executed sequentially, such that there is no need for any inter-task communications until a task is terminated, and inter-task communication only takes place between adjacent tasks, then deadlock can never

occur! Thus when partitioning the program, if a point in a task is reached where communication with another task is needed, this is the place where the user should partition the program (see 6.2).

It is suggested that the user, who writes the programs for his particular needs should do the partitioning of the tasks in this manner. This clearly is possible to implement automatically but it is beyond the scope of this present investigation to include software which will partition the tasks according to the above specification. In the present system this is carried out manually. Thus using data flow techniques (chapter 6) the only communication between one task and another occurs either at the beginning or at the end of the task. This highly pragmatic approach proved most useful, and suprisingly easy to implement. It is however only a partial and somewhat crude solution. As will be discussed in the next section, Ramrod has provided many other possible hardware mechanisms which may be used to implement Inter-Process Communication. Thus Ramrod is a useful testbed for evaluating a variety of proposals.

3.5.1 Communication mechanisms provided by Ramrod

Ramrod provides three mechanisms through which tasks can communicate with each other.

1. An intelligent Input/Output controller (Ethernet see appendix E), which allows any processor to be connected to any peripheral, or to any other processor.

2. A vector interrupt system to the master: A task can suspend itself once IPC is required and be woken up at a later stage. This is analogous to Hoare's communicating sequential processes (see chapter 6).

3. Tasks communicate by passing data through common memory (see earlier discussion on Dijkstra's semaphores, which can be implemented via this mechanism).

However it must be pointed out that the above are only mechanisms, and do not provide for deadlock avoidance! They do show however the power of the structure of Ramrod as an experimental tool.

3.6 User Task to Operating System Communication

Communication between any user task and the operating system is effected simply by means of a 'watchdog' timeout signal which interrupts the master controller. A task must contain instructions which continuously trigger a monostable multivibrator, which will time out if the task terminates or if a failure occurs. The interrupts of the slave processors are vectored so that the master can identify the interrupt. The master can then check whether the timeout was caused by a processor fault or task fault or if the processor has finished executing the task.

In summary this simple mechanism is extremely powerful and provides both for a termination indication, as well as the ability to detect a processor failure.

3.7 Conclusion

Implementing the operating system in hardware (by purpose designed architecture) makes the overall system reliable and flexible, because (as stated before) hardware is naturally more reliable than software. In addition there is an ability to microprogram the operating system , it is claimed that the system is flexible, as the architecture is easily

modified to suit the user's needs. The operating system functions have to be complemented by the minimal amount of software and this adds to reliabilty.

Furthermore the operating system is distributed in that Input/Output and certain local control routines are implemented in the slave processors. Thus the reliability of the system as a whole is enhanced.

CHAPTER 4

SYSTEM HARDWARE

"No man hath control over the spirit; and there is no control over the day of death; and there is no representation in that war; and wickedness will not deliver those that practised" [Eccl viii 8].

The block diagram of Ramrod was discussed in 2.6 and in this chapter the hardware of both the bit-slice master processor and the multiprocessors are outlined. A more detailed discussion of this hardware structure is to be found in appendix G.

In order for any new architecture to have economic relevance, it must be simple and efficient, and meet the needs of its potential users. In order to achieve these aims it should exhibit such features as fault-tolerance and provision of the neccessary redundancy. The multiprocessor structure of Ramrod fulfills these criterea.

4.1 System Overview

As has been discussed previously (sect. 1.5), distribution of work over veral conventional processors with common storage is one approach to increasing processing speed. However, a serious problem with common, shared-memory multiprocessor systems is that all the memory is accessible by all processors, and therefore special support is required to ensure that processors do not access the same address simultaneously, thereby corrupting each other's data [AGER 82].

Figure 4.1 shows the overall system block diagram with the Master Controller (MC) which is in charge of the system. The MC which is a bit slice hardware based real time operating system controls the data/address latches on both sides of the Time Division Multiplexed (TDM) bus. As the latches are identical the hardware can be said to be modular.

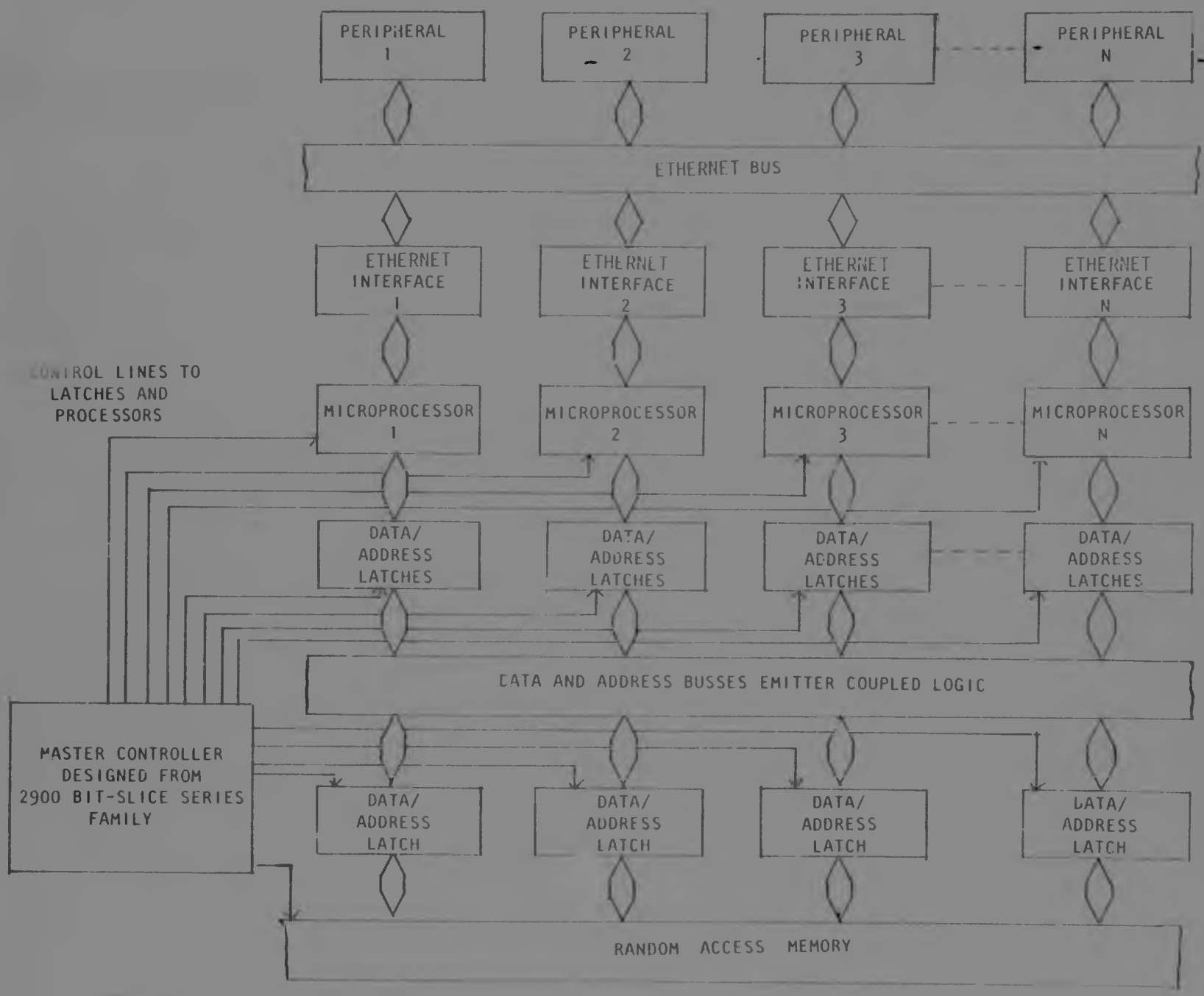


FIGURE 4.1 RAMROD BLOCK DIAGRAM

The multiprocessor architecture proposed and designed makes use of conventional microprocessors, with their relatively slow processing times. Of great significance is the fact that the cycle time of a single processor in the system is not significantly decreased by the processor-memory switch. The average cycle time for a conventional microprocessor is approximately 1 microsecond, the time being set primarily by speed of memory access. Figure 4.2 shows how all the processors communicate with the common memory by way of the TDM bus. While the memory is being accessed the bus is idle w.r.t use by the first processor and this time is available for use by the other processors. Each processor uses the bus for a very short period and if there are 50 processors then this period is 20 nanoseconds. Thus with the present system 50 microprocessors are able to communicate with each other with almost no degradation in performance of any of the microprocessors.

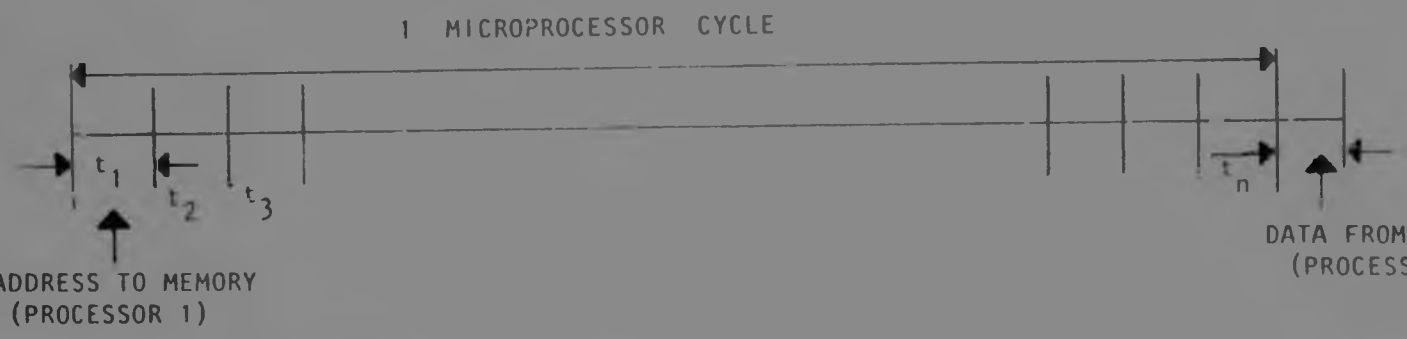


FIGURE 4.2 TIMING ON THE SHARED BUS

The cyclic operation occurs as follows: each processor deposits data and addresses into its latches and when these are given access to the bus, data are transfered into latches on the other side of the bus.

It should be noted that the data can be sent to more than one set of memory latches, thus giving the processors access to more than one memory segment simultaneously. In addition it should be noted that the concept of a distribued data base can be implemented easily on this type of computer system, as a global variable with many copies can be simultaneously updated by one processor. A distributed data base implies that each processor in the system has its own copy of the data base.

4.2 Basic Structure of Ramrod

The following section provides an overview of the various components of Ramrod. Full details of actual implementation, with the circuit diagrams, appear in Appendix G.

4.2.1 Microprocessor Module

The microprocessor slave module consists of an 8085 microprocessor together with memory and the associated support chips. It is also provided with a serial data channel to allow access during development (so that a terminal could be provided to each slave processor and hence allow direct control).

These slave processors have to be synchronised with the Time-Division Multiplexed (TDM) bus in order that there should be a minimal amount of processor idle time (as discussed in section 2.4.3). In addition, in order that the operating system can be distributed, a limited number of operating system functions must be present on each processor (so that failure of the master controller is not critical in the short term). The slave processors can therefore continue executing the tasks dispatched to them until the tasks suspend themselves, and thus the system can "gracefully degrade".

The local operating system is implemented in a resident Electronically Programmable Read Only Memory (EPROM) on the slave processor module and includes additional software to enable the slave to be self-tested.

4.2.2 TDM bus and Interface

The system designed is tolerant of processor failure but, as with the conventional common bus, it is very sensitive to bus failure. A catastrophic failure occurs if the bus fails and therefore a dual, redundant bus must be provided to minimize the possibility of system failure due to bus failure. The system can contain two sets of identical latches for each processor and memory segment. Thus, when one bus fails (which can be detected by the master processor), its associated latches are disabled and the second set of latches is enabled. A second control board (see 4.2.7.8) can achieve this switching of latches.

Most Multiprocesor systems with a common memory and bus suffer from bandwidth limitations, since the bus bandwidth will not increase even though more processors are added (as has been mentioned previously in section 2.2.1). Thus, what is needed is a state-of-the-art design, capable of high speeds of transfer, inexpensive and uncomplicated.

When choosing a logic family for the implementation of the bus and interface there are several factors to consider: i.e. noise immunity, logic flexibility, speed and some practical considerations. Obviously, for each application the factors must have a certain priority. In the case of the bus controller the highest priority is given to speed, as this determines the transfer rate across the bus. Then the priorities are: logic flexibility, practical considerations and noise immunity.

4.2.2.1 Speed -

In order to decrease the degradation in processor performance, the transfer rate of the bus must be high. Unfortunately the faster the logic, the higher the cost and the power dissipation! When considering high speed, the number of levels of gating becomes an added factor, which in turn is a function of the logic flexibility.

Gate propogation delay is perhaps the most important measure of speed. It is defined as the time taken for an output to appear from a gate after the signal has been entered at the input.

4.2.2.2 Logic Flexibility -

Reduction of the component count for a particular device is dependant on the flexibility of the logic family used. Flexibility is roughly related to the number of different outputs the integrated circuit (IC) has available. Wire-ORing, the capability of tying more than one output together also significantly reduces component count. Other factors to consider are:

1. Complementary outputs, so that inverters are unneccesary.

2. Transmission line driving capability, because the faster the signal the more closely a short line acquires the characteristics of a transmission line [MOT B].

3. Input/output interfacing, i.e interfacing to the bus and from the bus to memory.

4. Interfacing to other logic types if their levels are not Transistor-Transistor Logic (TTL) levels.

5. Multiple gates, thus reducing chip count.

4.2.2.3 Practical Considerations of Logic Choice -

Before committing a design to paper, the availability of the components has to be ascertained, and second sourcing has to be considered. Since a budget is normally to a project and subdived for the various sections, the cost factor plays a part in the selection of the component. If the logic to be used is "unusual" then the designer has to address problems such as what power supplies are required as this might neccesitate extra power supplies over and above the normal single +5 volts requirement of TTL based systems.

4.2.2.4 Noise Immunity -

As a system such as Ramrod might have to operate in an electrically noisy environment, it must have high noise immunity (it was originally conceived for use in process control). There are two types of noise immunity to be considered, i.e. internal and external. When the circuits themselves switch from one level to another, internal noise is generated, whereas external noise is caused by external devices. A good measure of immunity is the voltage difference between the two logic levels, as the greater the

difference between levels the higher the noise level must be in order to corrupt the data.

4.2.2.5 Comparisions of Logic Families -

As the highest Priority is speed, only Emitter Coupled Logic (ECL) and Advanced Schottky Transistor-Transistor Logic (AST) were considered for use in the bus system, as these are the only currently available, off-the-shelf, logic families fast enough for the application. The advantages and disadvantages of these two families are tabulated in the appendix.

Both ECL and AST have the same availability and second sourcing problems in South Africa, i.e both are difficult to obtain, and the costs are generally the same.

The major disadvantages of using ECL are the several power supplies required and the need for thoroughness in testing. However, as ECL s at least twice as fast as AST, it was chosen as the logic in which the bus and interface were to be implemented.

4.2.3 Bus Interface

The interface to the ECL bus is implemented via a bidirectional logic level translating latch, which provides a rapid means of converting microprocessor or memory TTL levels to the bus' ECL levels. In addition the latches need ECL control signals which are provided by simple one-way TTL to ECL translators.

Each microprocessor and memory module has its own set of latches,and as the latches on either side of the bus are identical, there is no need to design an extra latch module. The latches, as mentioned above, can translate in either direction and can thus be used on both sides of the bus. Its control signals, which are derived from the processor and memory boards, determine how its operating mode.

4.2.4 The Time-Division Multiplexed Bus (TDM)

The TDM bus adopted is unusual in that it is circular and is joined at the ends! The philosophy behind the structure is simple: to ensure minimum transmission time of signals on the bus the physical distance between any processor and the memory unit should be kept at a minimum (fig 4.3).

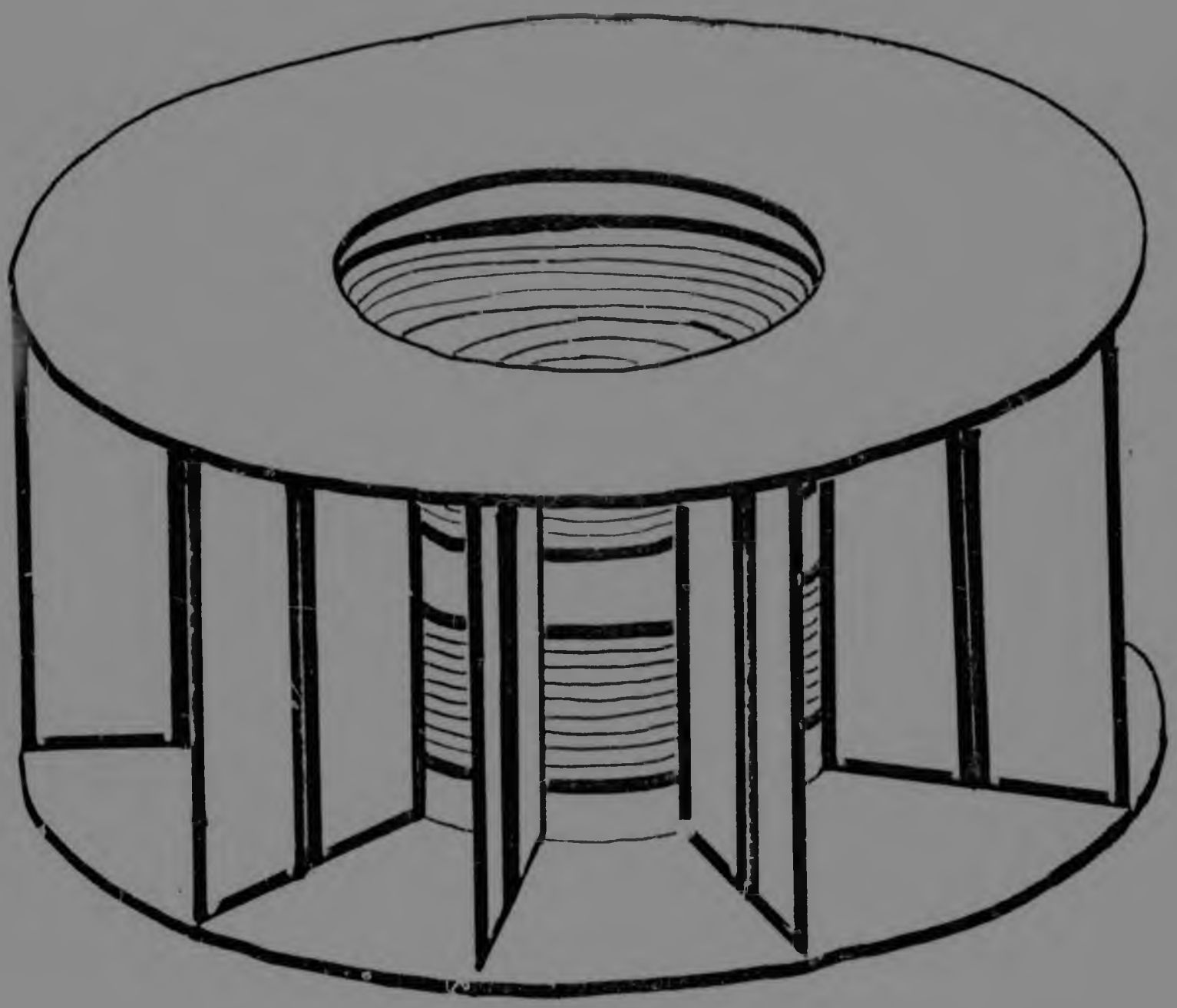


FIGURE 4.3 CIRCULAR CONSTRUCTION

The use of a circular bus for ECL has not been widely documented, although Sanderson and Zoccoli [ZOC] have designed a multiprocessor system using a circular ECL bus where they state the advantages of using such a construction, but they do not, however, go into detail. Therefore the modelling of the bus has been the subject of an additional investigation (Appendix C). This investigation has shown the princple to be viable and has revealed design parameters.

Note that there are actually two buses which the system uses:

a) The ECL TDM bus
b) A TTL bus for power and control signals

4.2.5 Memory Module

In the prototype each module contained a relatively small memory segment (256 Bytes Random Access Memory, RAM) together with the logic needed to produce the signals for reading from and writing to the latches. This was selected partly on economic grounds - a practical, full scale system clearly would have larger memory segments. The speed of this memory need not be particularly high because the slave

processors access this memory via the TDM bus and have to wait their turn for a time slice.

4.2.6 Input/Output Module

The implementation of the input/output section adopted is similar to the memory interface concept developed in that it is possible to pair a device on the I/O bus to any other device. It is advantageous to have interchangeability amongst processors for I/O functions just as for memory. Implementation must also take into account possible processor-to-processor communication as well as processor-to-peripheral communication. Thus the input/output interface must be highly flexible.

In order to implement this a highly intelligent and fast interface is needed. The only medium found to fulfil these criteria lies in the adaptation of a simple but high-speed link based on the principles of the Ethernet System. Ethernet has the highly desirable feature that no master controller is required. Any device wishing to use the bus simply 'listens' and seizes the bus when it is free. Simultaneous transmissions are ignored and retransmission takes place after a 'random' wait time (see Appendix E for more details of Ethernet).

However it must be pointed out that initially, in order to simplify testing of the system, I/O was achieved via dedicated processors. The software has an identity section which can determine whether the processor is connected to a terminal or a disk operating system, or whether it is simply a task processor. Work on the Ethernet controller is currently taking place in a related project. (see Appendix E for details).

4.2.7 Bit-Slice Master Controller

The need was established (sect 1.4) for a hardware based operating system with the following facilities:

1. An interface to the multiprocessor structure to schedule processors

2. A sophisticated interrupt hierarchy. Interrupts may come from the processors, after failure or task termination or from real-time clocks. Any hardware- implemented operating system must be able to deal with these interrupts and respond accordingly

However it must be pointed out that initially, in order to simplify testing of the system, I/O was achieved via dedicated processors. The software has an identity section which can determine whether the processor is connected to a terminal or a disk operating system, or whether it is simply a task processor. Work on the Ethernet controller is currently taking place in a related project. (see Appendix E for details).

4.2.7 Bit-Slice Master Controller

The need was established (sect 1.4) for a hardware based operating system with the following facilities:

1. An interface to the multiprocessor structure to schedule processors

2. A sophisticated interrupt hierarchy. Interrupts may come from the processors, after failure or task termination or from real-time clocks. Any hardware- implemented operating system must be able to deal with these interrupts and respond accordingly

3. The processor must have available a limited amount of high-speed storage in which it can hold Task Control Blocks, pointers, stacks and constants.

In order to implement these elements in hardware, there are two important factors which must be kept in mind: speed and flexibility. Flexibility is desirable so that the functions can be as universal as possible and so that the processor can be expanded if needed.

4.2.7.1 Bit-Slice Architecture -

It has been shown [ROD 76] that bit-slice architecture offers the best features for implementing the system discussed above. The designer has almost complete control over the architecture of the processor required, and in addition the bit-slice processor is microprogrammable, a feature that Prof. M.V. Wilkes [WILK] has said makes it superior to other architectural techniques. It offers a greater degree of flexibility in specifying a computer's instruction repertoire, while also resulting in considerable simplification in the logic.

Bit-slice microprocessors are capable of high-speed operation since they are based on bipolar technology, often resulting in cycle times of less than 100 nanoseconds.

Currently the following bit-slice microprocessor families are widely used and relatively freely available:

1. The Intel 3000 series

2. The Motorola 10800 ECL series

3. The Advanced Micro Devices 2900 Low Power Schottky (AMD) series

From a user's point of view the differences are few but the main designer's criteria are local availability and developement tools. The AMD series was chosen because of the local support and second sourcing and primarily because a cheap emulation tool was available in the form of an extension to the Motorola EXORciser (see Appendix F).

Turning to the actual design, the bit-slice processor can be subdivided into two parts: the Computer Control Unit (CCU) and the Central Processor Unit (CPU). Bit-slice architecture is essentially the same as that of a normal processor with one basic difference; each functional unit has only (for instance in the 2900 series) a 4 bit wide word and to make an 8 bit word two units need to be inter-connected. Fundamental to any microprocessor based system is the determination of the micro-instruction. Bit-slice processors have the advantage that their micro-instruction set is completely user definable and, before dealing with the actual architecture, the structure of the microinstruction must be discussed.

4.2.7.2 Format of the Microinstruction -

The principle decision which has to be made by the designer of a microprogrammable logic system is the format of the microinstruction [AGU 76]. In determining its format the designer has to bear in mind the facilities required and the external control desired for this format.

There are generally two classifications of microinstructions: Vertical or Horizontal. A horizontal microinstruction will control the operation of many resources in parallel, and can be unlimited in width but in practice is normally up to 64 bits wide. (In actual fact this is often determined by practical issues - such as the maximum size which a developement facility can support). In contrast, a vertical microinstruction is similar to a normal machine code instruction and affects only a single primitive operation. After reviewing the requirements of the bit-slice processor it becomes apparent that the chosen format must be horizontal, in order to achieve the parallelism required.

The designation of fields within the chosen microinstruction is shown in figure 4.4. This means that in one horizontal microinstruction the following operations may be specified;

1. Control of Central Processor Array (CPA)

2. Control of next address generation

3. Control of status of flags from the CPA

4. Control of input/output functions, including local memory control board, interrupt unit and microprocessors

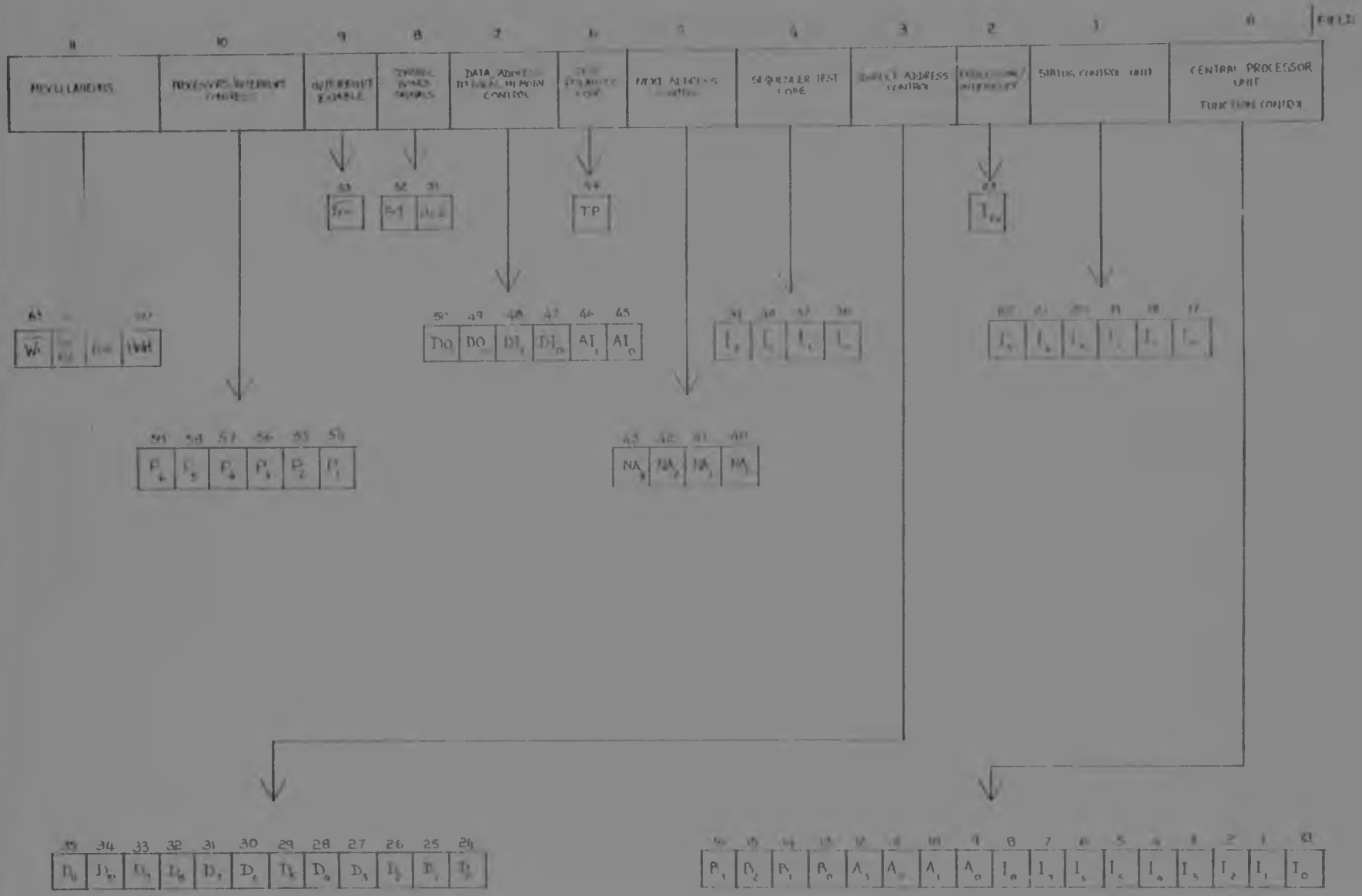


FIGURE 4.1 MICROINSTRUCTION FORMAT

4.2.7.3 The Computer Control Unit (CCU) -

The major function of the CCU is the sequencing of instructions, i.e the determination of the order in which instructions are to be fetched from the microinstruction store. Generally the microprogram sequencer contains:

1. A microprogram counter register which will increment after each clock cycle, thereby selecting sequential addresses.

2. A condition code multiplexer whereby the status of flags or other bits can be tested for conditional branching.

3. A multiplexer which can select between the counter register and a directly specified address.

The AMD sequencers and next address unit used allow the addressing of 2**12 = 4096 locations with 2**4=16 next address instructions for the control of conditional branching instructions. The actual address space is organised into a 1-dimensional array, 1024 by 64 bits wide. Each microinstruction supplies 4 bits for the next address control and 12

bits for the actual address. The scheme has an advantage in that it allows the user to write his instructions in a sequential fashion. In addition, most other conventional programming techniques can be used (for example subrouting where return addresses are automatically stacked and unstacked).

4.2.7.4 The Central Processor Array (CPA) -

The CPA of the bit-slice microcontroller is similar in functional operation to that of the Arithmetic and Logic Unit (ALU) of a conventional von-Neumann type structure. The CPA can execute the following operations:

1. ALU functions

2. Address and route data to and from local memory

3. Route data to and from I/O interface to the control board

4. Mask the interrupt control unit

5. Provide status bits to be routed to the CCU

The AMD CPA contains 16 general-purpose registers to hold stack pointers, memory address pointers and system constants. A fast look ahead carry unit is provided to make fast arithmetic computations possible. A status and shift control unit is included to control status and other functions usually associated with an ALU.

The logical operation of the ALU is determined by a 1?-bit control code and 6 bits determine the operation of the status contol unit.

4.2.7.5 Input/Output Control Unit -

The input/output control has the following functions:

1. Local memory read/write

2. Control board read/write

3. Microprocessor memory read/write

4. Microprocessor hold and reset

5. Masking of interrupts

The programming of the control board (4.2.7.8) is achieved by the I/O control unit. While this board is being programmed its outputs are inhibited to prevent unwanted processor-memory combinations from taking place.

Upon receipt of an interrupt the input/output unit can either hold the particular processor or reset it to begin executing a task. In order to interrogate the memory of any microprocessor the MC behaves like a slave processor and simply reads the memory.

4.2.7.6 Interrupt Unit -

Ramrod accepts interrupts from each microprocessor and can interrogate its memory to find out the type of interrupt. Only 5 levels of interrupt are used, although the number is theoretically expansible to any number of levels.

A slave processor generates an interrupt request signal which instructs the CCU to jump to the interrupt service routine, where the identity of the interrupting processor is determined. This is achieved by the interrupt controller which supplies the sequencer with an address corresponding to the

The programming of the control board (4.2.7.8) is achieved by the I/O control unit. While this board is being programmed its outputs are inhibited to prevent unwanted processor-memory combinations from taking place.

Upon receipt of an interrupt the input/output unit can either hold the particular processor or reset it to begin executing a task. In order to interrogate the memory of any microprocessor the MC behaves like a slave processor and simply reads the memory.

4.2.7.6 Interrupt Unit -

Ramrod accepts interrupts from each microprocessor and can interrogate its memory to find out the type of interrupt. Only 5 levels of interrupt are used, although the number is theoretically expansible to any number of levels.

A slave processor generates an interrupt request signal which instructs the CCU to jump to the interrupt service routine, where the identity of the interrupting processor is determined. This is achieved by the interrupt controller which supplies the sequencer with an address corresponding to the

interrupt level. This is normally known as a vectored interrupt.

4.2.7.7 Local Memory -

An operating system needs storage for tables, task blocks etc. The registers in the CPA are insufficient, and in addition scratch pad use is also neccessary, so ordinary Metal Oxide Silicon (MOS) memory is made available for this purpose. This is similar to RAM in a simple microprocessor system.

4.2.7.8 Control Board -

The control board consists of two identical sections of very fast RAM which are used to enable the microprocessor and memory modules respectively. The Master Controller can only write the enabling signals into this memory and the actual reading of the memory is accomplished by a modulo n counter where n is the number of processors in the parallel array. The data which is read from this RAM provide the enabling signals for the latches which allow processor/memory communication.

4.2.7.9 Control Store -

A key factor in the design of a bit-slice microprocessor is the cycle time of the processor. Bit-slice timing can be calculated from the worst time taken for data to traverse the data path. The access time of the control store has a direct influence on this cycle time.

The data path begins with the instruction being fetched from memory and being presented to the pipeline register. From there the individual bits are available for control of the relevant parts of the system. While the rest of the system is operating on this instruction, the sequencer generates the next address, which is supplied to the control store, the store is accessed and the instruction is ready for entry into the pipeline register.

Thus the read time of the control store is included in the cycle time, which is reduced by the method of pipelining outlined above. However there is still a need for a fast-access memory.

Bit-slice processors are usually designed with a decoding PROM which accepts macroinstructions, or normal instructions of processors, and calls subroutines of microinstructions to implement the macroinstruction. If this PROM is dispensed with,then the user can write his program on the microinstruction level, thus improving speed.

A disadvantage is that the user has to write the full 64 bits irrespective of how many bits he needs. The inner workings of the processor are also not transparent to the user. Figure 4.5 shows the general structure of the bit-slice processor discussed.

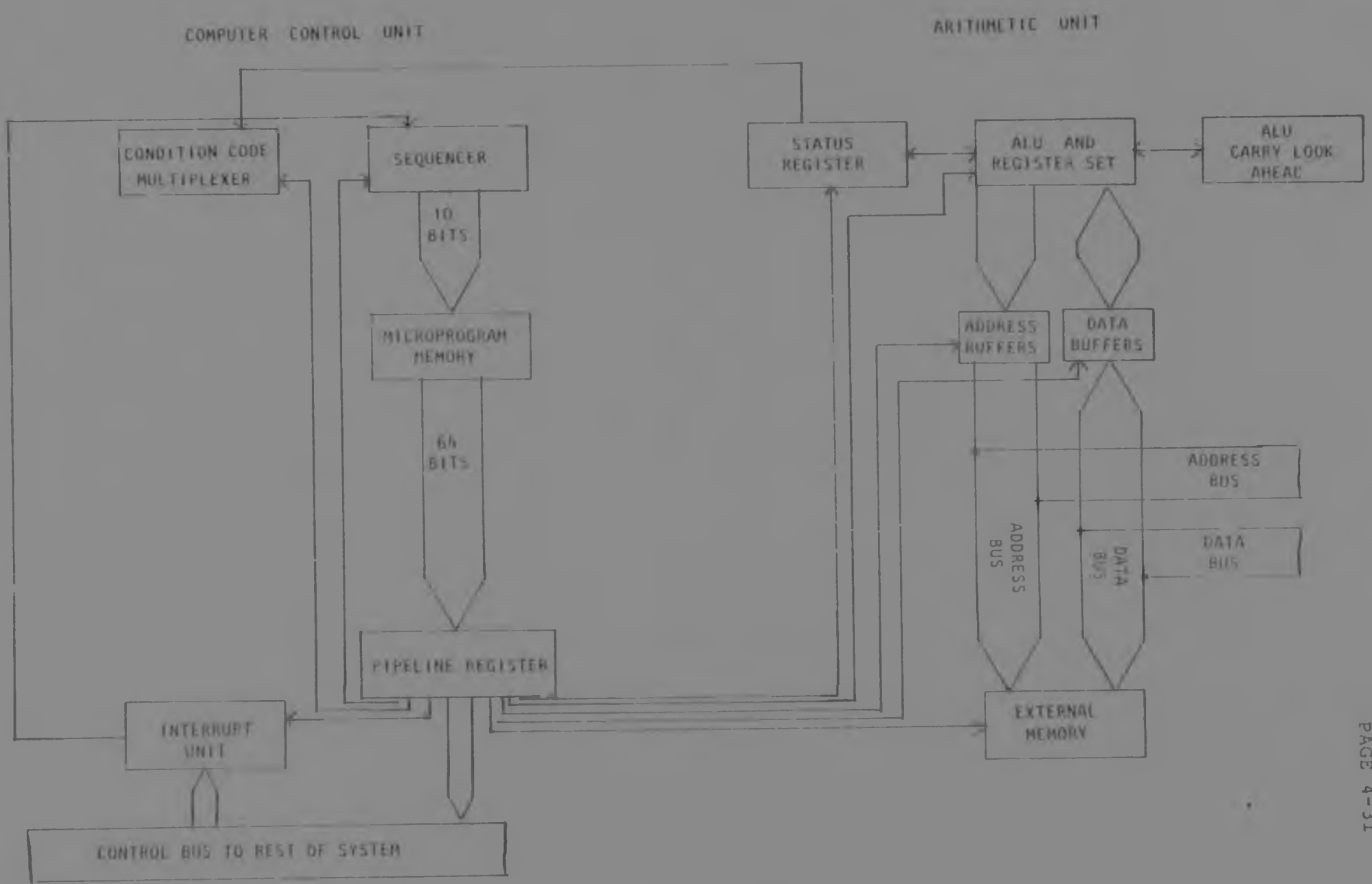


FIGURE 4.5 MASTER CONTROLLER

4.3 Physical Construction

The physical construction of Ramrod's bus structure is shown in figure 4.6. The latch boards plug directly on the buses so as to ensure that the ECL signals are generated as close as possible to the bus. The TTL control signals and power lines go through the latch board to the board that is plugged 'piggy back' fashion on to it. This 'piggy back' board can either be a memory or a processor board and it then logically determines the mode in which the latch board is to function.

The bus itself was implemented using double sided "scotch-flex" cable with one complete side grounded. Edge connectors were connected directly onto this cable.

The control board which supplies the TTL control signals plugs directly into the TTL bus and is driven by the MC via a set of cables.

The master controller is located external to the bus structure and, in practice, was located within an expansion chassis associated with the EXORciser developement system.

The slave processors, memory boards and latch modules are cooled by a fan which is mounted on top of the bus structure. ECL requires power supplies different from that of TTL and thus there are five supplies (+5, -12, +12, -5.2, -2 volts) connected and sensed at the top of the circular construction. These supplies are in addition to those of the bit-slice master controller as power requirements for the circular construction are high and if the supplies were not duplicated then there would be a significant power drop to the circular construction. Figure 4.6 shows a complete view of Ramrod.



FIGURE 4.16 VIEW OF LAMBDA

4.4 Conclusion

Ramrod has a simple architecture, uses currently available technology and is a cheaper alternative to a 'supercomputer'. The use of identical processor and memory elements results in a high degree of fault-tolerance. One unique feature is that no distinction is made between local and global memory.

Up to now the hardware structure has been developed and the following chapter describes the basis of the software structure of Ramrod.

CHAPTER 5

IMPLEMENTATION OF THE OPERATING SYSTEM

"For everything there is a season; and a proper time for every pursuit under the heavens. There is a time to be born and a time to die; a time to plant and a time to pluck up what hath been planted; a time to kill and a time to heal; a time a time to break down and a time to build up; a time to weep and a time to laugh" [Eccl iii 1,2,3,4].

As has been stated before, the operating system of Ramrod has been distributed amongst the various processors in the system in order to increase the reliability of the computer as a whole. The kernel of the operating system is implemented in the bit-slice master processor while most of the routines which control input/output and local processing are resident in EPROMs on the slave processor boards. Additional details of the operating system software are to be found in appendix I and only high level functional description are discussed in this chapter. Actual listings of the software can be obtained from the Dept. of Electrical Engineering at the University of the Witwatersrand.

5.1 Operating System Kernel

The operating system has been simplified as far as possible in order to implement only the essential functions which are required to evaluate Ramrod. It must be pointed out that additional features still have to be implemented to provide a full, commercial system.

The operating system kernel, as currently implemented in the master controller contains the following functions:

1. Dispatcher

A list is kept of the status of the tasks in the system (Task Control Blocks TCB). The dispatcher has the function of scanning the list of TCB's and when a tasks is waiting to be executed the dispatcher looks for an available processor on the processor status list.

2. Scheduler

Once a task has been dispatched to a processor, the processor has to be initiated so as to run. The scheduler has therefore the prime function of enabling processors. The 'round robin' scheme of scheduling is actually implemented in the hardware (see 4.2.7.8) whereby processors are allowed access to the bus, and hence to the common memory, in turn (i.e. time slicing the bus).

3. Memory Manager

This module keeps a list of the memory segments showing which are free or which are occupied. Once a task has been executed its memory segment joins the 'free' list. When a task is loaded this module is consulted in order to find a 'free' segment.

4. Interrupt Handler

The interrupt handler determines the source of an interrupt and proceeds to service the interrupt after saving the volatile environment of the interrupted routine. When the interrupt has been serviced execution of the interrupted routine is resumed.

5. Input/Output Module

It is the function of the input/output module to initiate an I/O operation, on request. Tasks are loaded from an external source or can be entered by the user from a console which is connected to one of the slave processors. This module is an interface between the nucleus of the operating system and the routines which are resident in the slave processors.

It should be noted that in the prototype system, tasks were resident on the disks of the associated EXORciser development system. The operating system obtained tasks from this system and loaded them into Ramrod's

memory as required. This technique obviated the need for a dedicated disk controller and disk.

5.2 Local Operating System

The component of the operating system contained in each slave processor has the following functions:

1. It can act as an extension to the input/output module of the operating system.

2. It can function autonomously in order to enable self testing of the slave processor

3. It can directly control the processing activities of the slave processor and will only execute user tasks as requested by the master controller

The microcomputer determines, on power up, what type of function it is to perform. This is achieved by the processor which writes its identity into a location of common memory and if the processor is reset then it can determine its mode of operation by reading this location. The determination of this

mode, in the final system, is automatic but during development this was basically determined by transmitting data through a Universal Synchronous Asynchronous Receiver Transmitter (USART), and reading immediately the data on the input. This feature enabled direct control of each slave processor during system testing. Once the identity of the processor is determined its mode can only be changed by a power down sequence or by the master controller which can reprogram the appropriate memory location.

5.2.1 Input/Output Extension to the Operating System

In this mode the slave processor behaves as an intelligent terminal, and can be connected to a user console or to a host computer. As mentioned before, this allows a user direct access to each slave processor - a most valuable aid during development. For example, programs which are to be run by a slave and which have been developed on, say, a development system can be loaded via this routine into Ramrod's common memory. In addition the user can view the system on the console. This gives the user a way of getting his programs into Ramrod's memory without direct control from the master controller- again a useful aid during development.

mode, in the final system, is automatic but during development this was basically determined by transmitting data through a Universal Synchronous Asynchronous Receiver Transmitter (USART), and reading immediately the data on the input. This feature enabled direct control of each slave processor during system testing. Once the identity of the processor is determined its mode can only be changed by a power down sequence or by the master controller which can reprogram the appropriate memory location.

5.2.1 Input/Output Extension to the Operating System

In this mode the slave processor behaves as an intelligent terminal, and can be connected to a user console or to a host computer. As mentioned before, this allows a user direct access to each slave processor - a most valuable aid during development. For example, programs which are to be run by a slave and which have been developed on, say, a development system can be loaded via this routine into Ramrod's common memory. In addition the user can view the system on the console. This gives the user a way of getting his programs into Ramrod's memory without direct control from the master controller- again a useful aid during development.

5.2.2 Self-Testing Routines

In order to test the microcomputer initially the following routines are included in the slave operating system;

1. Identify, on power up, the function the processor is to perform (i.e. it can be a slave processor executing tasks as set by the master controller, a processor which communicates with the user via a VDU, or a processor which can load tasks from an external source e.g. a disk operating system).

2. Substitute or update any memory in the slave processor's address space

3. Display contents of this memory on screen

4. Insert code into any of this memory

5. Execute program inserted by user

5.2.3 Slave Task Processing

In this mode the slave processors execute tasks at a specific location in the common memory address space. This location is in the common memory area and therefore tasks which have been loaded via the master controller I/O module and which have been dispatched to this particular processor are executed after a request by the master processor. On completion the master is notified by means of the mechanism described in 3.6.

5.3 Conclusion

Graceful degradation of the system is assured because if the master fails the slave processors can continue functioning until their tasks are completed.

As can be seen in appendix J, the hardware reliability of Ramrod depends on duplication of the master and the TDM common bus, whereas the software reliability is greatly enhanced by distribution of the operating system.

In summary, the operating system is distributed and contains the following:

1. Scheduler

2. Dispatcher

3. Memory Manager

4. Input/Output Manager

5. Interrupt Handler

In addition a list of information pertaining to the status of tasks, processors and memories is maintained by the master controller.

The distributed operating system which has been described is essentially simple and has proved to be most effective. It appears to be both an effective tool and a successful combination of hardware, firmware and software.

CHAPTER 6

APPLICATION SOFTWARE STRUCTURE

"For all this did I reflect over in my heart and to explain all this, that the righteous, and the wise, and their services are in the hand of GOD; that man knoweth neither love nor hatred; it is all ordained before them" [Eccl ix 1].

Whilst this thesis has set out to produce an operational system and has concentrated on the fundamental design issues, it is important to give some attention to methods which may be used to construct applications software.

Therefore this chapter discusses an appropriate method, and the techniques discussed are utilised in a relatively simple example which will form the basis for the practical evaluation of Ramrod.

It is common knowledge that parallel processing can be greatly enhanced by using techniques adopted from data flow languages. Computations represented by cyclic data flow graphs can be automatically unfolded to expose all parallelism to the underlying hardware [AGER 82].

The discussion that follows presents a pragmatic introduction to such techniques and shows how they may be implemented in production.

6.1 Data Flow Approach

Data flow machines attempt to provide concurrency in operation in order to achieve high speed of computation. Most high-level, parallel languages allow the computer architecture to be visible to the programmer in order to achieve parallelism. This is unnatural as the language then closely reflects the behaviour of the computer rather than the manner in which the programmer normally thinks [ACR]. The data flow language approach on the other hand, directly reflects the programmer's thoughts whilst making the computer's architecture transparent.

A data flow language is defined as a "language based entirely upon the notion of data flowing from one function entity to another" [DAV 81]. This data flow concept has the advantage of allowing the data flow language program to be represented graphically.

Typically a data flow language is extremely modular as sub-programs can be understood entirely on their own. These sub-programs, or modules, have no side effects, such as altering another module's variables. Thus when these programs are represented graphically, the modules that look independent can be executed independently, and modules can therefore run concurrently [DAV 82].

The data flow machine, which is a direct image of the language it supports, is in contrast to the sequential, one-instruction-at-a-time von Neumann computer model, and it is based on the following two principles:

> "Asynchrony. All operations executed when and only when required operands are available.
> Functionality. All operations are functional, i.e. there are no side effects". [GAJ]

Asynchrony denotes an execution mechanism in which data values pass through nodes in data flow graphs as tokens, and an operation is initiated whenever all input tokens are present at a node in the graph. Functionality implies that any two enabled operations can be executed in any order and concurrently.

Figure 6.1 shows how Z=(X+Y)*(X-Y) is graphically represented and therefore computed. The functions +, -, * are called actors and they reside at a node and nodes are connected by arcs. Data flows on arcs from one node to another in a stream of discrete tokens. Tokens are considered carriers of data objects [DAV]. It must be noted that an actor, or operator, cannot be initiated before all of its tokens are available (see chapter 7) (see Figure 6.1).

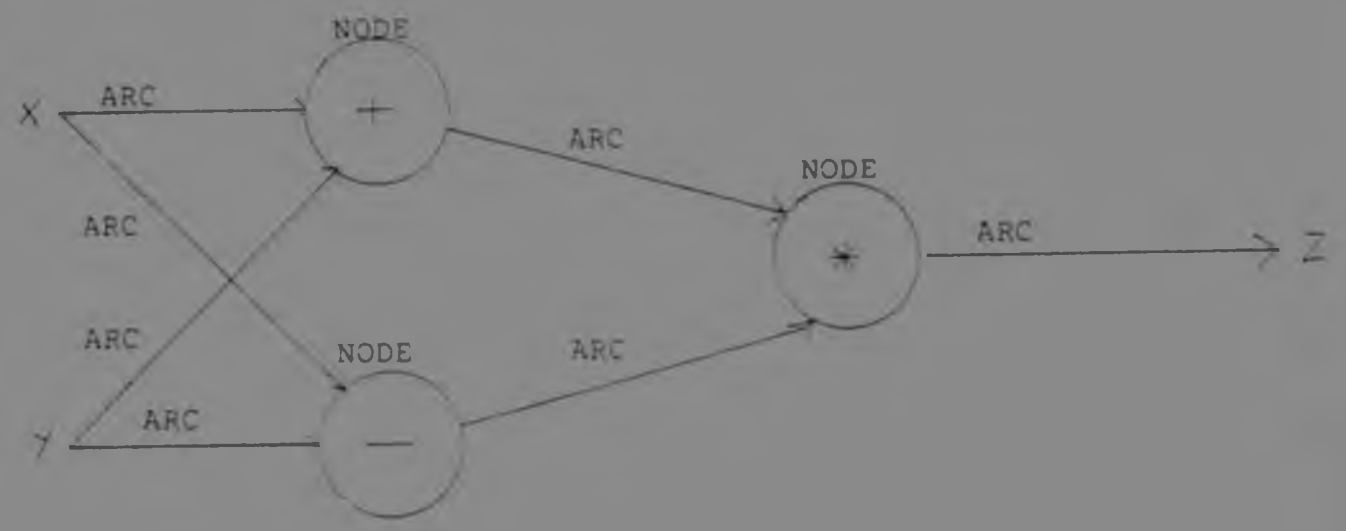


FIGURE 6.1 DATA FLOW INSTRUCTIONS

The data flow computer is designed to recognise which of the instructions are enabled. All such instructions are dispatched to execution units as soon as they are available.

The data flow concept can be extrapolated to conventional von Neumann structures by having the processing elements operate simultaneously on tasks rather than on instructions. If a more global outlook is taken, it can be seen that the task can be defined in a similar way to a data flow instruction such that it is enabled if all input conditions are met, and it is suspended when an output condition occurs.

6.2 Task Definition

The actual mechanism of automatically decomposing a program into tasks which will fit into the above category is beyond the scope of this investigation. The program must be decomposed before entry into the system. One solution is to write the programs in a data flow functional language, which has inherent properties for parallel processing (see 7.3).

Using the data flow concept, a task in this multiprocessor environment is defined as the smallest functional unit of software, which requires inputs for execution to begin and which only terminates when an output condition occurs. Therefore a task is autonomous and must run to completion before any communication with another task. This concept, is of course, very interesting as it reflects one of the original proposals discussed in section 3.5 to avoid deadlock.

Inter-task communication is thus kept to a minimum and each task has a single indivisible function. Thus the instruction in figure 6.1 is a task which accepts two inputs, X and Y, and produces an output Z.

However, it must be remembered (see section 3.5) that the above approach to Inter-Process Communication (IPC) is relatively limited, and other mechanisms should be investigated. Ramrod is an excellent vehicle for experimenting with these ideas. The next section discusses various possible selected IPC mechanisms, and it is shown how they may be implemented in Ramrod.

6.3 Inter-Task Communication

A distributed multiprocessor computer system needs a sophisticated communication medium to provide for the necessary inter-task communication. Reliability, redundancy and modularity (as mentioned in 1.2.1) are requirements for the implementation of such a medium. This communication medium is the key to flexible implementation of redundancy and expansibility [MAC].

The interaction between tasks arise when two concurrent (truly concurrent in the multiprocessor system and pseudo concurrent in the uniprocessor environment) tasks need to exchange data.

The concurrent tasks have access to common memory variables which represent the state of physical resources, and which are used to communicate data between cooperating tasks. In general the common variables can represent shared objects called resources, and in order to share resources the concurrent tasks need to be synchronised. Synchronisation is defined as an ordering of operations in time and in the multitasking environment this infers that "operations A and B must never be executed at the same time ", i.e mutual

exclusion [BRI 73]. A more detailed discussion of Inter-Process Communication primitives has been undertaken by Macleod [MAC].

The traditional ways of handling the Inter Process Communications are:

1. Semaphores

   A semaphore is a synchronising variable (flag) which informs a task whether the resource it wishes to share is available or unavailable [DIJ].

2. Critical Regions

   A concurrent task can only access common variables within a critical region. The task can only enter a critical region within a finite time, and only one task at a time can be inside a critical region. The task can remain in the critical region for a finite time only [BRI 73].

3. Communicating Sequential Processes [HOA 78]

Input/Output are basic primitives of programming. Parallel processing using communicating Sequential Processes (CSP) is a fundamental program structuring method. This Communication is considered as being synchronised input/output.

A process communicates with another process by naming it as its destination for output, while at the same time the second process names the first as a source for its input. When both processes are ready to transfer data the value to be output is copied from the source to the destination. A disadvantage of this close synchronisation scheme is that if one of the processes finishes before the other there will be a certain amount of idle time by the processor concerned, and there is also a limit on the amount of parallelism achieved.

Verifying programs in a uniprocessor environment is difficult enough, and Hoare therefore states that there is no method for verification of programs in a multiprocessor environment.

4. ADA [US DOD]

One of the most exciting developements in real time languages is the ADA language, which is a project of the United States Department of Defense. ADA is similar to CSP in that it has a low level construct for the synchronisation of parallel tasks. ADA incorporates the concept of a rendezvous, in which two processes communicate with each other at a specific (real) time, for interprocess communication.

5. Primitives for Distributed Computing[LISK]

An advantage of a distributed organisation is reduced contention for a single CPU, but this is replaced by contention for the communication medium. Other advantages are speed of response from the CPU's, better

reliability, higher capability and expansibility.

The basic construct of this IPC method is called a guardian which consists of objects and processes. An object contains data(integers etc.) and a process is an execution of a sequential program. Communication by processes in different guardians is by means of message passing. The guardian exists entirely at a single node of a distributed system. Once a message has been sent, the sending process can proceed. Receiving messages are associated with a timeout which is necessary because an expected response may not arrive due to errors or failures.

Ports, which have global names, allow queueing of messages as they provide some buffer space. If this buffer space is full the message is lost. The port is a unidirectional gateway into a guardian and is described by the type of messages that can be sent to it.

6. MARS [KOP 82]

In the MARS project IPC differs for state messages and event information. An event is a happening at a point in time, whereas state information deals with attribute values of objects which are are only valid for a certain period of time. However, event and state information are related as a change of state is an event.

An event message is queued at the receiver and can only be removed by that receiver when it is read. A state message is valid for a specified period of time and can be read by several tasks many times. The IPC mechanism is, on a high-level, a broadcast medium with a group addressing capability (Ethernet?).

The above mechanisms have been shown to be viable and the author does not wish to debate their merits. However, it is not clear how to determine which method is most suitable for a particular case of Inter Process Communication (IPC).

Ramrod, in this thesis, does not set out to solve the problem of choosing an IPC mechanism but rather provides a vehicle for testing them. All the above methods can be implemented in Ramrod as there are 3 ways to support IPC (see 3.5.1).

1. Via common memory. As a slave processor has access to any other slave processor's memory, all of the above IPC methods are able to be implemented in Ramrod

2. Via Input/Output. As the I/O interface is intelligent one processor can address another processor using unique names for each processor (CSP).

3. Via communication through the master using interrupts. The master can interrogate a slave processor and determine what it wants and most of the methods listed above can be implemented.

6.4 Conclusion

This chapter has attempted to provide a mechanism which may be adopted in order to produce application software for Ramrod. It is suggested that Data Flow techniques seem to be appropriate and in the next chapter a simple program is developed on this basis. In addition this chapter has looked at Inter Process Communication mechanisms and it has been shown that Ramrod is capable of implementing all of these - thus enhancing the value of the system as an experimental tool.

CHAPTER 7

EVALUATION OF SYSTEM

"For who knoweth what is good for man in this life, the number of the days of his vain life that he should spend them as a shadow. For who can tell a man what will be after him under the sun" [Eccl vi 12].

It has been claimed that Ramrod is more efficient than a conventional uniprocessor, but there is a difficulty in proving this. Efficiency is usually defined as the ratio between useful work performed and the total work performed. In this project, however, efficiency is evaluated on a comparative basis between Ramrod and a uniprocessor computer of similar power to one of the slave processors. Evaluation techniques (outlined by Rodd [ROD 76]) can really be only applied to one system and cannot form the basis of comparision between two fundamentally different types of computer systems. Ramrod's architecture is similar to that of an array processor and therefore it should be used for vector processing in order to utilise it as efficiently as possible. Therefore, when evaluating Ramrod this point must be kept in mind and thus merely obtaining a run time for Ramrod and a uni[illegible] would be misleading.

The problem is analogous to calculating the reliability of Ramrod, since conventional reliability theoery is really of little significance in a fault-tolerant system (appendix J). Therefore it was decided to limit the evaluation of Ramrod to using it as a simple vector processor operating on an array of integers, while allowing a uniprocessor to do the same operation on the array and then comparing the respective performance. This comparison is naturally not totally valid but, more than anything else, it does illustrate the vital factor that the system developed has much merit and provides an indication as to how it can be used.

7.1 Practical Limitations

As the project was by definition very large, and because many of the ideas such as sharing common memory and time slicing the Emitter Coupled Logic (ECL) bus are almost unique, some of the architectural features developed have not been fully implemented in the prototype. The software, as well, has been simplified in order that the basically novel thoughts of Ramrod be demonstrated and proved to be viable.

The operating system has been simplified by allowing the user to load his tasks via the console in addition to using the master operating system functions to load tasks. Purpose designing the operating system for the evaluation of Ramrod also reduces the complexity of the operating system i.e. all the features in a complete operational system have not been included - only those that are absolutely necessary to run the test program.

It must be clearly understood that the omission of the above features does not in any way undermine their importance and contribution to the project. These features can easily be incorporated into the system because of the modularity of Ramrod.

An additional factor which is usually examined in the area of evaluation, is the question of memory utilisation, but as the control store is large enough for the operating system and because tasks reside in the common memory, this evaluation is not relevant in the present situation.

7.2 Factors influencing the Relative Comparison

In order to synchronise the slave microprocessor to the Time- Division-Multiplexed (TDM) common bus the "Ready" line of the 8085 has been used. Thus for a read memory cycle the address is first transfered across the bus and the 8085 is held 'unready' until the data returns from the memory. Initially this double cycle sequence only applied to the read memory cycle, but as the 8085 has a multiplexed data/address bus it was found necessary to make the write memory cycle a double cycle as well, thus effectively doubling the time taken for writing data to memory. This factor obviously influences the run time of the 8085 slave processor and must be kept in mind when comparing the execution figures of Ramrod and a uniprocessor system. The Bus Enabling Signals (BES) (figure 4.2 time slots) were originally chosen as having a period of 1 micro-second and therefore the logic on the memory cards was designed with this in mind to provide the read/write, select and clock pulses using monostable multi-vibrators. It has subsequently been determined that the BES frequency can be increased to 2MHz, thereby significantly reducing the run time of a task.

The method used for interrupting the bit-slice master processor is via the watchdog circuitry which takes 14 milliseconds to time-out. Therefore the execution time of the task should be reduced by this time period as an alternative method for interrupting could be designed. The slave processor can generate an exclusive address in order to signal the master, though the watchdog circuitry is still needed to indicate a malfunction. Thus there would be two interrupts from each slave.

The uniprocessor system used in the comparative studies was one of the slave processors executing in isolation. This is a preferred solution as it incorporates the double cycles mentioned above, and hence provides a direct comparison in terms of execution times.

7.3 Program Used in Relative Comparision

As was mentioned earlier, the architecture of Ramrod is similar to that of an Array Processor so it was decided to undertake the evaluation by making Ramrod do an exercise on an array of integers.

Figure 7.1 illustrates how the program runs and shows how data flow techniques are applied.

The program calculates the maximum of an array of numbers. The array is divided by the number of slave processors that are present and each subdivision becomes the input for the operators (slaves) 1, 2 and 3. The routines are initiated by the apearance of tokens (subdivisions). They operate on the arrays and are terminated when the output (maximum) occurs. Operator 4 can only be iniated when all of its tokens (maxima) are present at the input. It terminates once the output (absolute maximum) appears.

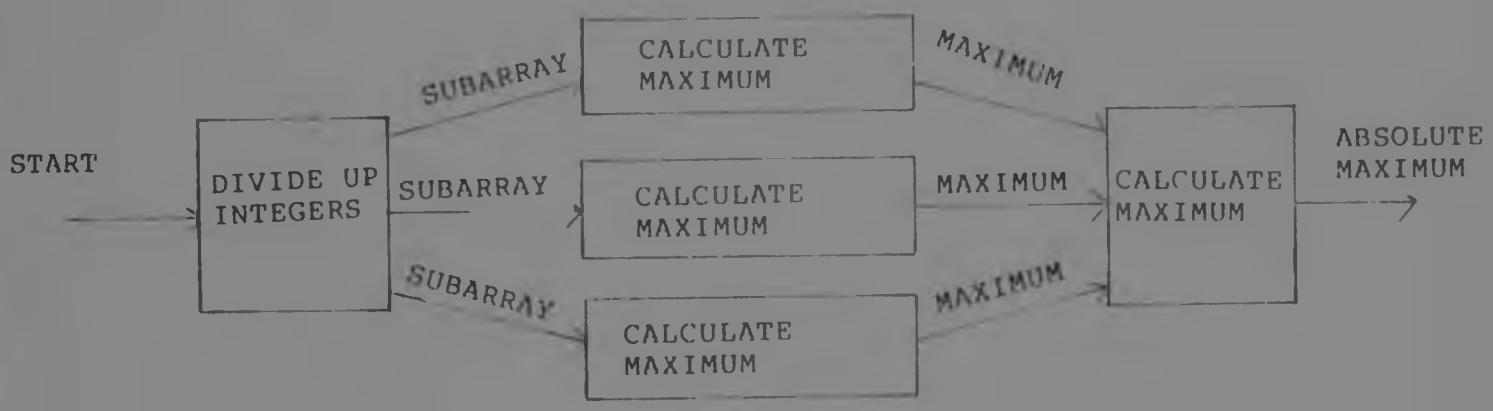


FIGURE 7.1 OPERATING SYSTEM SEQUENCE

7.4 Results

Figure 7.2 shows the execution sequence of the test program. When several slave processors are executing identical tasks simultaneously there is a possibility that two or more tasks will send interrupts to the master controller simultaneously, therefore these tasks contain dummy loops so that their execution times are not similar.

MEMORIES TO PROC4	INT	MEMORIES TO PROCS 1 3 5	INT	MEMORIES TO PROC2	INT	MEMORIES TO PROC4
FOR USER TO ENTER DATA		PROCS GET MAXIMA		PROC2 GETS ABSOLUTE MAXIMUM		USER VIEWS DATA

FIGURE 7.2 EXECUTION SEQUENCE

Below is a table which shows the steps for the test program with their execution times.

Function	Execution Time
1. Assign 3 memories to a slave processor so that user can enter array	13 microseconds
2. User generates integers, after insertion, which is detected and serviced	550 nanoseconds
3. These 3 memoriess are re-assigned to 3 slave processors, for calculation of maxima	10 microseconds
4. Slave processors calculate their maxima	1600 microseconds
5. Three interrupts are generated and serviced	1250 nanoseconds
6. These 3 memories are assigned to another slave processor so that it calculates absolute maximum of array	10 microseconds
7. This slave processor calculates maximum of 3 numbers	180 microseconds
8. Generation and service of interrupt	550 nanoseconds
9. Assign memories to console slave processor so that user can view result	12.5 microseconds

Table 7.1 Execution Times

Note: The time taken for the user to input the data is not relevant, and applies to the execution time of the single processor as well.

Thus the total run time for the system to calculate the maximum of 96 integers is approximately 1.8 milli-seconds. If one slave processor were to operate on the entire array it would take 5.2 milli-seconds whereas a processor which does not have any wait states inserted, takes 2.6 milli-seconds to operate on 96 integers (see figure 7.1)

7.5 Conclusion

Ramrod performs very well under the given conditions and when the Bus Enabling Signal (BES) frequency was indeed increased the machine became even more powerful. The results must be viewed whilst keeping this point in mind. The fastest time for Ramrod to do the above example was in the region of 1 milli-second thus making it 2.5 times faster than a single processor. However one must bear in mind that Ramrod has the ability to allow the processors to inter-communicate and therefore its overall power is difficult to estimate. In addition, if the operating system were better scheduled then there would have been no need to have a separate processor to maximise the relative maxima and the total run time could then be reduced by an added factor of 300 micro-seconds. It is estimated that it would

take a processor 1.3 milli-seconds to operate on an array of 24 integers (96/4 =24).

CHAPTER 8

CONCLUSION

"The end of the matter is, let us hear the whole; Fear GOD and keep his commandments; for this is the whole duty of man. For every deed will GOD bring into the judgement concerning everything that hath been hidden whether it is good or whether it is bad" [Eccl xii 13,14].

Very Large Scale Integrated (VLSI) microcomputer components are highly cost-effective because of the high volume at which they are produced, and therefore future computer architectures must utilise this dramatic advance in technology [GIL BEHR].

This thesis sets out to define a computer system which is highly cost-effective and whose architecture is based on data-flow techniques in order to provide a more efficient way of data access than the conventional computers.

However, the architecture was also based on a high degree of fault-tolerance, modular extensibility and simplicity.

8.1 Uniqueness of Ramrod

The project brought out the unique features, detailed below, in order to reconcile these seeminglv conflicting demands of modular architecture on the one hand and simplicity and fault-tolerance on the other hand.

1. The Time-Division Multiplexed (TDM) common bus

   It was shown that a common shared bus does not necessarily have a low bandwidth, and can indeed be used very efficiently to time division multiplex many devices - the key being the very short time required by each processor to access the bus.

2. Circular Bus

   The TDM bus is constructed in a circular fashion and joined at the ends. Ramrod proves the viability of using a circular bus to improve signal levels and hence to decrease the maximum delay between devices.

3. Distributed Operating System

In order to increase the reliability of the system, as a whole, and to provide for graceful degradation the operating system is distributed amongst the processors, and the kernel of the operating system is built into the hardware of the bit-slice master processor.

4. Local/Global Memory

Another feature of Ramrod is that it does not differentiate between local and global memory. This offers many useful properties such as a simple mechanism for implementing a distributed data base.

5. Inter Process Communication (IPC)

There are three methods by which Ramrod can achieve IPC, thus making it a good test bed for developing ideas about IPC:

(a) via the I/O module using Ethernet

(b) via the common, shared bus

(c) via the Master Processor

6. Readily Available Components

The architecture, allthough novel, uses freely available components and thus maintainability and extensibility are assured.

8.2 Commercial Viability of Ramrod

Ramrod can be used in such diverse applications such as process control on the one hand and data base management on the other hand, and this is perhaps one its most important contributions to technology. In addition the system is relatively cheap but powerful. A cursory calculation shows that the cost of developing and marketing this computer can be in the region of R25,000 - R35,000 thus placing it in the lower bracket of minicomputers, with, of course, more relative power.

The cost of software development for the purpose of testing Ramrod has been included in the above calculations but the cost of producing software for making the machine as versatile as is claimed in Chapter 1 could not be ascertained and is clearly considerable.

8.3 Critical Analysis of Ramrod

It has been claimed that Ramrod can support 50 Microprocessors and 50 memory seqments, but in view of the investigation carried out (see appendix C) into the ability of ECL to drive the circular bus, additional circuitry is required and this might slow down the propogation delay which would have an effect on the overall system throughput.

The actual operation of Ramrod was marred by problems relating to the construction of the ECL bus. Whilst the timing was shown to be viable, the critical nature of this timing made it subject to temperature problems.

ECL has a very high heat dissipation problem which depends on the level of the power supplies and the termination resistors which in turn affect the ECL logic levels. Any variation in ambient temperature clearly affects all the parameters, and during hot weather the system suffered from occasional intermittent faults - attributed to timing problems.

Finally, the choice of the actual physical bus was a poor decision - the interconnection from the edge connectors to the scotch-flex system proved to be unreliable and was the source of many mechanical failures.

8.4 Future Enhancements

The basic design of Ramrod is sound but there is still room for improvement which can be achieved by:

1. Improving the present design to overcome mechanical problems resulting from the physical bus construction

2. The incorporation of those features mentioned in Chapter 1 so as to permit a fully operational vehicle which may be used in long-term experiments.

In particular the following ones required attention:

1. In order to ensure stable power supplies on each board, regulators must be resident on each printed circuit board.

2. The latch module needs to be redesigned so that the ECL chips are closer to the bus.

3. Both the TTL and ECL Busses must be constructed from a flexible printed circuit board so as to provide a more reliable mechanical structure.

4. In order to increase the number of slave processors

and memory segments, the loading of the ECL bus chosen needs additional investigation.

5. In order to include an intelligent I/O interface the work on Ethernet needs completion.

6. Better software support is needed to develope the microcode. A related project is investigating a highly flexible microassembler and emulator [WILD].

7. Perhaps standard processor and memory cards could be used instead of purpose-built hardware thus making Ramrod universal.

8.5 Conclusion

In summary, Ramrod has been designed and built to a prototype stage and tests were run to show its viability. Although it suffers from certain problems relating to the mechanical structure and also is somewhat temperature dependant, it has proved to be a most successful and in many ways unique design. In providing an extremely powerful computer which makes use of freely available components it has met its prime design objectives and illustrated much promise for future development.

APPENDIX A

EMITTER COUPLED LOGIC

A.1 Introduction

A comparision of the fastest commercially available state-of-the-art technologies (ECL and AST) as well as a discussion on the use of ECL is outlined in the following pages.

Emitter Coupled Logic

Advantages	Disadvantages
Propogation delay 2-3ns	Has different power supplies from standard TTL
Low output impedance	Has different logic levels from standard TTL
Can drive transmission lines	Extra power supply for transmission line
Very high fan-out	All outputs need pull down resistors
Complementary outputs	High power dissapation
High output drive capability	large ground plane needed
Slow rising edges	
Wire-oring possible	

High input impedance, therefore unused inputs go low

Advanced Schottky TTL

Advantages	Disadvantages
TTL compatible	Propogation delay twice as long as ECL
i.e same levels, powe. supplies	cannot drive transmission lines
Low power consumption	No wire-ORing
High noise immunity	Fast output transition therefore reflections and crosstalk
	Thresholds low levels slightly offset from TTL

Emitter Coupled Logic is a non saturating form of digital logic which eliminates transistor storage time as a speed limiting characteristic and permits very high speed operation. "Emitter Coupled" refers to the manner in which the emitters of a differential amplifier within the integrated circuit (IC) are connected. The differential amplifier provides high impedance inputs and voltage gain within the circuit. Emitter follower outputs restore the logic levels and provide low output impedance for good line driving and high fanout capability.

A typical ECL gat structure is shown in figure A-1 as well as the available separate functions.

ECL has two ground inputs which eliminate crosstalk between circuits in a package. In order that unused inputs may be left open 50 Kilo-Ohm "pinch" resistors drain input transistor leakage current and hold these unused inputs at a fixed logic zero level.

Typical logic levels for ECL are -0.98v which is a logic high level and -1.75v the logic low level.

In order to increase logic flexibility, speed and power efficiency two techniques of connecting the differential amplifiers are used. Figure A-2 illustrates the SERIES GATING technique which permits the generation of upto 2^n logic functions from n inputs with one current source, while COLLECTOR DOTTING (illustrated in figure A-3) allows the logic AND function to be achieved by interconnecting one collector node of separate differential current switches together. A third technique, WIRE-ORing, enables the logic OR function to be generated by tying together two or more emitter follower transistor. A disadvantage of ECL is that there is a limitation of the number of WIRE-OR connections of 6. Therefore bus drivers need to be used when this limit is exceeded.

BY COURTESY OF MOTOROLA INC.

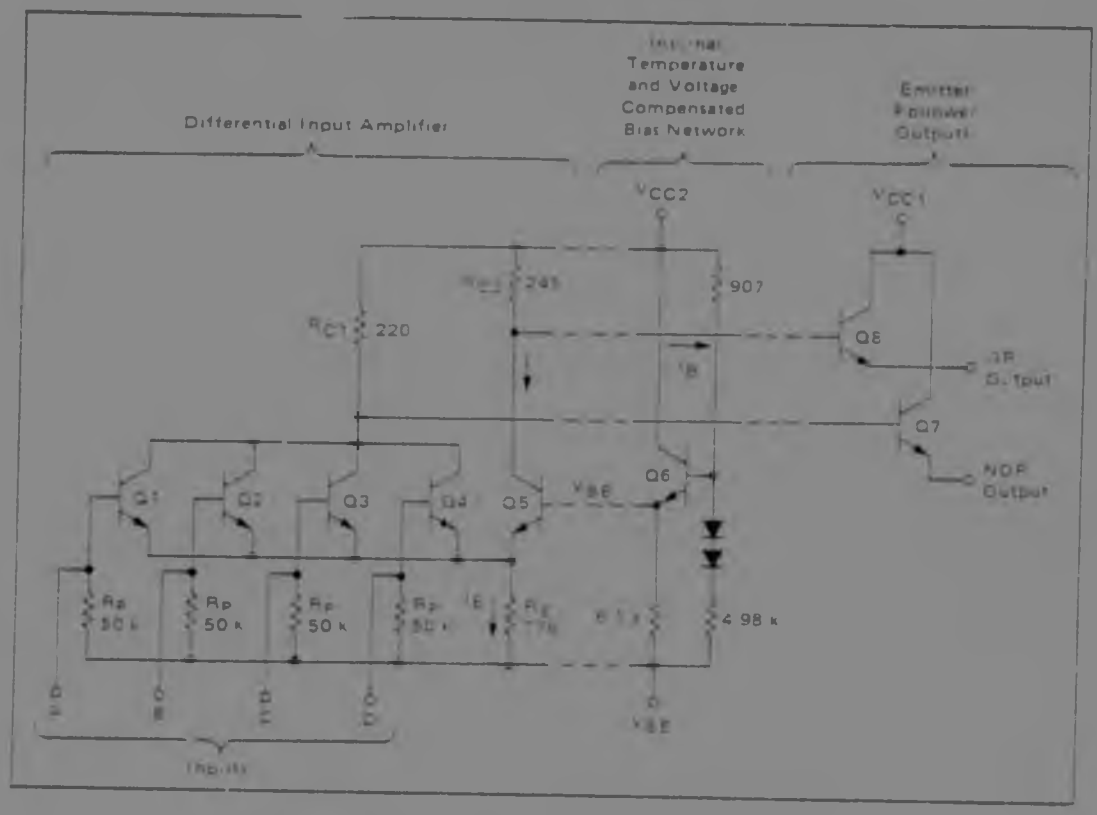


FIGURE A-1 ECL STRUCTURE

BY COURTESY OF MOTOROLA INC.

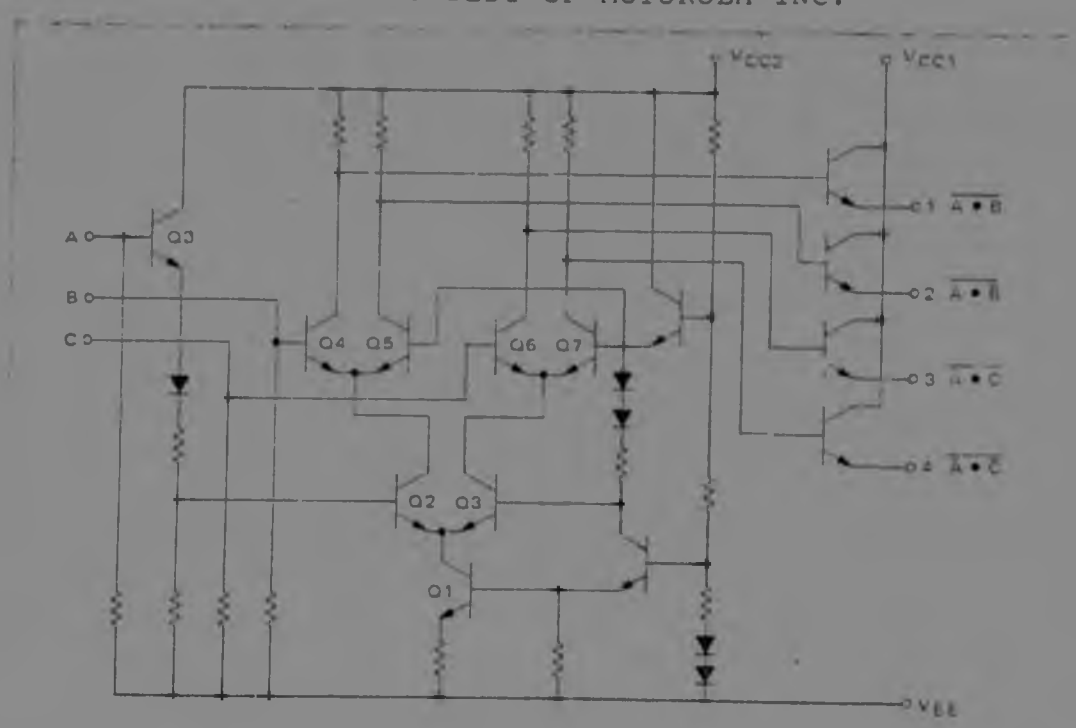


FIGURE A-2 SERIES GATING

BY COURTESY OF MOTOROLA INC.

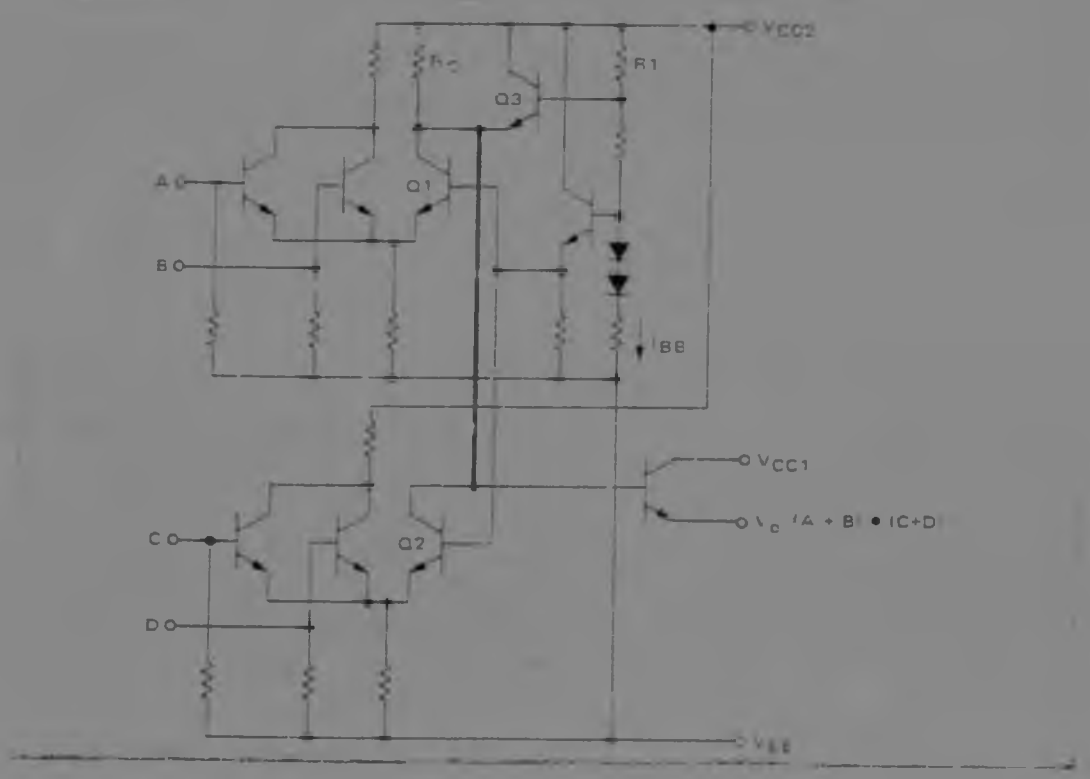


FIGURE A-3 COLLECTOR DOTTING

BY COURTESY OF MOTOROLA INC.

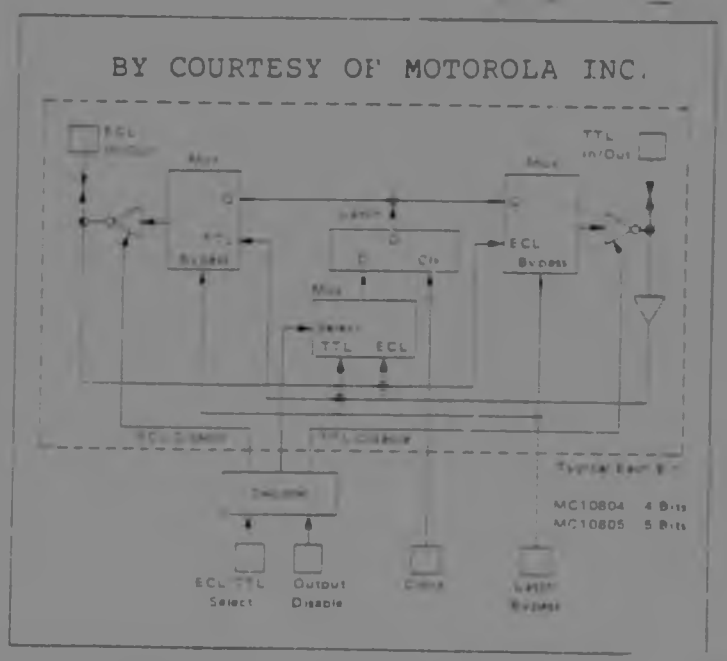


FIGURE A-4 10804 LATCH

Using ECL for high speed logic design can result in more problems than using AST transistor transistor logic (the propagation delay is about 2ns, and thus delays are introduced from the wiring). Therefore wiring lengths should be reduced as much as possible. Using wiring with 2.0ns/ft delay means that there is approximately one gate delay for every foot of wiring.

Transmission line principles should be employed in order to design interconnections between ICs. Line lengths approach the quarter wavelength of the signal and therefore distortion and reflections can occur. Lines must be properly terminated with matching impedances to avoid these and other associated problems.

ECL designers have further minimised crosstalk by deliberately slowing the rise and fall times to more than 3ns.

Manufacturers recommend that only one-sided printed circuit boards be used, keeping the second side as a ground plane in order to reduce noise generation as well.

The characteristic impedance $Z_\emptyset$ of a single line over a ground plane separated by a dielectric medium ,i.e. microstrip lines, is calculated by;

$$Z_\emptyset = \frac{87}{(e_r+1.41)^{**}1/2} \cdot \ln \frac{\{5.98h\}}{\{.8w+t\}}$$

where e_r = relative dielectric constant
w = width of microstrip
t = thickness of microstrip
h = thickness of printed circuit board

ECLs logic levels of -0.98v and -1.75v are derived from the -5.2v power supply. The reason for this power supply as opposed to the normal +5v supply is that it helps to reduce noise generation when the emitter followers switch from one level to the other.

The designers of ECL circuits incorporated another useful feature into their designs by including at least one inverted output signal in an IC package. For example the 10104 quad 2 input AND gate has one inverted output i.e. the NAND function is derived.

The 10195 HEX INVERTER/BUFFER has 6 EXCLUSIVE-OR gates with one input commonned. Therfore the IC can be configured as a buffer or inverter.

A.2 Sample Data

The 10104 Quad 2 Input AND gate

Propogation delay is 2.7ns typical while rise and fall times are approximately 3ns. The power consumed per gate is 35mw(no load).

The 10804 ECL/TTL Inverting Bidirectional Transeiver with Latch

Referring to the block diagram in figure A-4 the reader will notice that there are four control signals needed to operate this package. The OUTPUT DISABLE when at a logic low level disables both the ECL and TTL output buffers, while at a logic high level these buffers are enabled. The ECL/TTL signal allows control of the direction of data transfer and translation.

The LATCH BYPASS select line allows the latch circuitry to be bypassed for fast data transfer. When it is a logic low level data is directed to both the latch input and output buffer simultaneously, and this enhances the speed of translation and throughput.

APPENDIX B

MICROPROGRAMMING AND BIT-SLICE TECHNOLOGY

Bit-slice technology and microprogramming are reviewed in this appendix , in order to provide a general background of the master controller which has been developed to control the operation of the processors in the multiprocessor structure.

Bit-slice microprocessor families are not revolutionary, rather they represent a new stage in the evolution of the design of central processing units (CPU's).

In machines designed from small scale integrated technology where integrated circuits could only hold a small number of basic components the ALU would occupy one printed circuit board, and the registers another board etc. so a complete CPU would occupy many boards or cards. The logic was commonly separated into n bit wide sections thus one card would contain a small chunk of the total processing unit,and the cards were cascadable.

With the introduction of MSI and LSI it became economically feasible to include more of the control logic onto one 'chip', and eventually the single 'chip' microprocessor was developed.

The bit-slice microprocessor represents a further stage in the developement of microprocessor technology in that the processor is again sliced as before, but this time each 'chip' is a complete chunk, and can be cascaded to form a n bit wide processor. In addition to that the bit-slice microprocessor has been specifically designed to be used in microprogrammed machines.

The organisation of a conventional computer is shown in figure B-1. Essentially, four major sections may be identified:

the memory
the input/output facilities
the ALU
the control unit

The control unit or central processing unit (CPU) provides for overall control of the various sections of the computer.

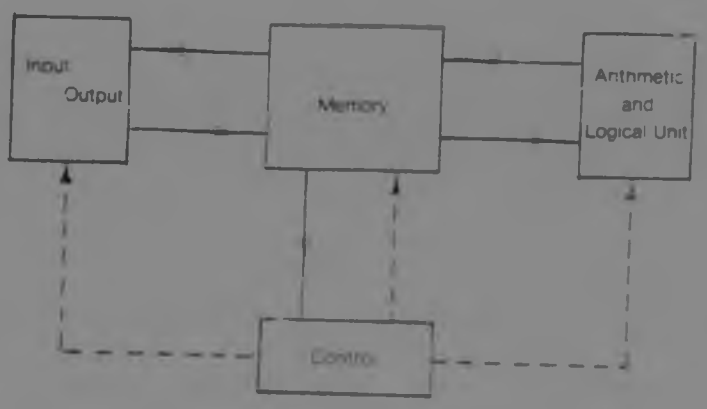


FIGURE B-1 CONVENTIONAL COMPUTERS

The organisation of a microprogrammed computer structure is shown in figure B-2. The essential dfference between the above two structures lies in the mode of operation of the CPU. In the microprcgrammed computer, the control store contains sets of primitive operation codes, which are termed microinstructions. Each component part of a microinstruction specifies an elementary logical or arithmetic process to be effected in the computer. A machine code instruction is executed by a series of microinstructions contained in the contrcl store.

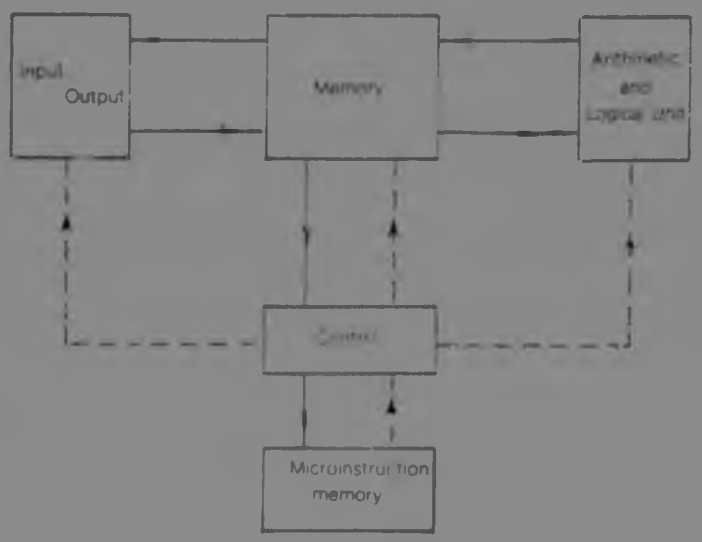


FIGURE B-2 TYPICAL MICROPROGRAMMED COMPUTER

Microprogramming allows the designer flexibility in the design of his instruction set. A macro instruction or machine code instruction is performed by executing several micro instructions in sequence. The machine code instruction is used a pointer to this sequence. These micro instructions are usually stored in a control memory within the bit-slice architecture.

The micro-instruction word is broken up into several fields, each of which defines a particular function within the machine. Thus the longer the word the more is achieved in any one instruction and the faster the computation. The designer has to analyse the trade-off between the width and the depth of the instructions. A micro-instruction word is typically 32, 56, 64 or 128 bits wide.

The structure of the system can be altered by the organisation of the microprogram word fields, allowing the design to closely match the function it must perform.

A bit-slice microprocessor system requires a lot more components than the two previously mentioned microprocessor designs, and will therefore be more expensive and consume more power, but will be more powerful and faster.

A control memory, usually a programmable read only memory (PROM), contains the microprogram words. The operation of the system is as follows: A sequence of micro-instructions in this memory is executed to fetch an instruction from external main memory, which is then decoded and passed through a mapping PROM to generate the address of the first micro-instruction which is to be executed to perform the required macro-instruction. The sequencer controls the branch to the required address. The instructions are fetched from the control memory and then other operations such as , ALU functions,testing etc. are performed by the rest of the system. Then a branch is made back to the instruction fetch cycle, at which point there may be branches to other sections of micro-code.

The pipeline register essentially splits the system into two parts. It contains the micro-instruction currently being executed. This instruction is fed to the rest of the system which performs the required operation while the next instruction is fetched and placed in the pipeline register. Thus the presence of this register allows the micro-instruction fetch cycle to occur in parallel with the data operation rather than serially, effectively doubling the clock frequency.

APPENDIX C

THE MODELLING OF THE CIRCULAR BUS

C.1 Introduction

An investigation into the operation of the parallel ECL circular bus was undertaken by Messrs Bradford and Hunter as a final year undergraduate project and was supervised by the author.

A preliminary literature survey showed that very little information is available in the field of circular busses. Zoccoli and Sanderson [ZOC] claim that they use a circular ECL bus for their computer but however do not give enough detail. It is known, as well ,that the Cray super-computers use circular ECL busses but there is no information about this for general public consumption.

Therefore in order to fully understand the operation of the bus it was decided to model the bus as well as conduct practical experiments.

C.2 Model of the Bus

A computer program was used to simulate the operation of the bus and this mathematical model was compared against the measurements observed practically. (The program can be obtained from the Dept. of Elec. Eng. at the University of the Witwatersrand).

The model assumes that there are no dielectric or copper losses and therefore the characteristic impedance of the bus Z becomes:

$$Z = (L/C)^{**}1/2$$

Where L = .56 micro-henry's/m

and C = 82 pico-farads/m .

therefore

$$Z = 82 \text{ Ohms/m}$$

But taking into account the capacitive loading of the edge connectors of 2 pico-farads/connection Z = 72 Ohms/m.

The propogation delay of the bus

$$T = (L.C) ** 1/2 = 7.85 \text{ nano-seconds/m}$$

Similarly the characteristic impedance of the tracks and its propogation delay are:

$$Zt = \frac{87}{(e_r+1.41)^{**1/2}} \ln\frac{(5.98t)}{(8w)} = 130 \text{ Ohms}$$

where t = thickness of the track

and w = width of the track

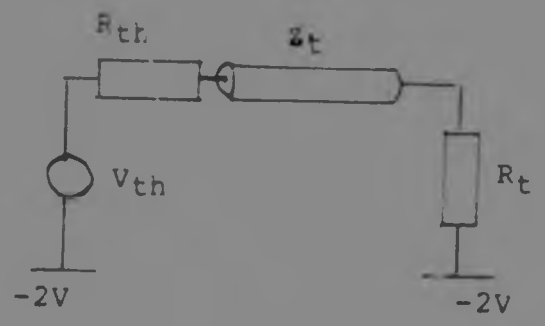
Tt = (L.C)**1/2 =17.36 nano-seconds/m

e_r = relative dielectric constant

Each board in the system has a terminating resistor to -2 volts and when all of the gates are disabled the voltage on the bus settles to -2 volts. If a transceiver is enabled to transmit a high level (-.85V) then there is a voltage swing of 1.15 V in 3.5 nano-seconds (propogation delay of the gate). It must be noted that receiving gates represent the same high impedance to the bus as do inactive gates. This disabled to high level transition as well as the inverse transition only are considered as the voltage swings are large compared to the other voltage swings (.375 V).

The Thevenin Equivalent of a driving gate is shown in figure C.1 and has a Vth= .7 V and a source impedance of 7 Ohms irrespective of the load current.

As the model assumes no dielectric or copper losses, direct modelling of lumped capacitance is prevented, and the capacitance is rather modelled as being distributed. The rise and fall times are modelled as being linear.



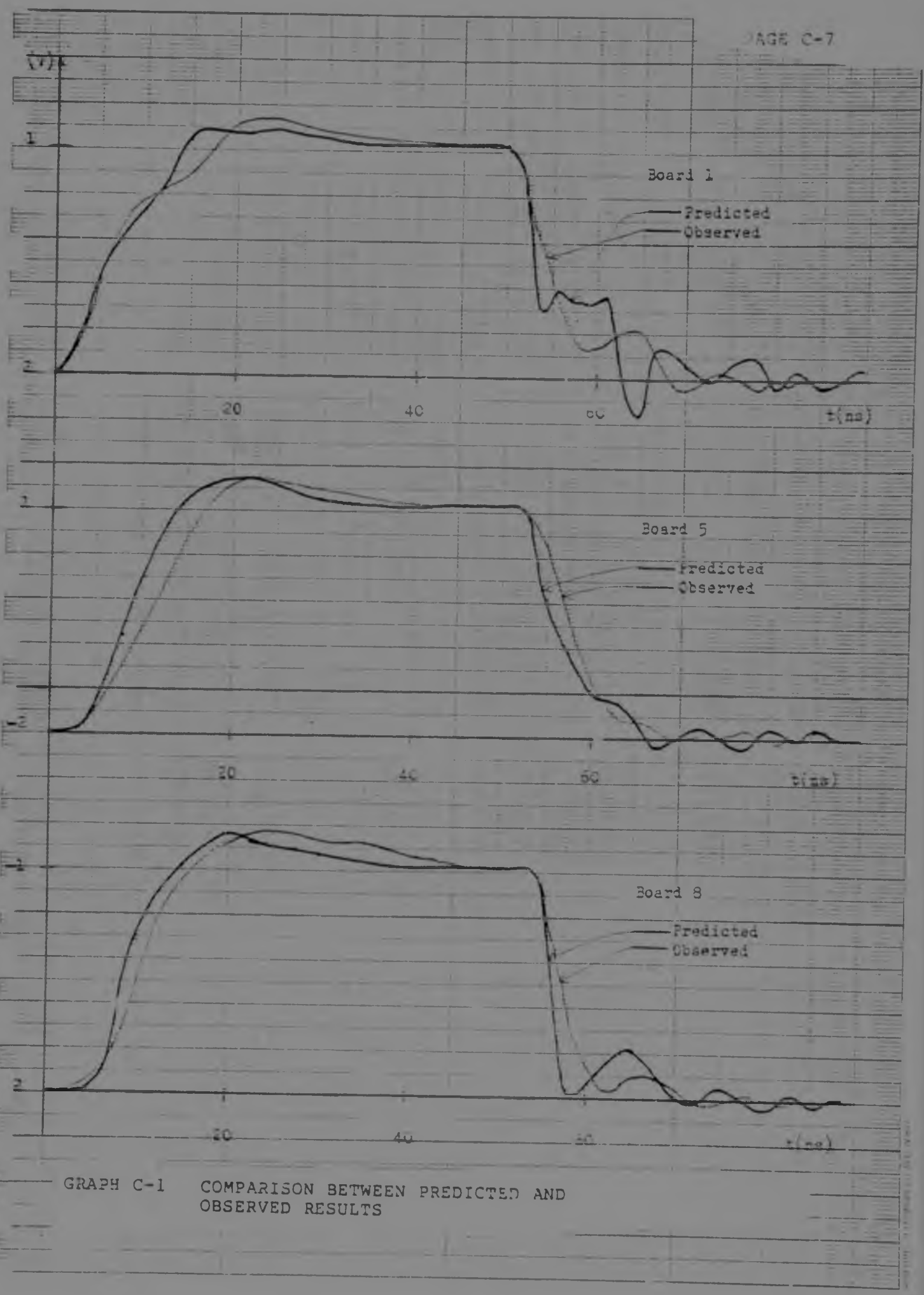
R_t = 50 ohms Z_t = 50 ohms

R_{th}= 7 ohms V_{th}= -0,689 Volts

FIGURE C-1 THEVENIN EQUIVALENT OF DRIVING GATE

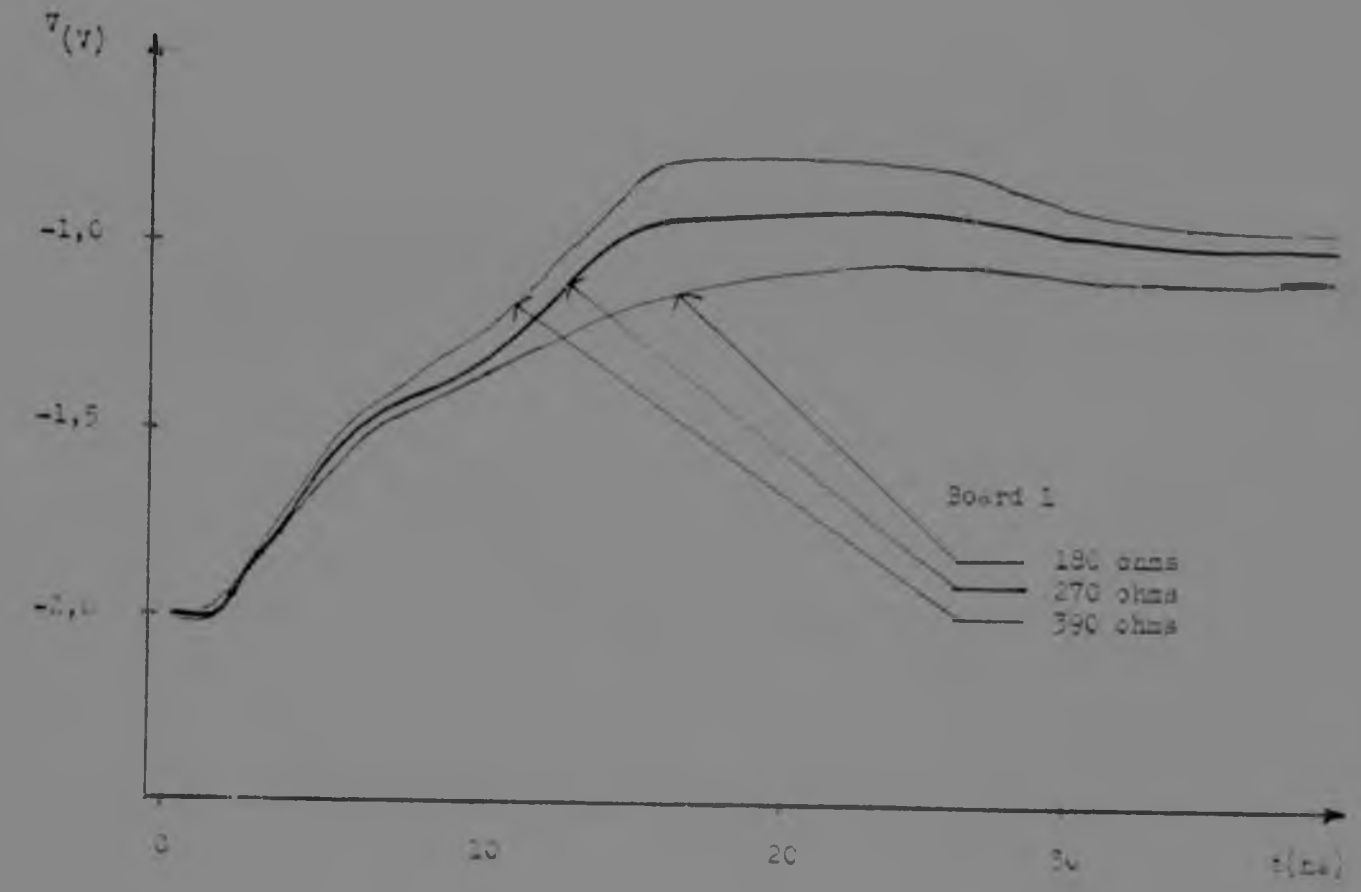
C.3 Results

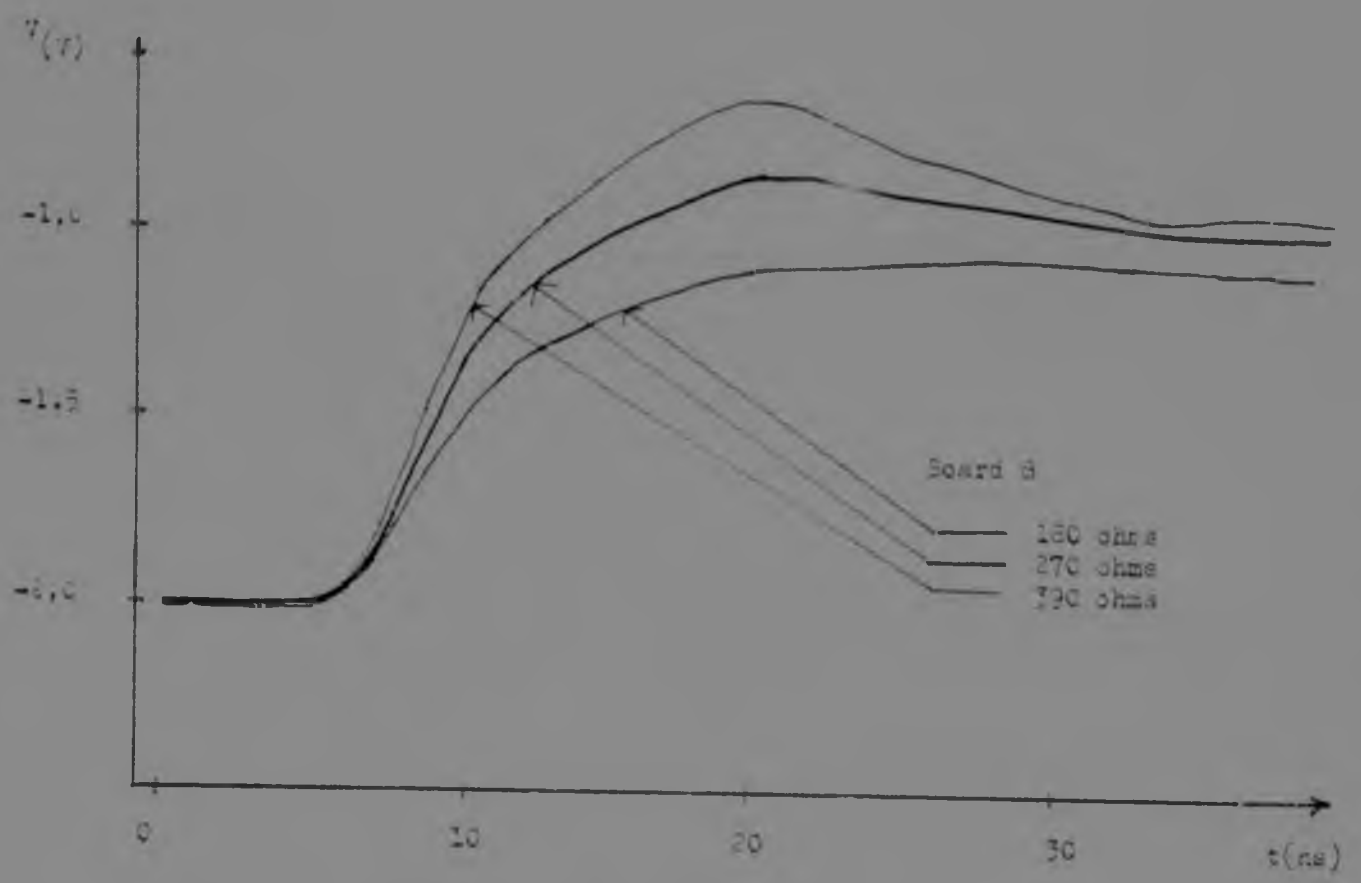
Graph C.1 shows a comparison of the predicted and observed results on various boards, on the bus , for a termination resistance of 270 Ohms. It reveals a difference in the rise and fall times of 2 - 3 nano-seconds which can be attributed to the assumption of a lossless line.



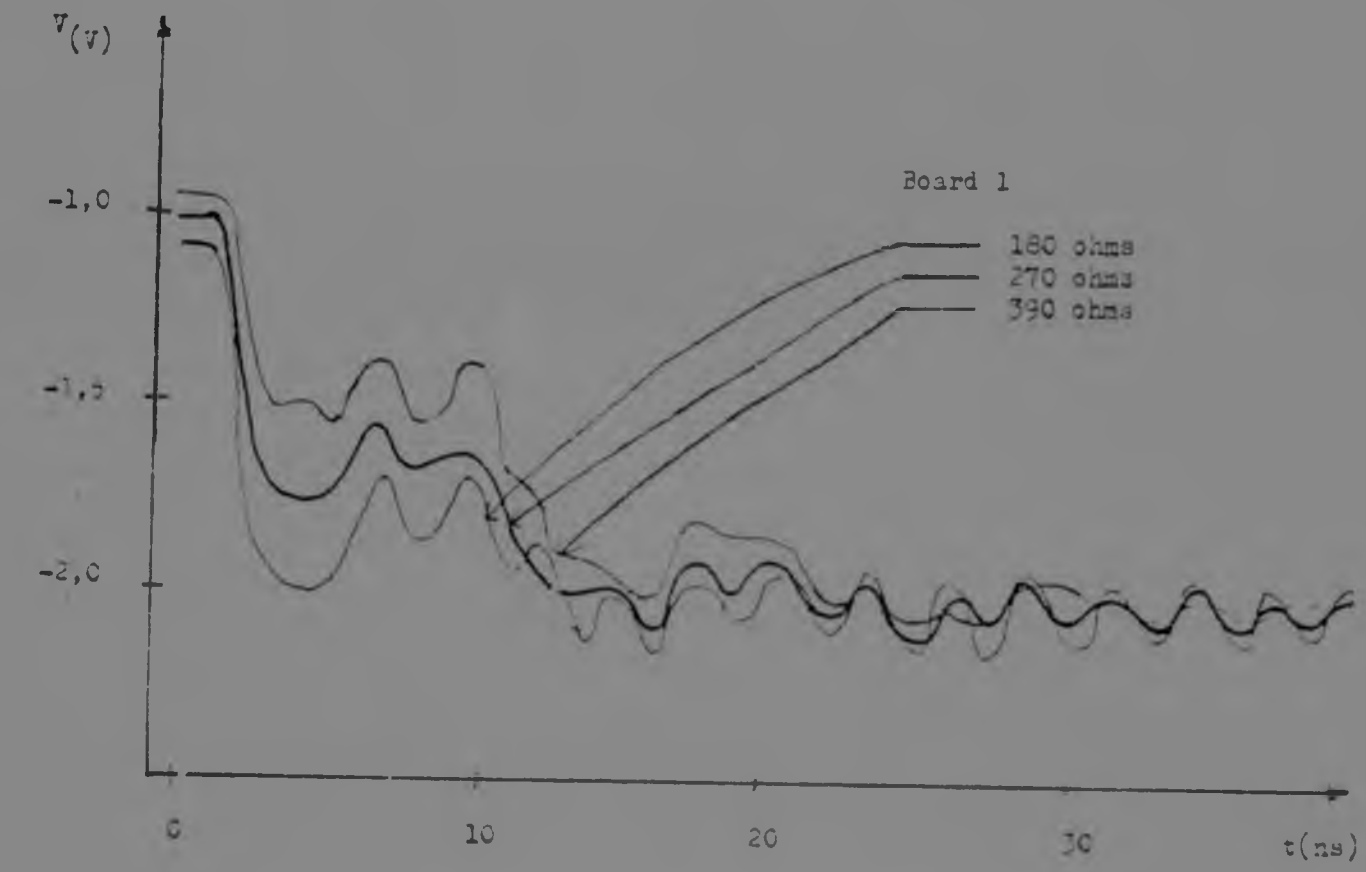
GRAPH C-1 COMPARISON BETWEEN PREDICTED AND OBSERVED RESULTS

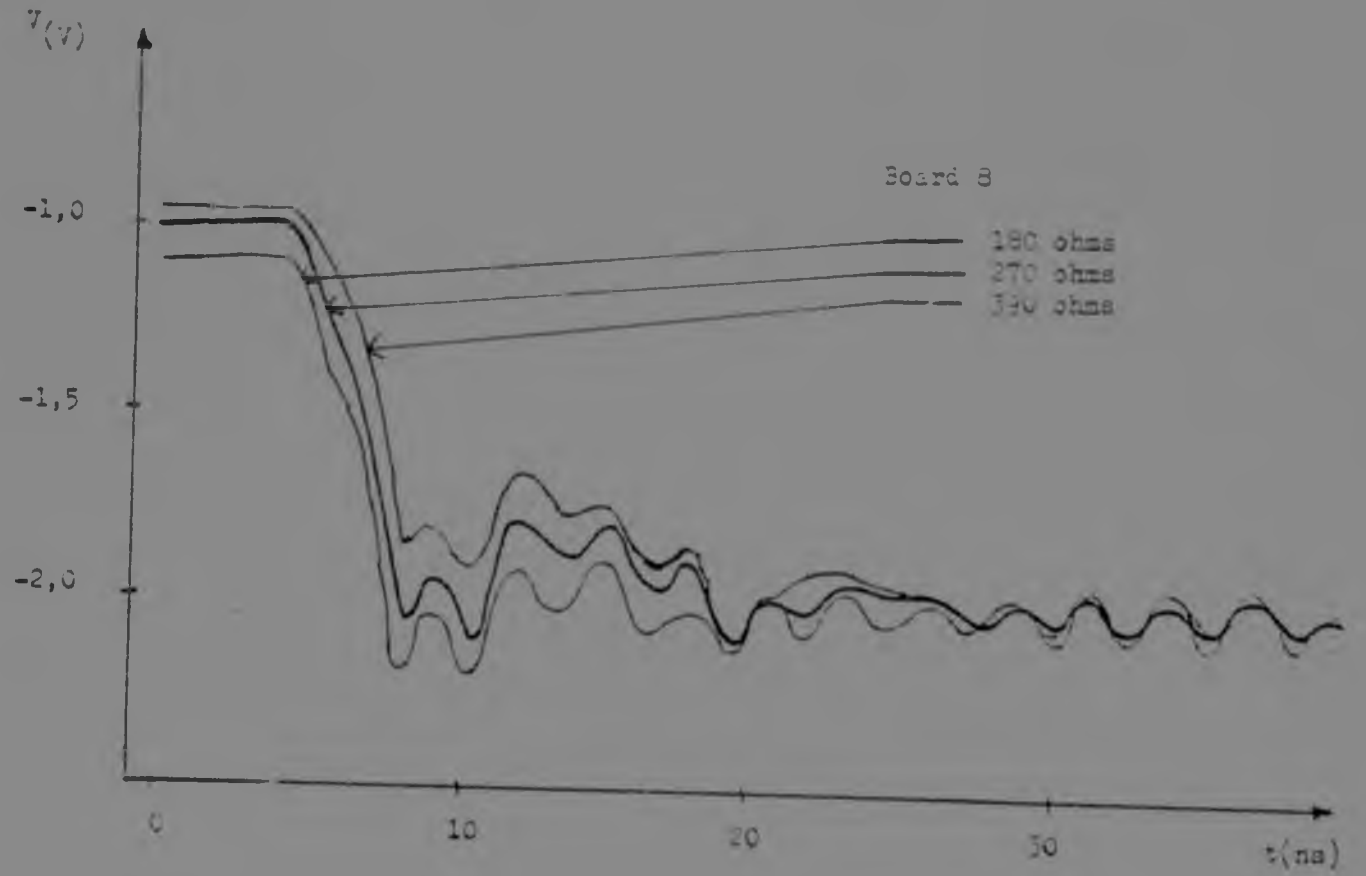
Graphs C.2 and C.3 compare the rising and falling edges for different terminating resistors and it appears that there is a critical resistance for a good termination.





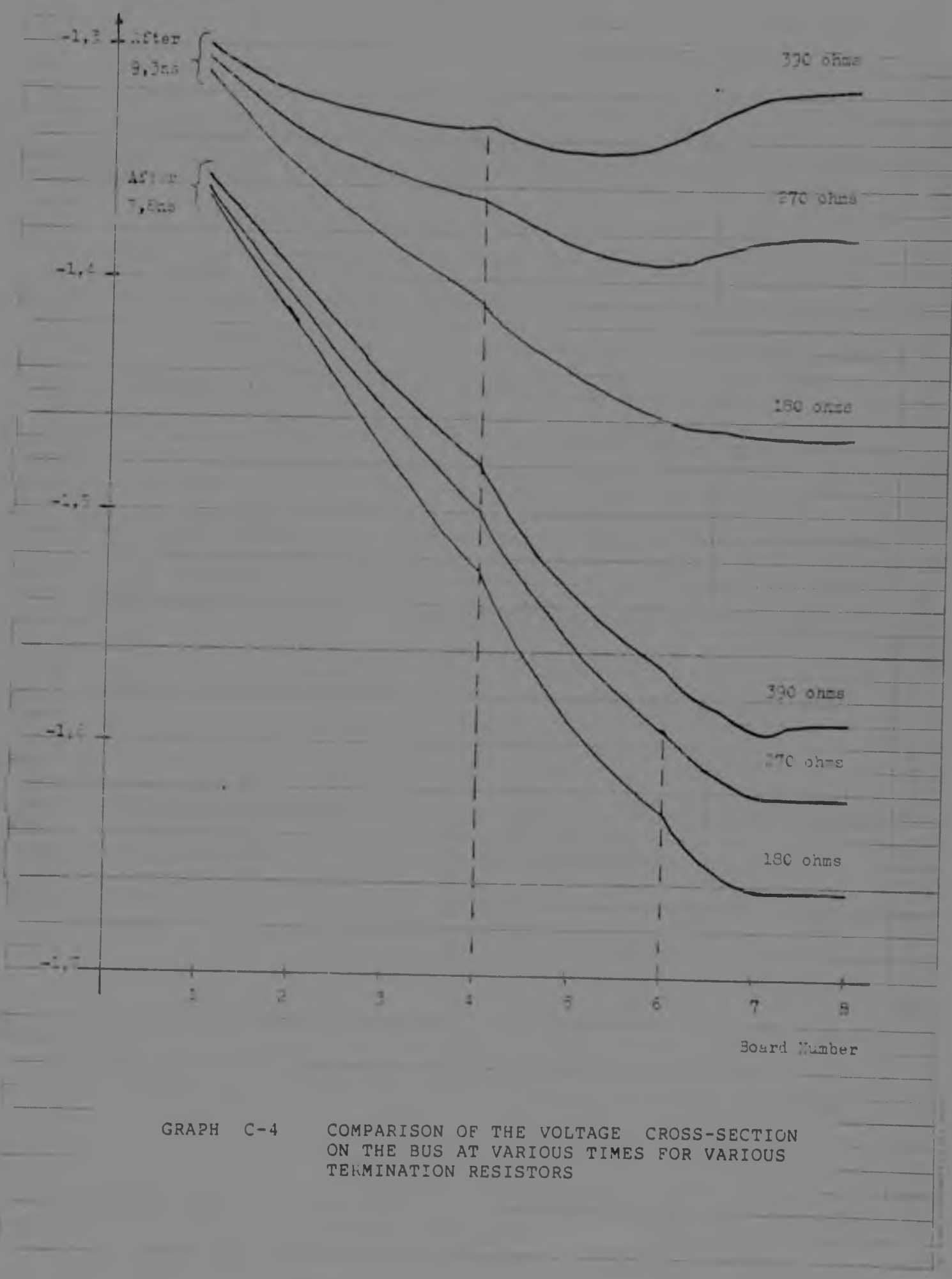
GRAPH C-2 COMPARISON OF RISING EDGES FOR VARIOUS TERMINATION RESISTORS





GRAPH C-3 COMPARISON OF FALLING EDGES FOR VARIOUS TERMINATION RESISTORS

Graph C.4 shows the cross section voltage along the bus at various boards. The slope of the wavefront determines whether an overshoot will occur or not.



GRAPH C-4 COMPARISON OF THE VOLTAGE CROSS-SECTION ON THE BUS AT VARIOUS TIMES FOR VARIOUS TERMINATION RESISTORS

As the signal passes each board its magnitude is decreased and hence the slope of the voltage cross section becomes steeper than the critical slope and no overshoot occurs. Decreasing the termination resistance decreases the transmission coefficient and no overshoot is obtained. This also increases the reflection coefficient and allows less of the incident pulse to arrive at the gate. If this resistance is chosen carefully enough then the reflection coefficient can be increased to allow large reflections but not have too much of an overshoot, and allow enough of the pulse to arrive at the gate for correct detection.

C.4 Conclusion

ECL gates can drive a 50 Ohm load terminated to -2 V. For a high level output (-.85 V) the current drawn is 23milli-amps (ma) but the manufacturers claim that MECL 10,000 series can source 50ma for surge conditions.

Once a stable steady state logic level is reached then there is a constant flow of current and only DC conditions apply. Thus for 10 boards there are 10 resistors connected in parallel therefore the effective R = R/10. At the high level (-.85 V) the current drawn is 1.15/R/10. This current must not exceed 50ma thus R > 230 Ohms and a 270 Ohm termination resistance is recommended.

This termination is however for a fixed number of boards and if the number varied then the current and logic levels would be changed. Figure C.2 shows a resistor-capacitor network which overcomes this problem. R2 is chosen so that at stable conditions the equivalent load is 50 Ohms and Rcap =25 Ohms which allows the maximum 50ma surge current to flow. The capacitance slows down the rise and fall times but improved logic levels are introduced.

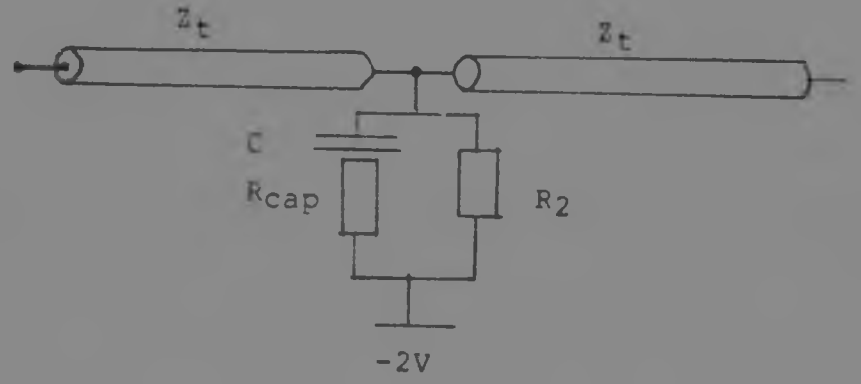


FIGURE C-2 CAPACITOR RESISTOR NETWORK

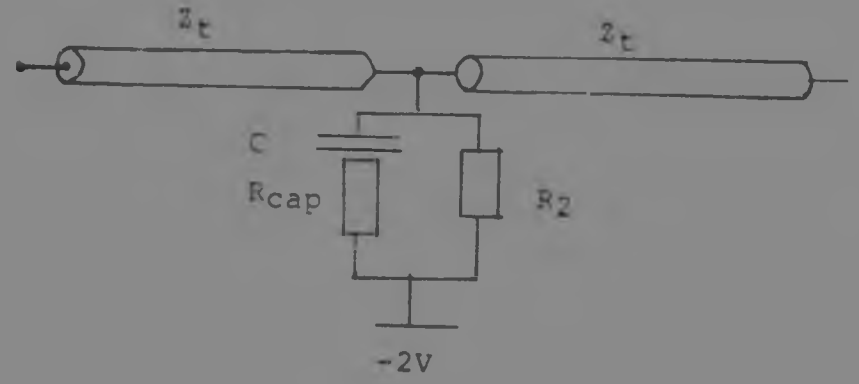


FIGURE C-2 CAPACITOR RESISTOR NETWORK

APPENDIX D

CURRENTLY AVAILABLE MULTIPROCESSORS

Currently available multi-microprocessors are reviewed below, in order to appreciate how the author chose the present structure of Ramrod.

D.1 CYBA-M

Cyba-M was a vehicle for research into multi-microprocessor systems initially undertaken by Swansea University College, and now at UMIST in Manchester. Figure (D-1) shows its basic structure, consisting of 15 identical Processing Elements, each of which comprises a microprocessor, a switch and some local memory. The global memory is a 10 Mbyte/sec memory, accessed through a 16 port switch, which determines the highest priority request generated by the node switches The Image memory (which provides the I/O/ facilities), is a distributed bus structure with a maximum data rate of 2.5 mbytes/sec. It is accessed through another 16 Port switch which is functionally identical to the Global Memory Switch. The 16th port is for use by the command console, which exercises total system control.

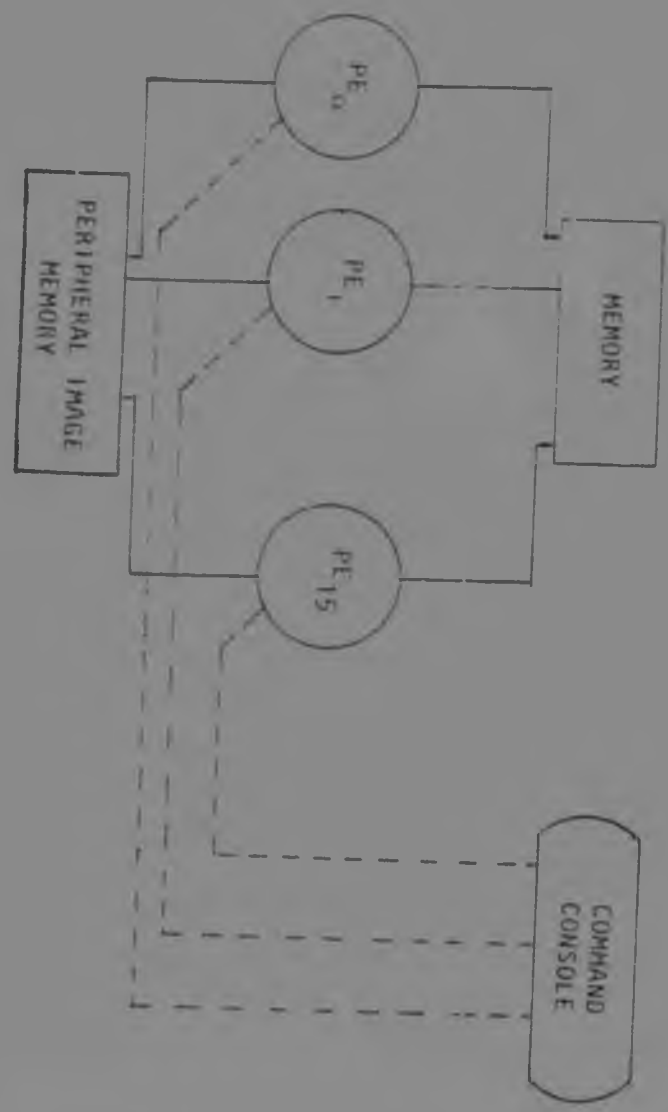


FIGURE D-1 CYBA-M

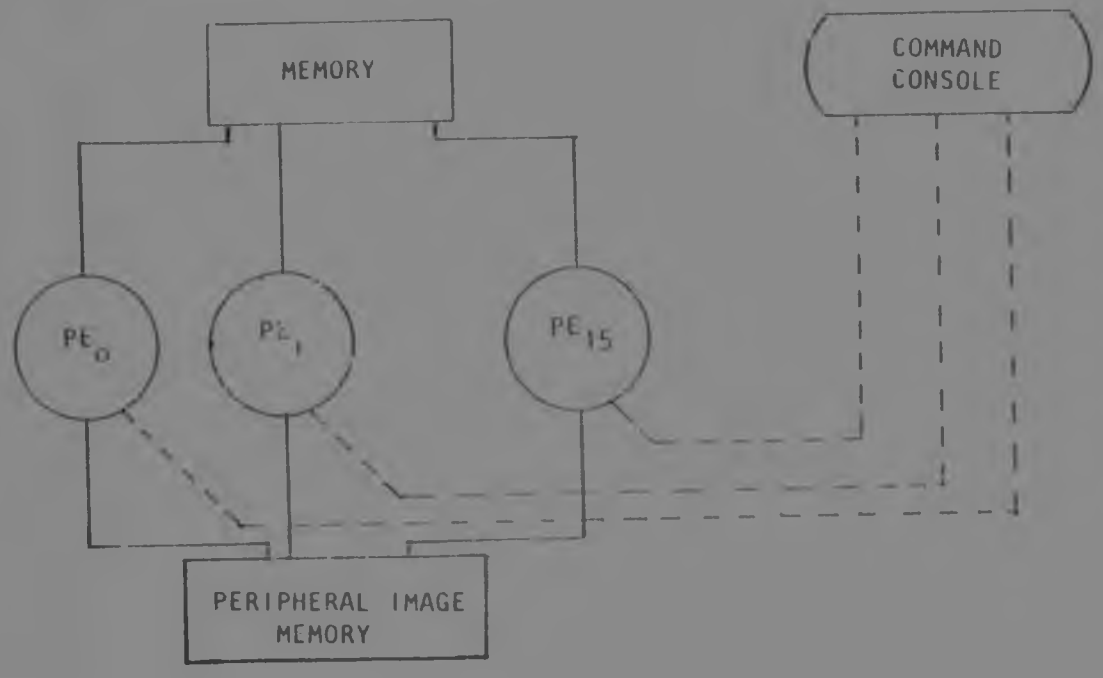


FIGURE D-1 CYBA-M

The disadvantages are:

The command console, similar to a Master Processor, is very complex from both the hardware and software points of view. The Global Memory is very fast and , being multiport, is therefore very expensive. In addition , the priority circuitry is complex. The switches are relatively simple (2-1 multiplexers) but nevertheless add to the complexity of the whole system.

D.2 The Siemens 4004/220/230

The design is based on the star configuration and comprises a dedicated central processor, a dedicated input/output processor , a hard wired maintenance processor and a memory system. All these work asynchronously and exchange information via a co-ordinator (figure D-2).

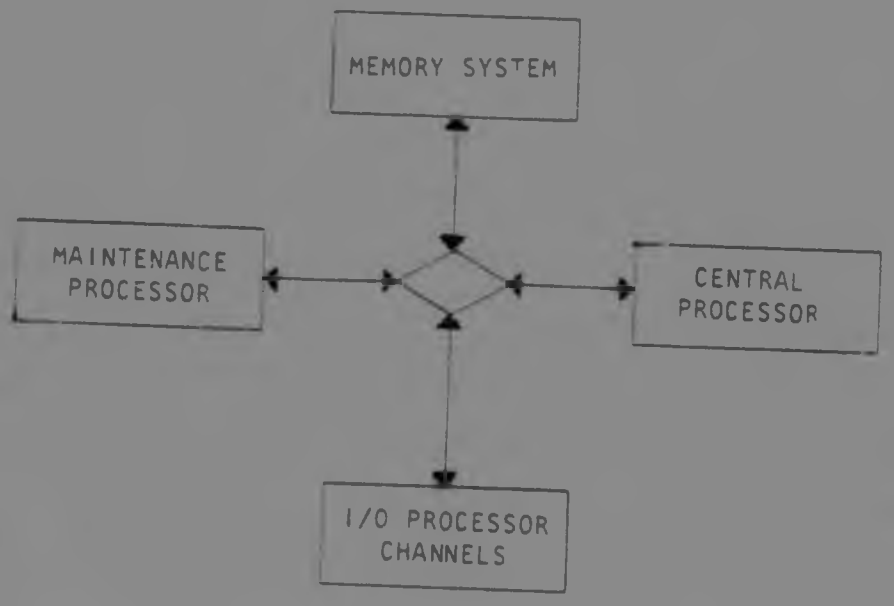


FIGURE D-2 SIEMENS 4004/220/230

The disadvantages are:

Each processing element is dedicated to a paticular function and therefore if it fails, that function can no longer be carried out. There is no redundancy in the system to allow for such failures, and the system allthough it has a maintenance processor , is not able to readily recover from faults.

D.3 The Siemens SMS 201

The SMS 201 has a multiple Instruction Multiple Data (MIMD) structure for high speed numerical computations. Each processor (PR) has a dedicated Arithmetic Processing Unit attached to it. In addition each processor has its own program and data memory as well as a communication memory (CM) which connects the module to other modules, and to a main processor (MPR) via an interconnection network (ICN). (figure D-3)

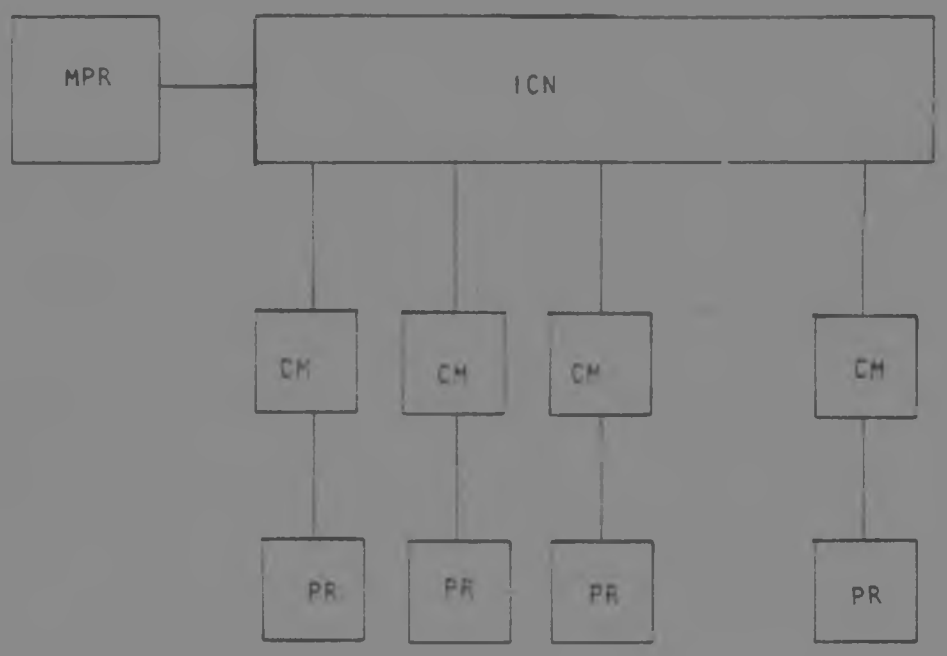


FIGURE D-3 SIEMENS 201

The disadvantages are: Too much reliance is placed on the main processor. The communication memory is the channel for inter-processor communication and as such, is quite complicated and therefore expensive. The interconnection network must be sufficiently intelligent to cater for priorities and to resolve conflicts.

D.4 The Carnegie-Mellon C.mmp

The multiprocessor is comprised of 16 DEC PDP-11 minicomputers, each having its own private memory space and own input/output device. The PDP-11 Unibus is used for I/O as well as for inter-processor communication. There is a large shared memory which is accessed by the processor's address translator through a 16 by 16 crossbar switch (figure D-4).

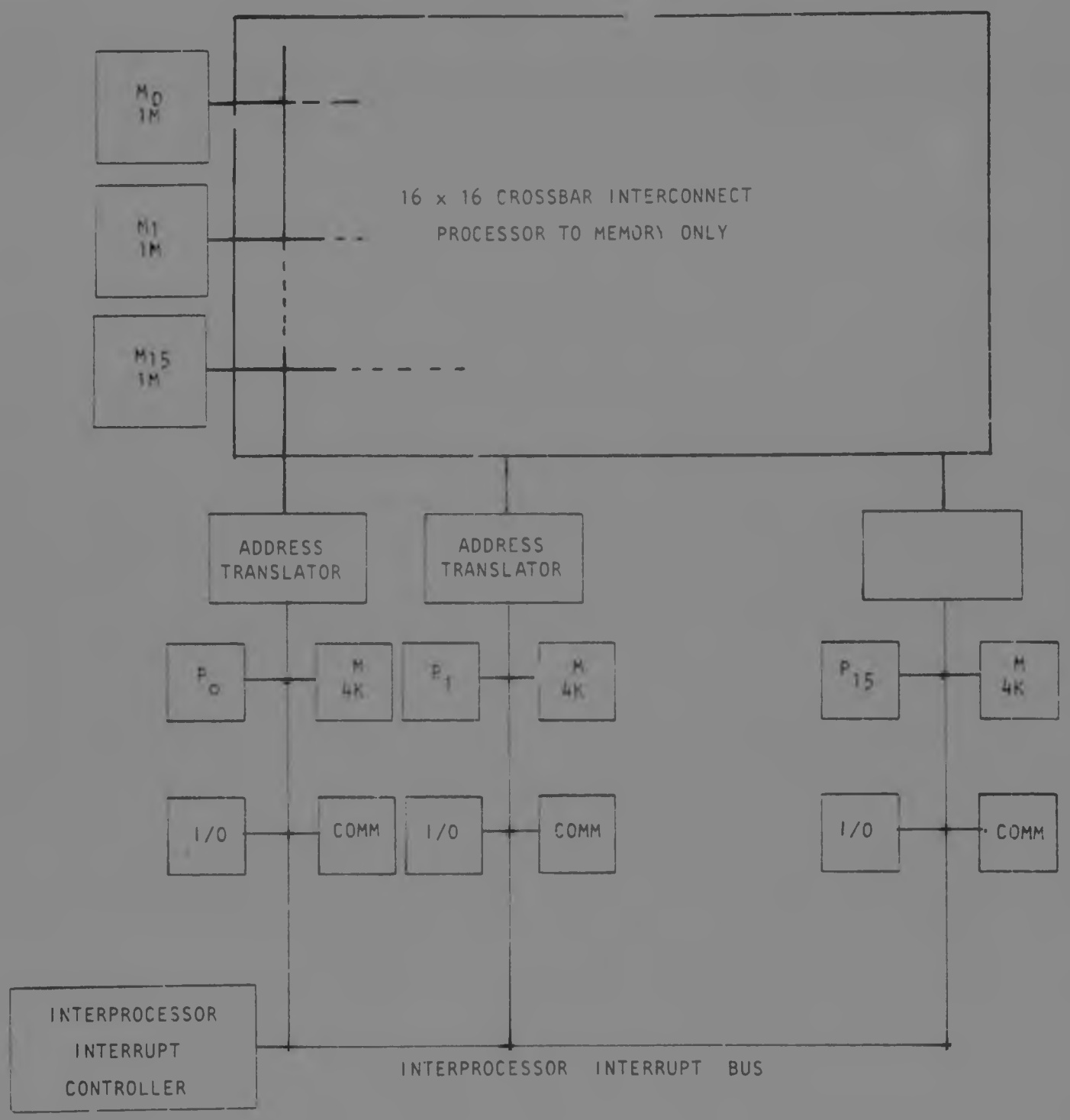


FIGURE D-4 Cmmp.

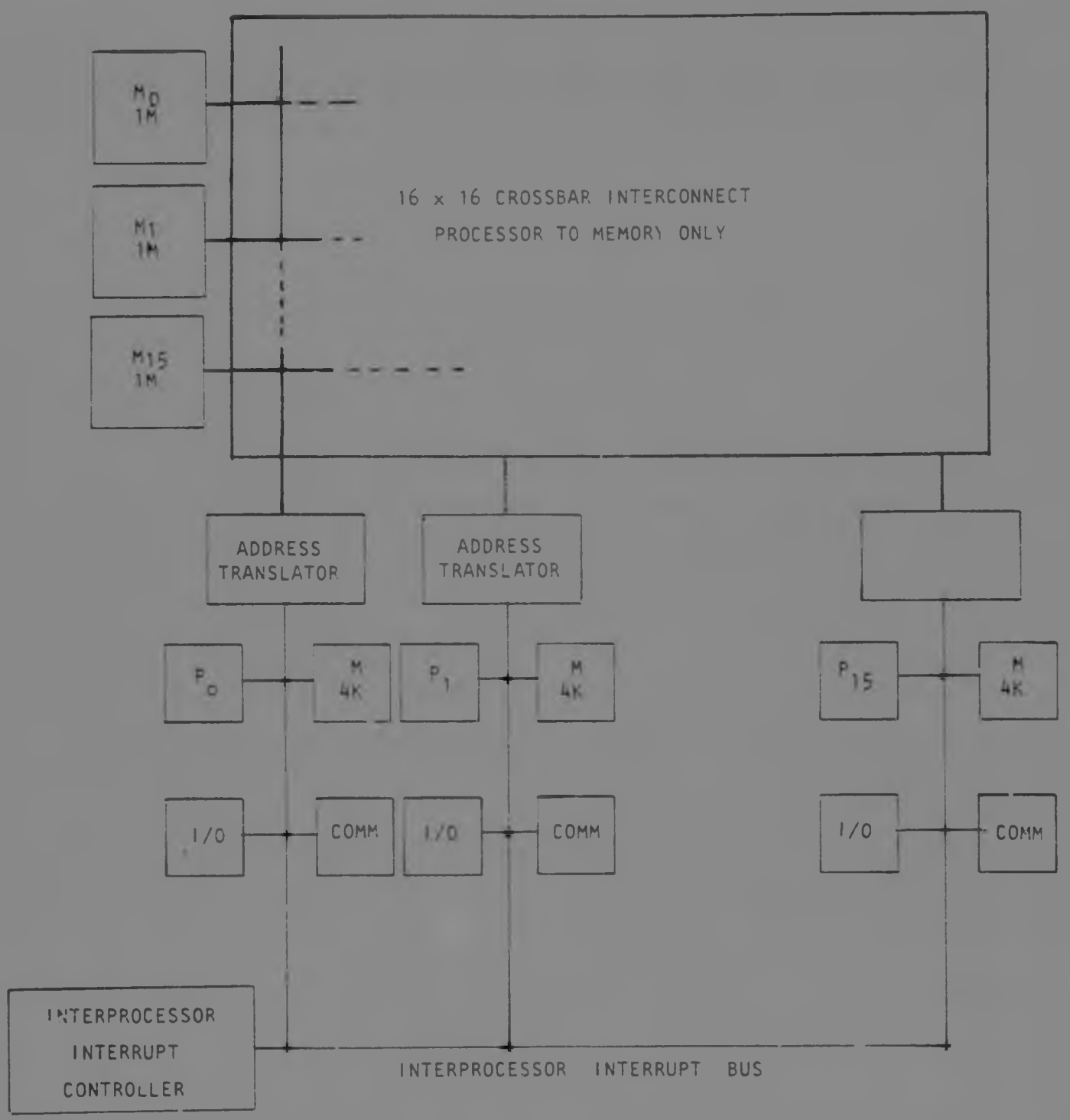


FIGURE D-4 Cmmp.

The disadvantages are :

The crossbar switch is complex and expensive and the address translator has to be able to resolve memory conflicts. Although there is no main processor the system is not fault tolerant, as a task on a failed processor module cannot be re-allocated.

D.5 The Banyan Multi-microcomputer System (BMS)

The BMS is composed of 15 Z8001 processors interconnected with 15 memory segments by a 4x4 crossbar switch. The interconnection is fully parallel, unidirectional and is packet switched. Overall control resides in a Vax 11/780 which accesses the rest of the system, via a Unibus adapter, using I/O transactions.

The BMS has the disadvantage of a complex crossbar switch. In addition there are local interfaces (I/Fn) to provide communication between the crossbar switch (SN) to the processors (Pn) or the memories (SMn) (see figure D-5).

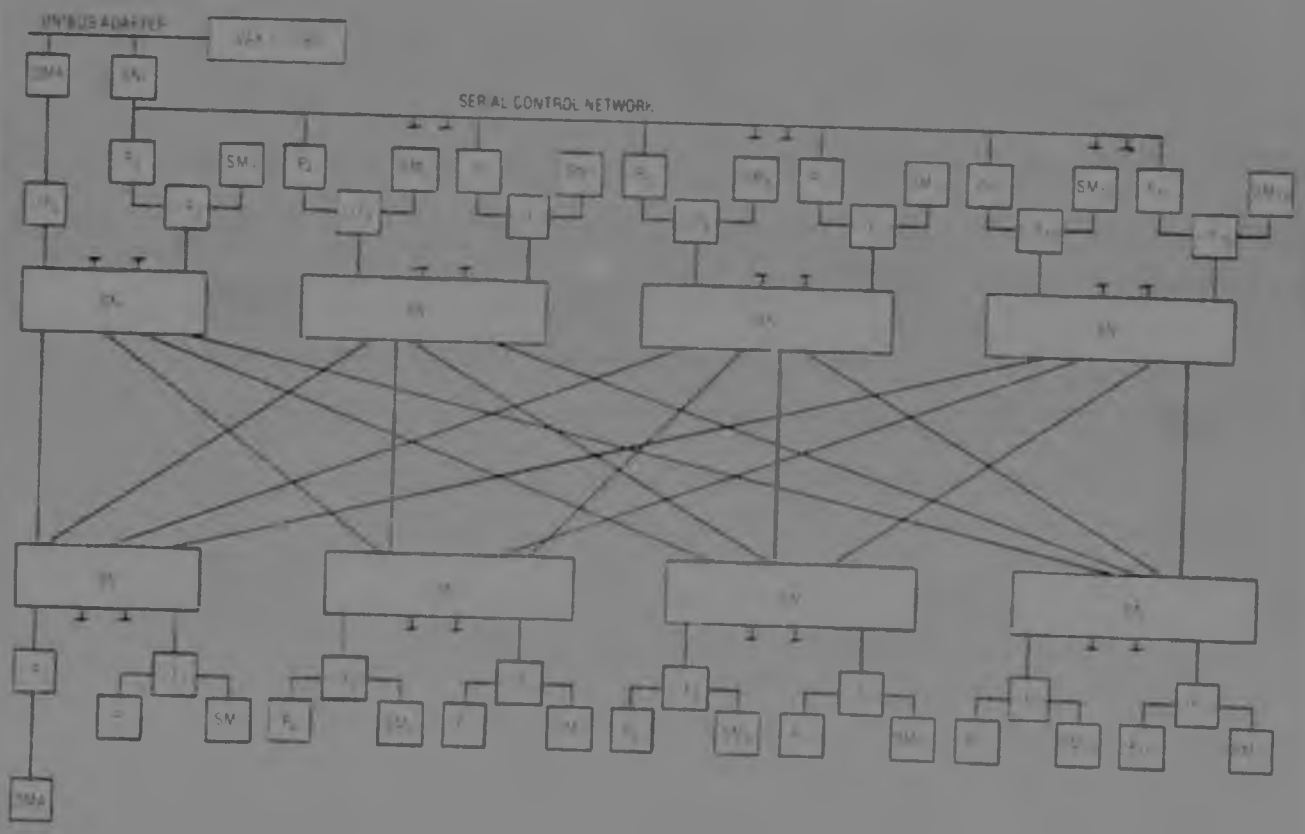


FIGURE D-5 THE BANYAN MULTI-MICRCOMPUTER SYSTEM [McD]

D.6 INTEL iAPX 432 Multiprocessor System

The iAPX 432 is a 32 bit microprocessor which has an ADA compiler. It comprises of two chips forming a General Data Processor (GDP). It has been designed for multiuser applications and offers the user transparent multiprocessing, i.e. the number of GDP's can be increased or decreased without the software having to be rewritten.

The designer is free to choose his own bus structure and the 432 uses a standard interconnection protocol. Input/Output is achieved through the Interface Processor (IP) which programs a group of programmable associative memories (window registers) to map the I/O subsystem's address space.

The 432 uses virtual addressing such that only 7% of microprogram space is used. The 432 can operate in two modes: In the master mode a component operates normally whilst in the checker mode the output pins reverse themselves and operate as special input pins. These pins sample data and compare this data to the data that would have been sent if the chip was operating in the master mode. Thus a highly fault-sensitive system can be built.

Instructions can vary in length from zero to three operands, and can thus support scalar, vector and record data types, such as found in ADA. There are no registers and memory and a hardware supported special stack are used for operands.

The architecture is object-orientated, and the objects provide an identical framework from simple bytes till messages that are sent to another processor. Objects are stored in segments of the address space, and they are always addressed via an object descriptor which contains information pertaining to the type and location. An access descriptor indicates the location of the object descriptor which is the only way to address an object. Thus the 432 has a two level operation for memory requests.

The 432 has a hardware operating system and can handle complex software applications and has many software protection mechanisms and has an extensive hardware fault detection mechanism. Thus it is very powerful and offers the computer architect an ideal basis for developing a real-time multiprocessing system (figure D-6).

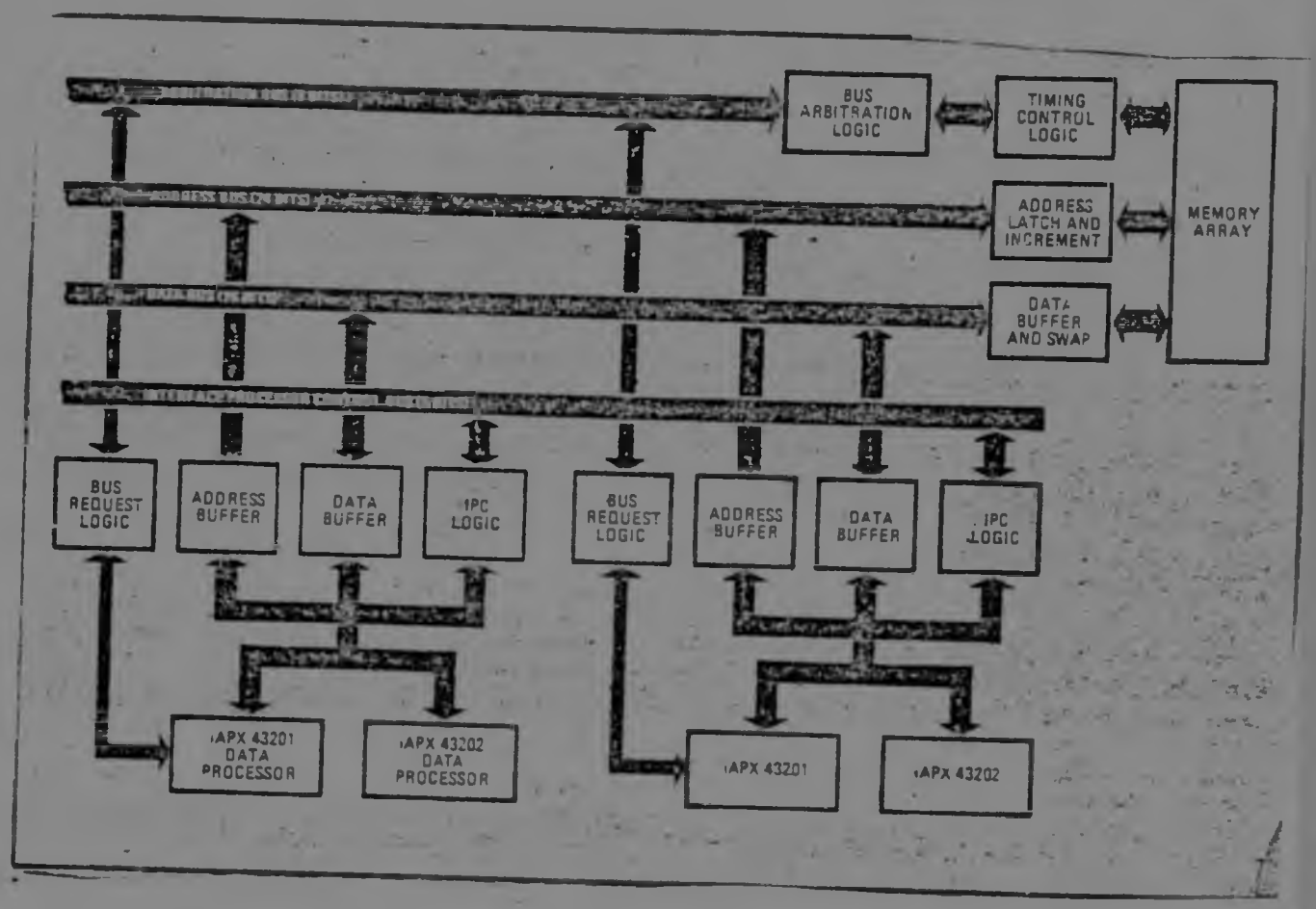


FIGURE D-6 THE INTEL 432 SYSTEM [RAT]

APPENDIX E

INPUT/OUTPUT INTERFACING

The requirements of the input/output modules of Ramrod are listed , and a brief introduction to Ethernet is discussed, with the view to using Ethernet as a communication medium on the I/O side of Ramrod.

E.1 Requirements

The input/output section of the multiprocessor system is required to handle communications between peripherals and processors on the one hand and between processor and processor on the other hand. Therefore the I/O bus must have a high degree of intelligence.

The I/O bus must have the same facilities as the TDM commor memory bus discussed earlier, that is if a processor fails then another processor must be able to 'hook' onto the now vacant peripheral. Processors must be transparent to other and to the peripherals, and must be able to communicate with any device that is connected to the bus.

Another important feature required from the intelligent I/O bus is that there be no master controller of the bus, is that if the controller fails another device can become the controller. This increases reliability and provides for redundancy.

In order to implement inter-task communication on the I/0 bus the message's destination will probably be another task's identity , and the bus will have to be clever enough to determine which processor is executing this task.

The interface to the bus needs to be modular and relatively simple so that it can fit onto one printed circuit board similar to the TTL/ECL interface boards, and it should be bidirectional. If an interface board is removed the system should not be affected, and at least 50 processors and 50 peripherals must be able to be connected to the system.

In addition the software overhead for protocols which control the information transfer between transmitting and receiving devices must not be too high.

E.2 Ethernet

E.2.1 INTRODUCTION

A project involving the design of an Ethernet Controller was undertaken by S.A. Eliasov as an MSc project in the Dept. of Elec. Eng. at the Univ. of the Witwatersrand, with the idea of incorporating Ethernet on the Input/Output side of Ramrod.

Ethernet is a local area network which evolved out of the Aloha network at the University of Hawaii. Studies of the Aloha network revealed a number of problems and refinements were undertaken at the Xerox Paulo Alto Research Centre in the mid 1970's.

E.2.2 The Topology of an Ethernet Network

E.2.2.1 Network Configuration -

The maximum network configuration is as follows:

1. A coaxial cable, terminated in its characteristic impedance at each end, constitutes a cable segment. A segment may contain a maximum of 500 meters of coaxial cable.

2. A maximum of 100 station transceiver connections may be made per segment.

3. Segments can be joined together using repeaters, provided that the longest path between any two transceivers is less than 1500 meters, and that there are no more than 2 repeaters in the path between any two stations.

4. Repeaters do not have to be located at the ends of segments, nor is the user limited to one repeater per segment. In fact, repeaters can be used not only to extend the length of the channel, but to extend the topology from one to three-dimensional.

Every station on the Ethernet Network is connected to the coaxial medium via an ethernet controller. The controller is joined to A transceiver, which is fixed on to the coaxial cable by a transceiver cable, consisting of six shielded twisted pairs not more than 50 meters in length.

E.2.3 Message exchanging in Ethernet

E.2.3.1 The Transmitting Station -

Before broadcasting, the transmitting station must ensure that no other station is busy using the medium. This is acheived by "carrier sensing" whereby the transmitter of a station is prevented from becoming active until all transitions on the coaxial cable have ceased.

As there is nothing to prevent two or more stations from scheduling a transmission for the same message slot, collisions will occur. Due to the ability of a station to "carrier sense", collisions will only occur at the start of a messages. The time interval during which collisions can occur is called the "collision window", which is long enough to allow for signals to propagate throughout the medium.

When a collision does occur, the transmitting station must stop transmitting its message and start transmitting a "jam". A "jam" is a burst of noise that ensures that all nodes will detect that a collision has taken place. After sending the jam, the station controller will enter a binary exponential backoff algorithm to randomise the re-scheduling of the transmission. In order to take into account increased traffic during busy periods, the backoff algorithm increases its mean value exponentially with the number of

collisions of the message.

E.2.3.2 The Receiving Stations -

The receiver must continually monitor the line to detect any broadcasts. Message packets are broadcast randomly over the medium. In order for the receiver to extract the data from the information stream, a synchronization burst must precede the transmission.

All messages must be examined to determine their destination address. Each station on the Ethernet can be addressed in the following ways:

1. Physical Address : A unique address associated with the station, and distinct from the address of any other station on any Etnernet.

2. Multicast Address : An address that can be setup under software control that will be accepted. This means that more than one station can use the same address.

3. Broadcast Address : This address is accepted by all stations on any Ethernet system. It can be used by a station when it is connected on to the network to indicate that it has become an active station.

Once a message has been accepted by the receiver, it must first perform an error check to determine if there were any transmission errors, before handing the message packet to the host processor.

E.2.4 Comparison of Ethernet

In Token Bus [RAV] nodes are connected to a common bus in a virtual ring. In order to transmit a node must be in possession of the 'token', and therefore the method of access is highly organised and there is an absence of collisions. However there is a possibility that a faulty node could create a duplicate token or that the token could get lost. This means that extra logic is needed to prevent these posibilities.

Ring network [RAV]on the other hand interconnects nodes in a loop with messages travelling around the loop in one direction. Access is deterministic and priorities can be assigned thereby preventing collisions. However as each node acts as a repeater , the reliability of the network depends on the reliability of a single node. The removal of a node from the network can result in messages circulating indefinitely.

Ethernet has a major disadvantage in that as the loading becomes heavy collisions increase and the channel utilisation decreases.

E.2.5 Summary

Ethernet ,a bit serial communication medium, can operate upto 10 Megabits per second . A typical packet has a 64 bit preamble, 48 bit destination and source address, 16 bit data type word, 368 to 12000 bits of data, 32 bit Cyclic Redundancy Check and a 96 bit packet gap.[CRA] Thus an information packet can range from 672 to 12304 bits.

Ethernet , which consists of coaxial bus segments, can be expanded passively by adding transeivers and coaxial cable. If needed signal strength can be buffered by connecting a simple packet repeater.

E.2.6 Protocols

Transfering information packets from one device to another requires methods for error correction, flow control, process naming, security and accounting. These methods are usually termed protocol. Ethernet has a simple error controlling packet protocol, called Ethernet File Transfer Protocol (ETFP), which is implemented in the interface to Ethernet.

E.3 Conclusion

Ethernet fulfills all the above criterea and is therefore the most suitable bus communication medium. However as the hardware is not so readily available the actual design of the Ethernet bus is being designed in a related project and until then the I/O bus will have dedicated processors for each peripheral.

It should be noted that until recently Ethernet implementations were not commercially available. Intel has announced their NDS-11 network development system [HUG].

APPENDIX F

THE EXOSLICE DEVELOPMENT SYSTEM

An introduction to the Motorola Exorciser Development system and the FAST package, which allows a user to emulate and design his bit-slice hardware, is described below.

The EXOslice bit slice development system has been designed to be run on the M6800 EXORciser microprocessor development system. It allows the user's slice system to be slaved to the EXORciser via Peripheral Interface Adapter(PIA) cards.

The Flexible Aid for Sliced-processor Test(FAST) monitor allows the designer to develope and debug programs for use in his hardware.

FAST used in conjunction with the Motorola Diskette Operating System(MDOS) can be operated in a floppy disk environment.

F.1 The Exorciser

The M6800 EXORciser is a system development tool used in the design and development of M6800 Microprocessor systems. Basically the EXORciser assists the system designer by allowing debugging of software and hardware emulation.

Once the EXORciser has been loaded the user can look at the contents of memory and perform the Motorola Active Interface(MAID) functions as listed below.

MAID enables the user to;

i)Examine and change, if necessary, contents of a memory location or an MPU register.

ii)Execute a program

iii)single step the program or run until a previously inserted breakpoint is encountered.

iv)Perform decimal-octal-hexadecimal conversions as well as calculate offsets for the relative addressing mode.

F.2 MDOS

The M6800 Diskette Operating System(MDOS) enables the user to develope his software easily on the EXORciser. It is an interactive operating system that interprets commands from the operator's console.

The user can store or retrieve data, in the form of files, on a diskette,process this data or activate other user commands from the diskette. There are various system commands that allow the user for example to initiate and format diskettes and check them for errors. Command chaining can be achieved by storing commands in a special command file and then invoking this file. MAID is entered

once an object file has been loaded into the memory space so that the program can be executed.

Files can be edited either by using the Co-Resident Editor or the updated version EDIT1. The EDIT1 editor automatically assigns line numbers to each file line, but otherwise is faster and more efficient than the former editor.

F.3 MASM

The Macro Assembler (MASM) has been designed for microprogrammed bit slice processor developement.

The user must first of all define his microword size and then the mnenomics and the format of the microword in the DEFINITION PHASE, which reads a definition source file and creates an assembly source file. The definition allows for implicit or explicit field lengths. Overlapping fields can be achieved by using 'dont care' fields.

Once a program has been written using the assembly language defined in the previous phase it can assembled during the ASSEMBLY PHASE.

once an object file has been loaded into the memory space so that the program can be executed.

Files can be edited either by using the Co-Resident Editor or the updated version EDIT1. The EDIT1 editor automatically assigns line numbers to each file line, but otherwise is faster and more efficient than the former editor.

F.3 MASM

The Macro Assembler (MASM) has been designed for microprogrammed bit slice processor developement.

The user must first of all define his microword size and then the mnenomics and the format of the microword in the DEFINITION PHASE, which reads a definition source file and creates an assembly source file. The definition allows for implicit or explicit field lengths. Overlapping fields can be achieved by using 'dont care' fields.

Once a program has been written using the assembly language defined in the previous phase it can assembled during the ASSEMBLY PHASE.

When the program has been successfully assembled then the resulting object file can be merged with another system file to allow it to be loaded during the execution of FAST.

A disadvantage of the macro assembler is that the user must actually list the whole microword even though he may not wish to use all the fields.The number of fields are limited and therefore a long microword with too many fields will have to have some of its fields joined together.

F.4 EXOSLICE

Exoslice has been designed to extend the EXORCISER'S emulating capability.Once the program has successfully been assembled the user's bit- slice hardware can be directly coupled to the main system. This is achieved by using the Flexible Aid for Slice Testing (FAST) program.

The EXOslice subsystem is capable of being connected to the ECL 10800 bit-slice family or the 2900 bit-slice family.

The subsystem is made up of the following components: (a) Input/Output modules which feature 32 ECL output lines, 16 ECL input lines and 4 ECL output control lines. These can be expanded to 5 modules thus allowing a 160 bit word length. The I/O module has 3 Peripheral Inteface Adaptors (PIA) thus allowing the EXORCISER to read and write words

greater than the 6800's 8 bit word. A decoding Programmable Read Only Memory (PROM) allows the FAST software to consecutively address all output lines followed by all input lines.

(b) In order to interface to a TTL 2900 series bit slice system an ECL to TTL module is provided for each I/O module.

FAST generates control signals from the EXORCISER in order to allow the user's bit slice system to be slaved to the EXORCISER. The user's microprogram storage is then effectively replaced by the main system's Read/Write storage. The EXOR Clk signal enables the user's system e single stepped.

FAST can also be used without previously using the macro assembler. Definition can be achieved during the running of the FAST program and instructions can be loaded, examined, changed, inserted or deleted as in any other available emulator.

While debugging the program FAST emits a clock pulse each time a new microword is put out, and the special reset command produces 5 clock pulses.

Figure F-1 shows the functional steps during a micro-instruction execution. The line table is a buffer which temporarily stores all data going to or coming from the hardware interface.

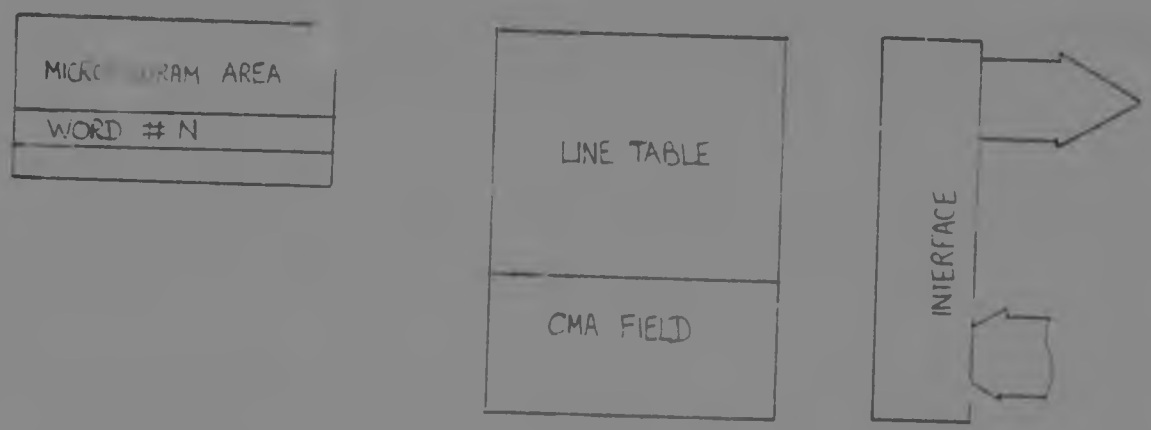


FIGURE F-1 MICROINSTRUCTION EXECUTION STEPS

Similarly to MAID FAST enables the user to insert, display or remove breakpoints for subsequent program running. The program can be executed step by step or free run. User's data can be manipulated as files from the diskette nd thus previously saved or assembled programs can be loaded directly while operating in FAST.

Once a word has been successfully defined a hardware configuration list can be obtained .

FAST unfortunately has a maximum of 11 fields and the designer must keep this in mind when designing his microword.

In the DEBG or MPGM modes the format of the micro word is hexadecimal by word and vica versa. A far better system would be to divide the word into fields defined by the user and allow him to use the hexadecimal format for each field. This would decrease debugging time considerably.

APPENDI G

THE CIRCUITRY

G.1 The Microprocessor Module

A timer is gated into the RST pin of the 8085 processor so that the processor is reset after a power up sequence. This can also be done manually by a RESET button or by the Master Controller.

A monostable and D type latch form the basis of the 'watch dog' alarm. When a trigger, in this case a read common memory, is not received by the alarm, the processor is held by the READY signal and the MC is notified and an LED is lit.

The latch control signals are triggered by a master processor pulse which enables the processor to latch its address and then its data into the latches(write cycle). A read is accomplished by latching the address and holding the processor until the next cycle when the data returns.(figure G-1)

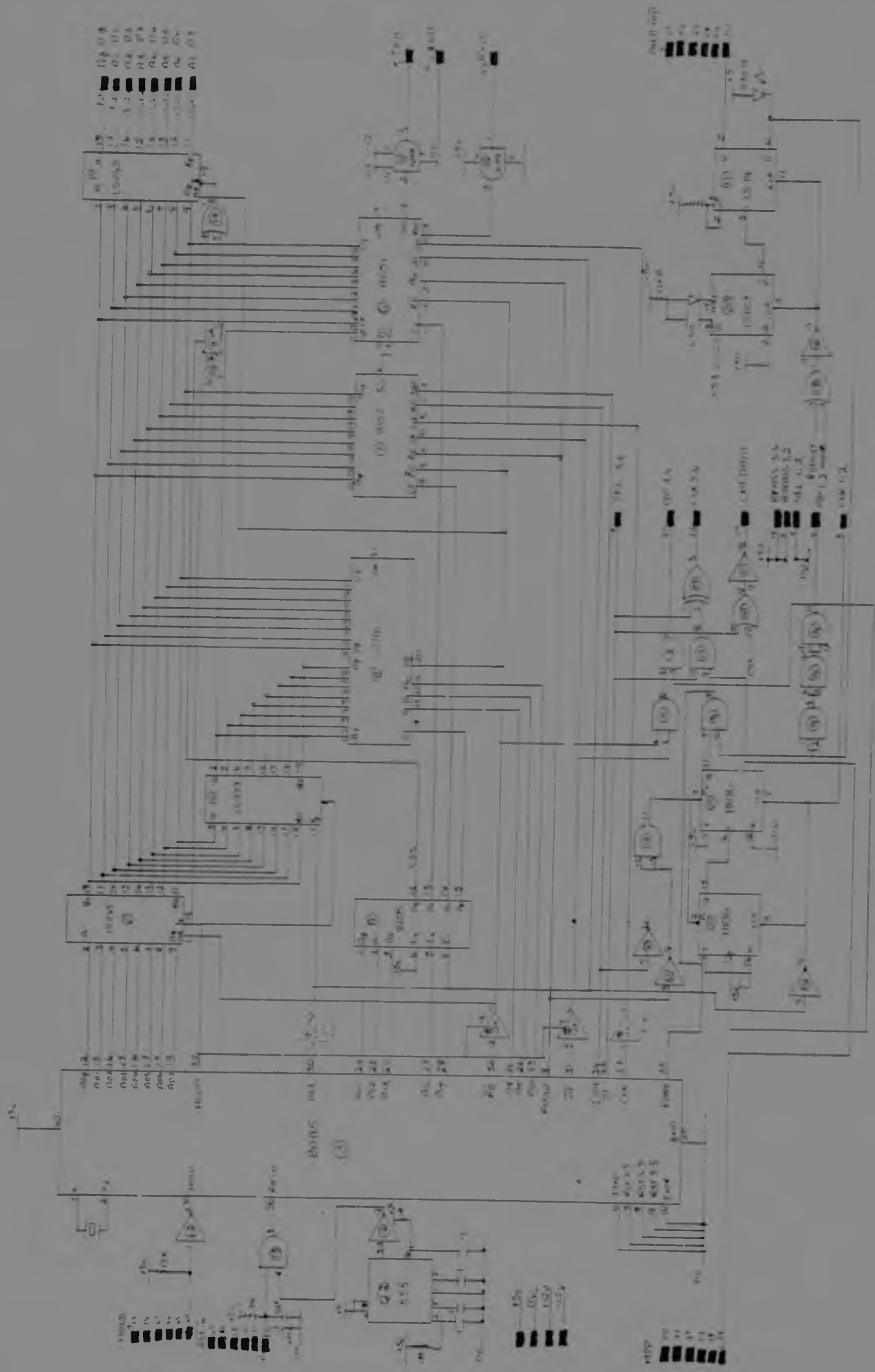


FIGURE G-1 MICRP'OCESSOR MODULE

G.2 The ECL latch Module

This module consists of Bidirectional Translating latches which are controlled by ECL signal translated by a TTL to ECL tranlator. There are termination resistors on every point of access to the ECL bus so that the bus is terminated at every output (figure G-2).

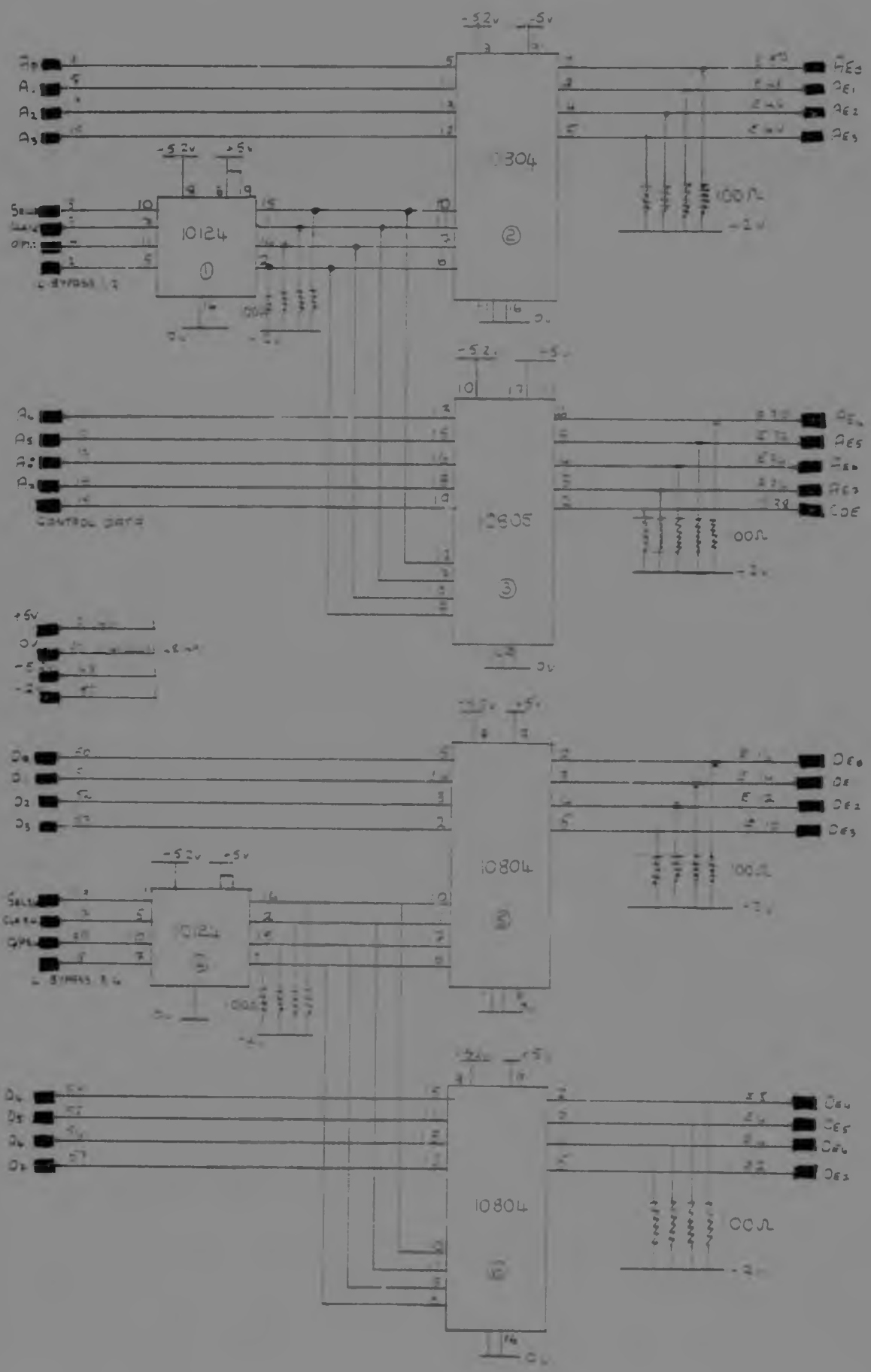


FIGURE G-2 ECL LATCH MODULE

G.3 The Memory module

A read/write signal is generated from the R/W signal received the ECL bus by the monostables and a JK flip flop. This is done in order to generate the correct width pulses for controlling the latches on the memory side.(figure G-3)

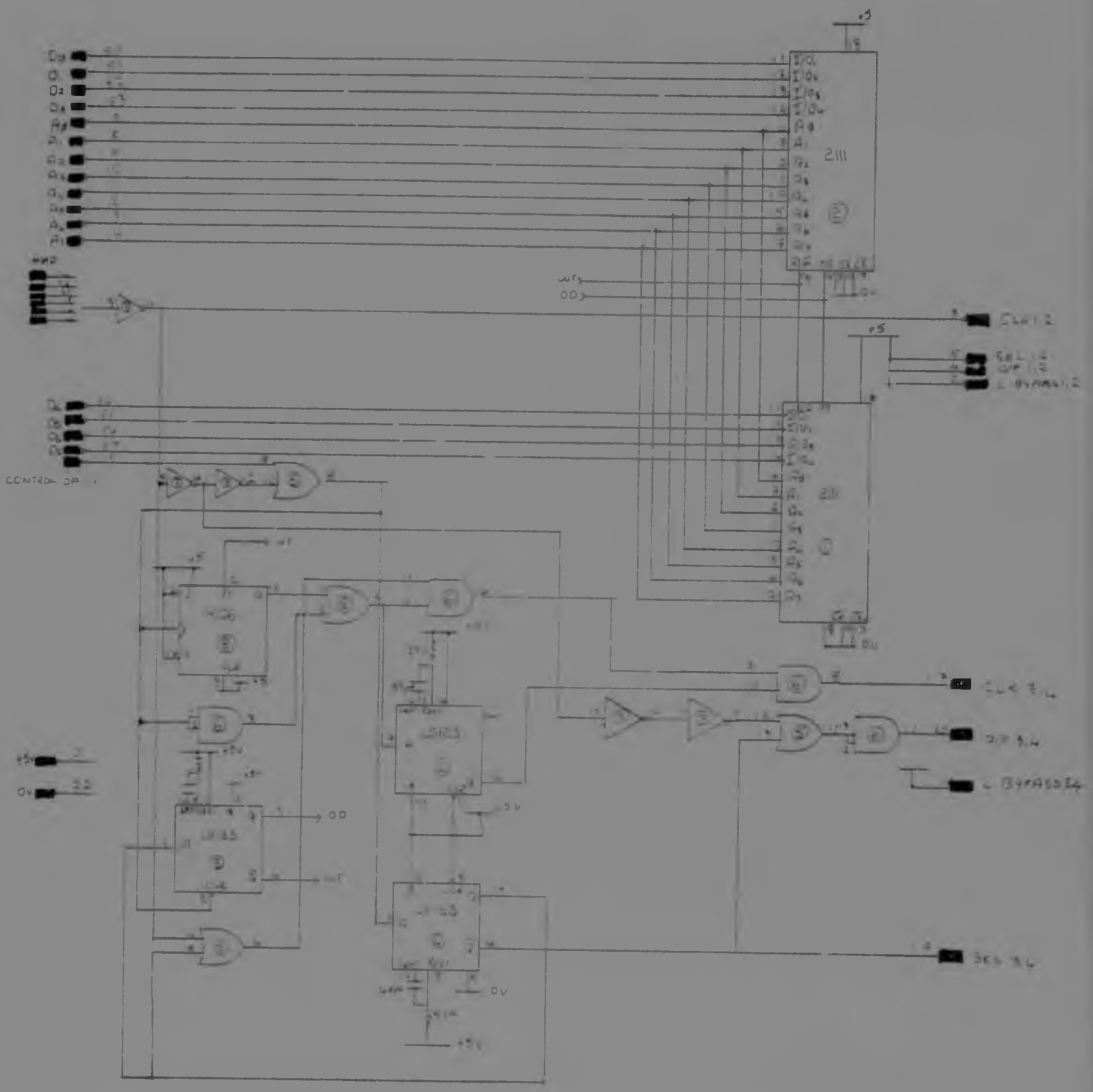


FIGURE G-3 MEMORY MODULE

G.4 The Control Board

A modulo 5 counter, driven by an external clock , accesses two identical sets of fast memory. The MC can write to these memories the data desired and then the counter reads successive locations. The output of the memories are the master memory pulses and the master processor pulses. It should be noted that any combination of pulses can be obtained. (figure G-4)

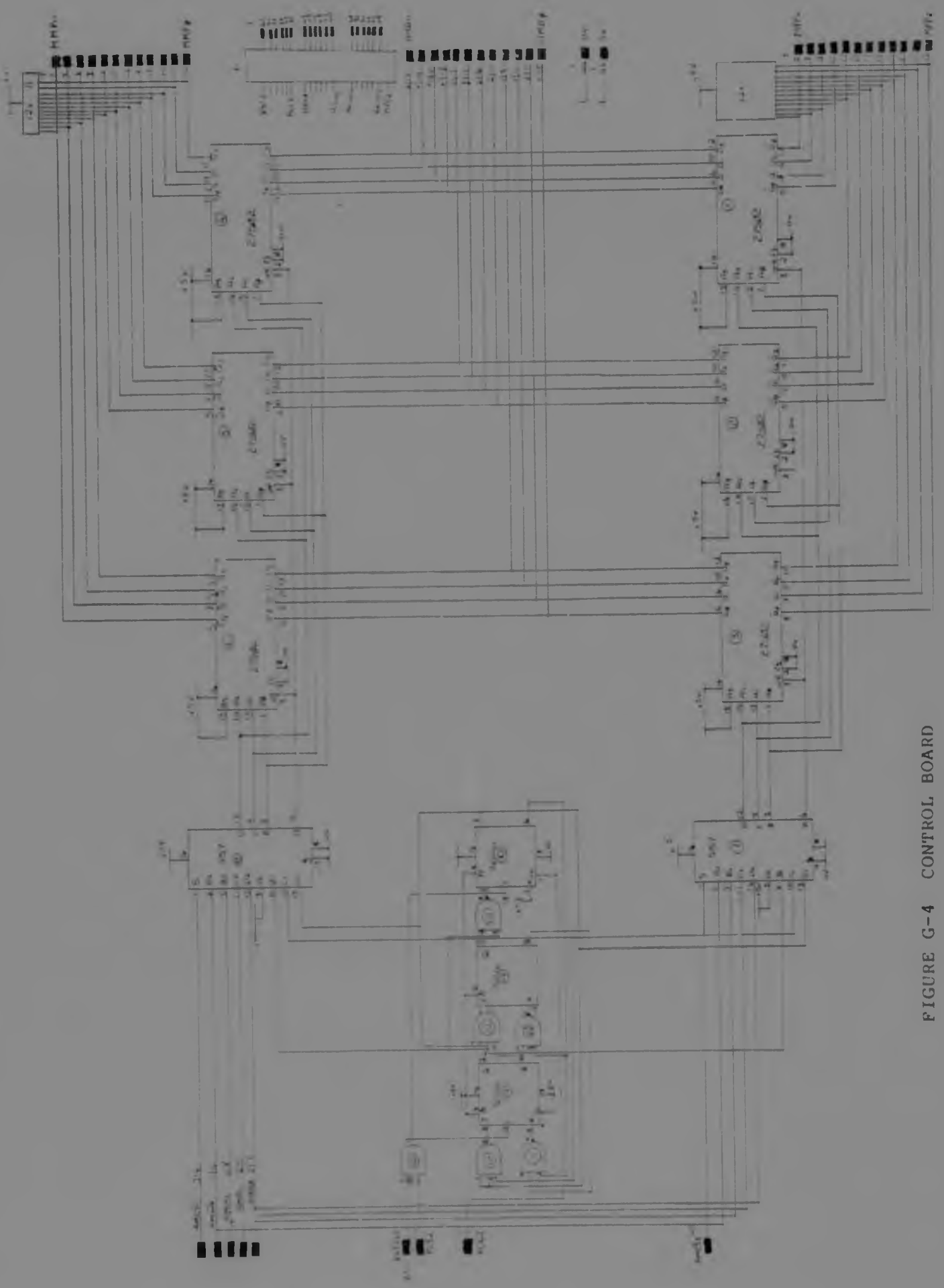


FIGURE G-4 CONTROL BOARD

G.5 The Central Processor Array

Four 4 bit slices are joined to form a 16 bit ALU. The status and fast look ahead units are included to speed up computations. A multiplexer enables either data from the local memory or from the pipeline registers. (figure G-5(a))

One way latches enable the ALU to communicate with local memory, common memory and other I/O.

Real time execution is enabled via a multiplexer or alternatively the EXORCISER provides all the neccessary control signals. (figure G-5(b))

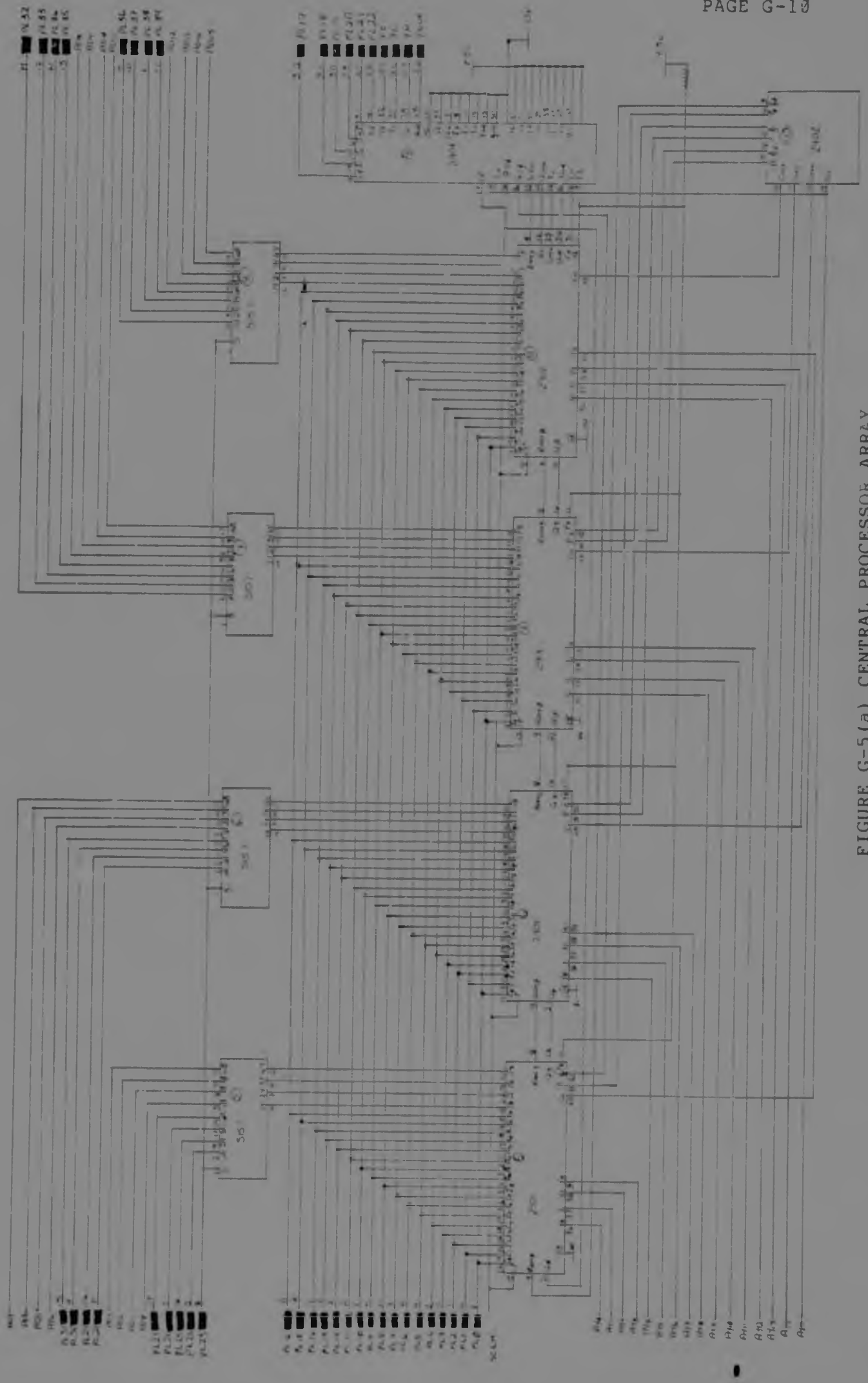


FIGURE G-5(a) CENTRAL PROCESSOR ARRAY

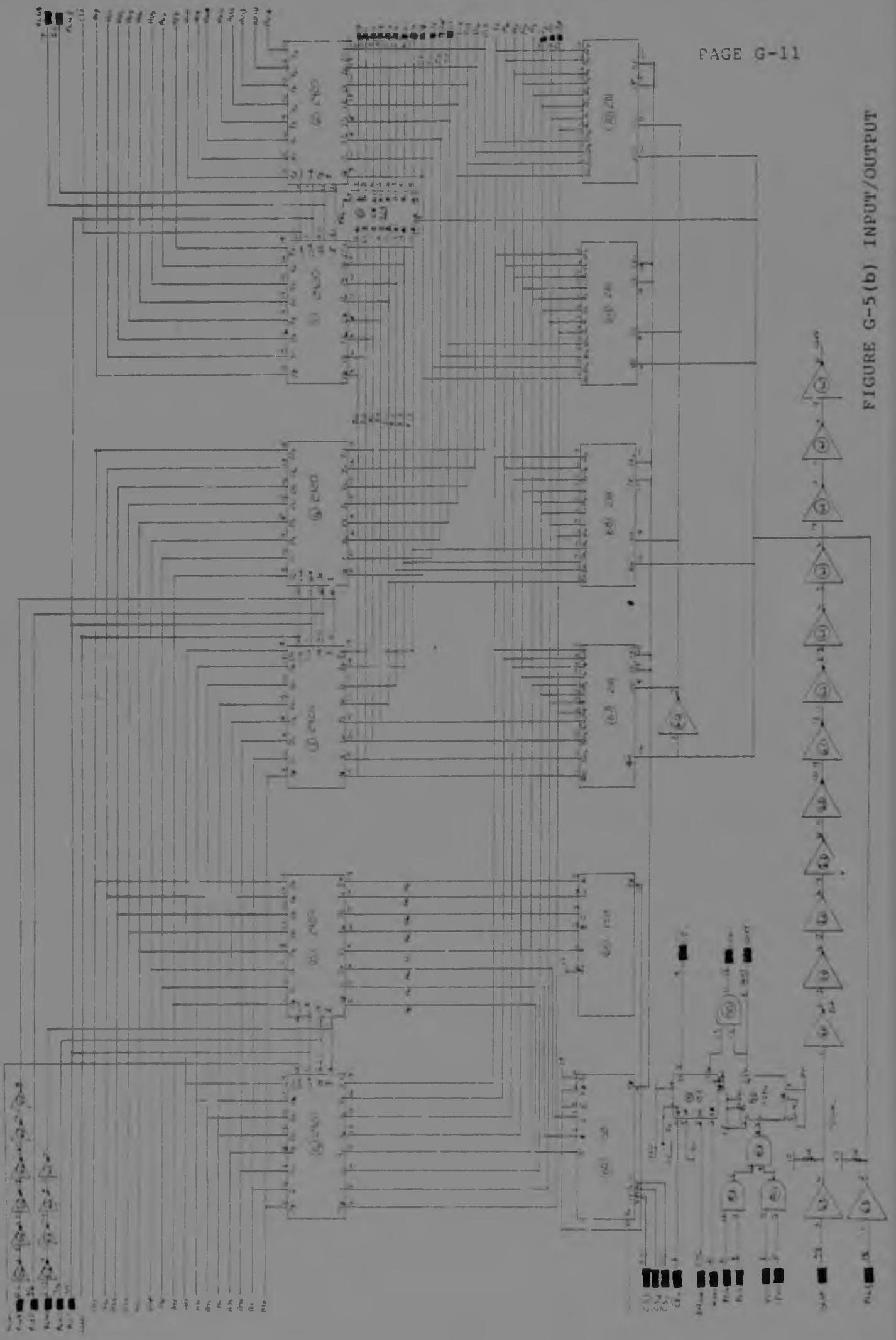


FIGURE G-5(b) INPUT/OUTPUT

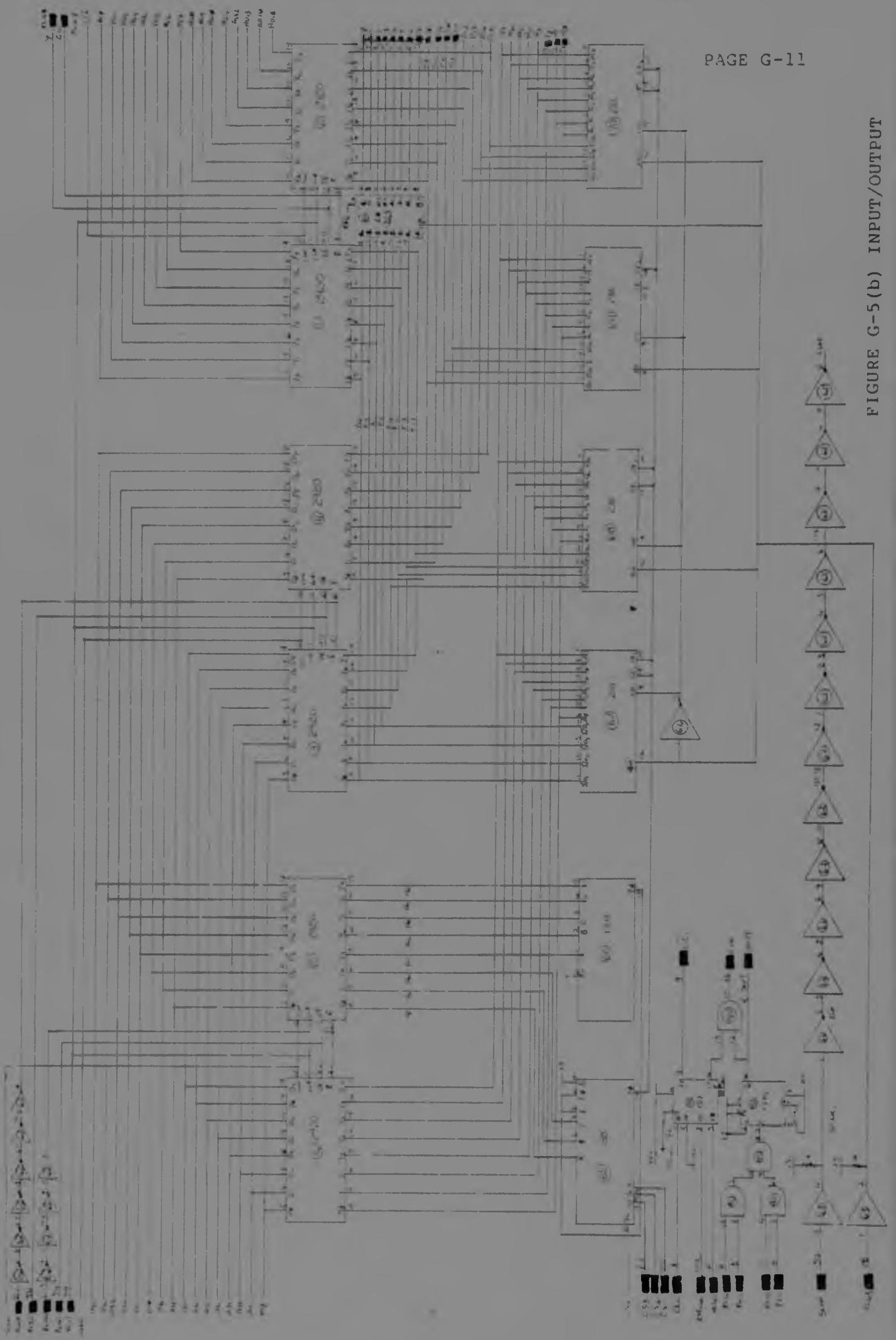


FIGURE G-5(b) INPUT/OUTPUT

G.6 The Computer Control Unit

Three 4 bit sequencers give 2**12=4K by 64 locations in the control store for the microinstructions. The next address unit enables the sequencers to function more efficiently,by adding extra codes. A vector input to the sequencer can be obtained from the interrupt unit circuitry, or directly from the pipeline.

The interrupt unit can recognise an interrupt from each of the processor modules. The interrupts however have to be correctly pulsed by a set of monostables.

The next address unit also controls a 12 bit counter which produces a signal once a preset condition has occurred. This signal as well as status flags are routed through a multiplexer to be tested, with polarity, by the next address unit. The interface to the processor common memory is derived in a similar fashion to that of the processor modules.(figure G-6(a))

The pipeline registers are one way latches which collect their data from very fas RAM (control store). This control store is replaced by the EXORCISER during operation of FAST, but the RAM can be loaded from the EXORCISER for real time processing . The control signals for the rest of the system are derived from the pipeline registers.(figure G-6(b))

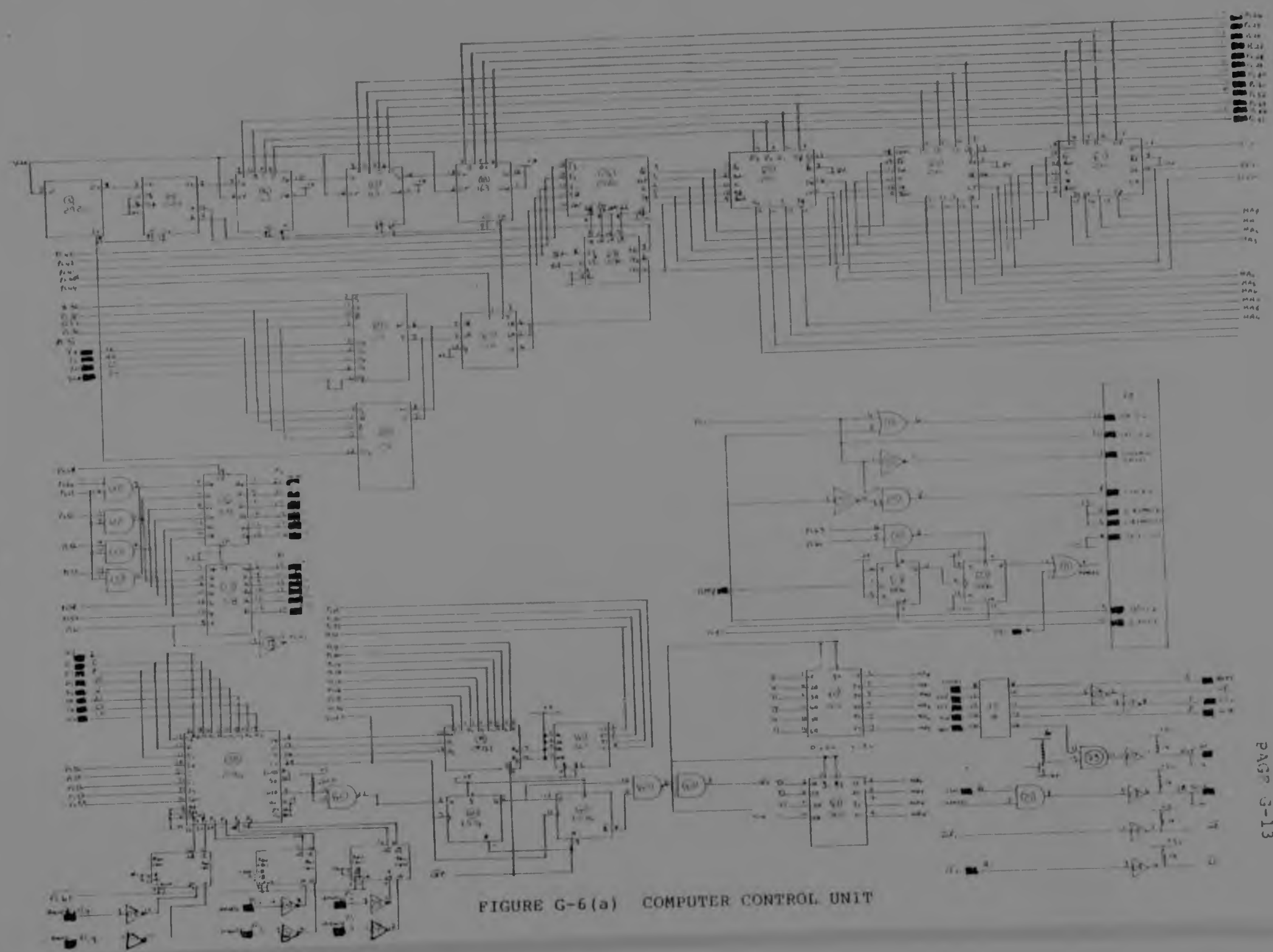


FIGURE G-6(a) COMPUTER CONTROL UNIT

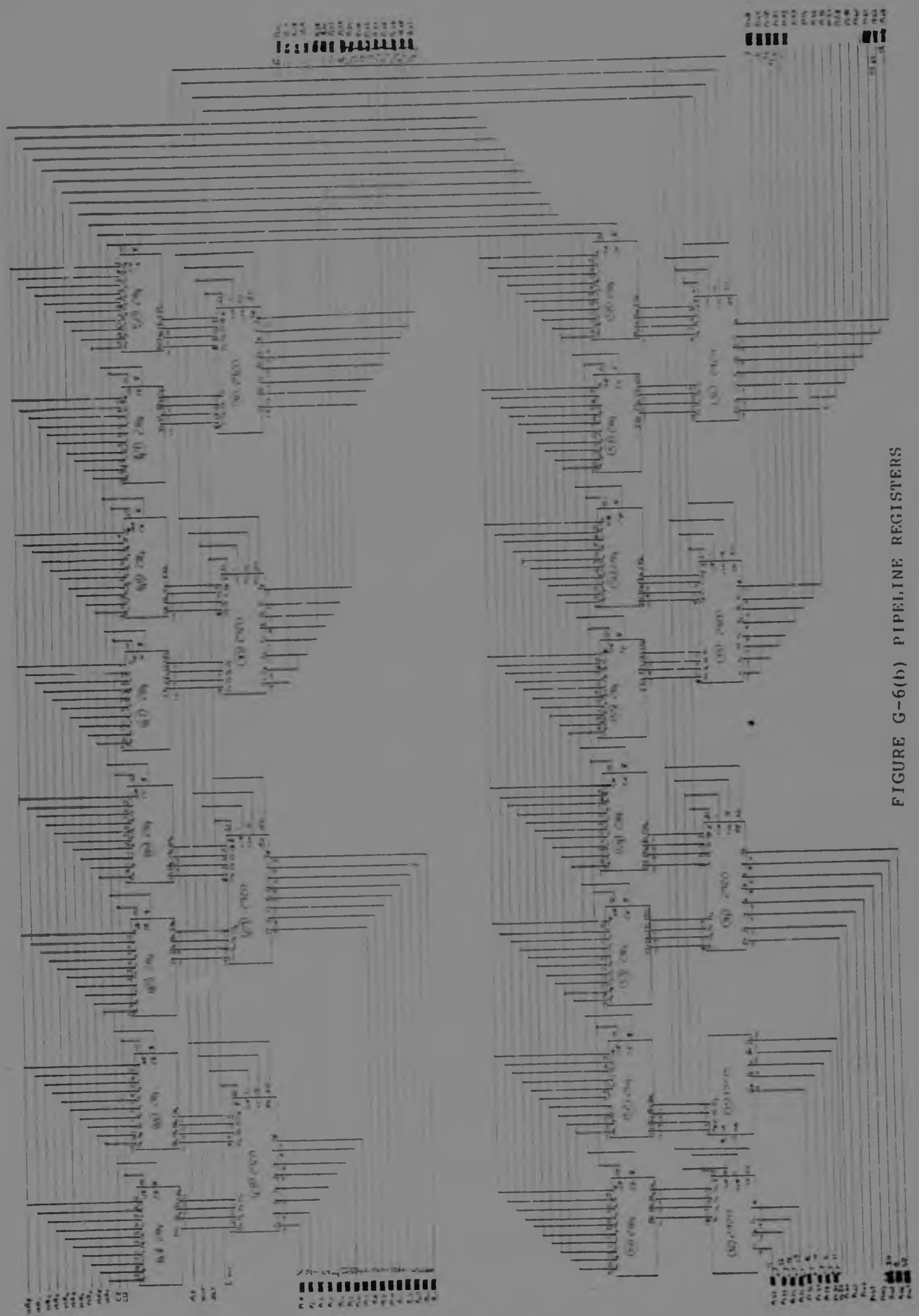


FIGURE G-6(b) PIPELINE REGISTERS

APPENDIX H

MARKETING COSTS OF RAMROD

The cost of marketing a product can be basically divided into two areas:

1. Development cost

2. Production cost

H.1 Development Costs

The development costs of the complete Ramrod system must take into account the following:

1. Microcoding @R100/4 lines - R3000

2. Other software - R3000

3. Research and development @R250/day for 3 man years - R320,000

4. Equipment such as Logic Analysers, Oscilloscopes, Multimeters etc - R30,000

5. Emulation on a development system costing R50,000

Clearly the last three items are the most costly and dominate the development cost of a commercial product. They ,perhaps , overlap on the other costs . Thus the total development cost is in the region of R400,000.

H.2 Production Costs

Production costs include purchasing components for the complete system and the production of a single Ramrod system can be calculated from the following:

1. Printed circuit board layout for 4 boards - R1000
2. Printed circuit board manufacture - R500
3. Mechanical work and structure - R1000
4. Technical work (soldering etc) - R2000
5. Integrated Circuits - R1300
6. Printed Circuit Boards - R1200
7. Miscellaneous components (Fan etc) - R500

Thus it costs approximately R9000 to produce a single Ramrod system which consists of 5 slave processors, a master processor, 5 memory modules and 10 latch modules.

H.3 Marketing Cost

The selling cost of a marketable product is a function of the amortised development costs, production costs, normal application software, sales and support necessary to maintain the product.

APPENDIX I

HIGH LEVEL DESCRIPTION OF SOFTWARE

This section provides a High Level Description of the main routines used in the master controller and the local operating system. The description is based on a simplified form of Pascal.

I.1 MAIN ROUTINE

```
Read number of tasks to be loaded;
WHILE memsegment still free ;      search memory table
  Read in Tasks
  IF freeProcessor found THEN ;    search processor table
     BEGIN
       Dispatch task to processor
       Schedule processor to run
     END
UNTIL no tasks left.
```

I.1.1 INTERRUPT ROUTINE

```
Disable Interrupts
Read interrupt ID.
Call IntService (ID.)
Return to main routine.
```

IntService (ID.)

Enable Interrupts

Service interrupt

Return

Dispatch Task to Processor

Assign FreeMem to FreeProc; Program control board memory

Schedule Processor to Run

Assign Timeslot to Proc; Program control board memory

I.2 Local Operating System

```
START:  If Identity Equals slave
              then Execute user task
        ELSE BEGIN
                   set up Usart;
                   IF identity equals load;
                      THEN load tasks from disc;
                   ELSE BEGIN
                          Ask user for command
                          CASE of Command
                            1: display memory
                            2: Execute program
                            3: Test Common memory
                            4: Inform status of Ramrod
                            5: Insert data into memory
                            6: Move data
                            7: Substitute data
                            8: Display registers
                          END
                        END
                 END
        END
```

Ask User for Command

Read Command from Console

Call Command Routine

Display Memory

REPEAT

Read start address, end address

Display address, data

UNTIL end of address

Execute Program

Read start address

Put start address in Program Counter

Execute

Test Common Memory

REPEAT

Write random data into memory

Read data and compare

UNTIL end of memory

Inform User Status of System

DO 200 times

BEGIN

Read number of processors

Display data

Read number of memories

```
    Display data
    Read number of Tasks
    Display data
END
```

Insert Data

```
REPEAT
    Read address, data
    Write data into address
UNTIL End Of Command (EOC) Character
```

Move Data

```
REPEAT
    Read destination address
    Read end address
    Read source address
    Move data from source to destination
UNTIL end address
```

Substitute Memory

```
REPEAT
    Read address
    Read data
    write data into address
UNTIL EOC
```

Display Registers

REPEAT

    Write contents of register into memory

    Display "Reg", data

UNTIL no more registers.

APPENDIX J

RELIABILITY [SMI]

The reliability of a system is primarily influenced by its complexity. The fewer the parts and the fewer the types of materials and components involved then the greater is the probability of an inherently reliable product. In addition the use of redundant parts , whose individual failure does not cause the overall product to fail, is a common method to achieve a higher reliability.

It is good engineering practice to satisfy reliability requirements, but the engineer must bear in mind that the mathematical aspects of the subject, although important, serve only to refine requirements and do not themselves create a reliable product.

It is clear that the cost of making a system more reliable must be offset , in part , by a saving in maintenance to justify it. Maintainability and reliability , together, dictate the availability of the equipment, and are interdependant for the following reasons:

1. If the system's reliability is partly dependant on redundancy , it will be more reliable if the repair time (maintainability) of an SRU is improved. Thus maintainability can contribute directly to the reliability.

2. The design and assurance activities to achieve both of these parameters are , generally, the same.

3. The overall availability of the system, i.e. the `up time` is also dependant on both these parameters.

Availability is defined as the ratio of the up time to the total time. Up time is defined as the Mean Time Between Failure(MTBF) whereas total time is the sum of up time and `down time`. Smith [SMI] makes a distinction between down time and the Mean Time To Repair (MTTR) but for the purpose of this thesis they are considered the same.

$$\text{Thus } Av = \frac{MTBF}{MTBF+MTTR}$$

Availability is achieved by a combination of maintainability and reliability and there is a trade off between these two parameters as explained in the following example:

A system which has a MTBF of 100 hours and a MTTR of 101 hours has an Av= 100/101, has the same Av as a system with MTBF=200 hours and MTTR = 202 hours. Clearly the reliability of the former case is greater than that of the latter while the converse is true of the maintainability.

Reliability as mentioned above is influenced by the complexity of a system, and thus a uniprocessor system will probably be more reliable than a multiprocessor system. However if the factors of redundancy and repair are introduced the the multiprocessor becomes much more reliable.

The repair time is defined as the inverse of MTTR. Therefore if a redundant system is periodically repaired, whether or not faults are present, each time it is repaired the reliability calculations begin anew.

There follows calculations of the reliability and the MTBF of various systems including Ramrod .

Figure J.1 shows the reliability of a system consisting of several parts where the failure of any block causes a system failure (eg. a two board computer).

Thus $R = R_a . R_b$

Figure J.2 shows the situation where all blocks must fail in order to cause a total system failure (eg. a redundant processor system).

Thus $R = R_a + R_b - R_a . R_b$

Figure J.3 illustrates a situation which is composite of J.1 and J.2.

Thus $R = R_a.(R_a+R_b-R_a.R_b)$

The reliability diagram of Ramrod is illustrated in figure J.4. It will be analytically proven , and it can be seen from the diagram as well, that failure of either the master or the ECL bus causes a system failure. However it must be noted that the system will gracefully degrade to a uniprocessor computer if the master fails as mentioned in chapter 6, and unfortunately there is no way to show this in the mathematical model. Therefore the reliability of Ramrod as calculated is much worse than the actual reliability.

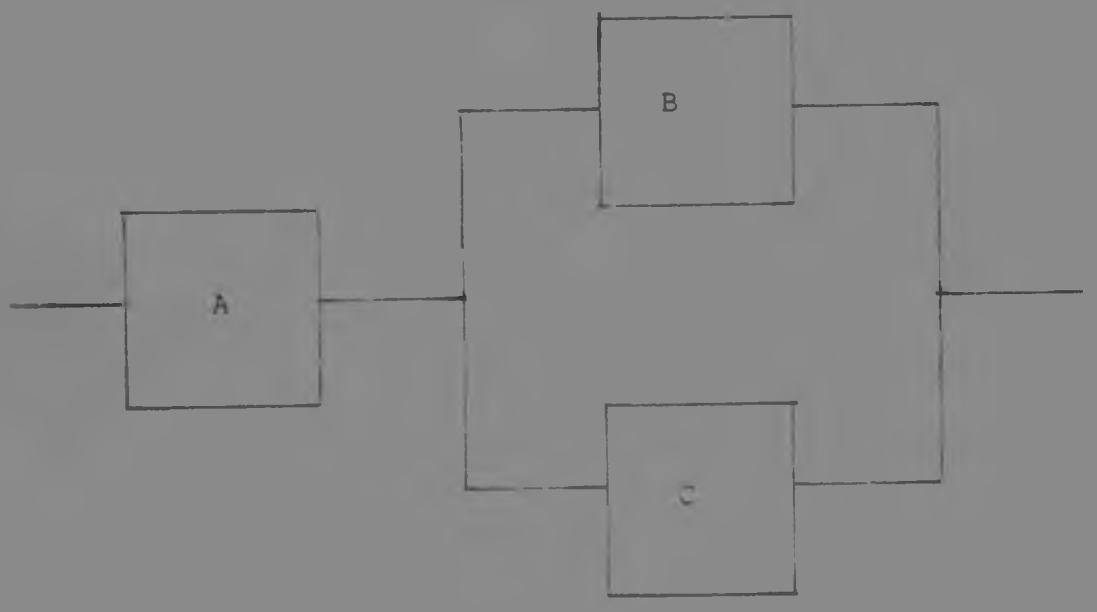


FIGURE J-3 COMPOSITE RELIABILITY

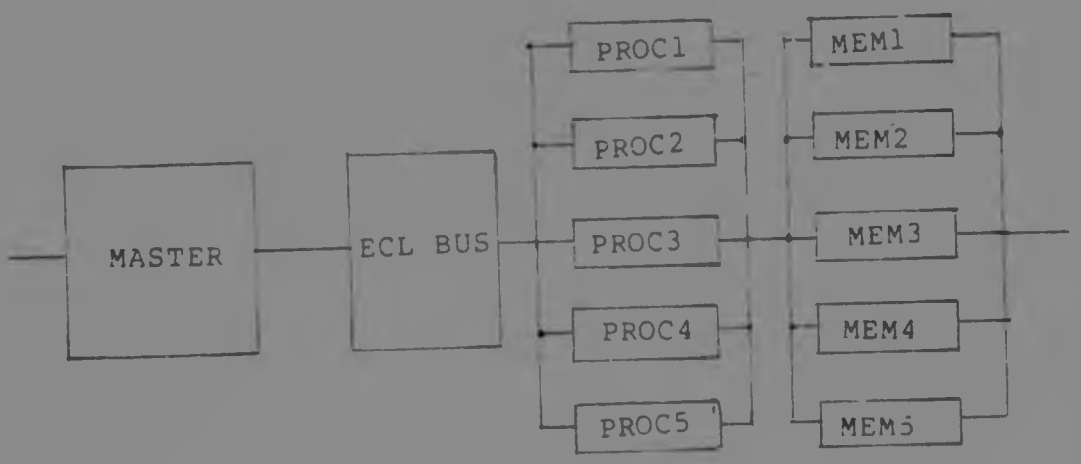


FIGURE J-4 RAMROD'S RELIABILITY



FIGURE J-1 SERIAL RELIABILITY

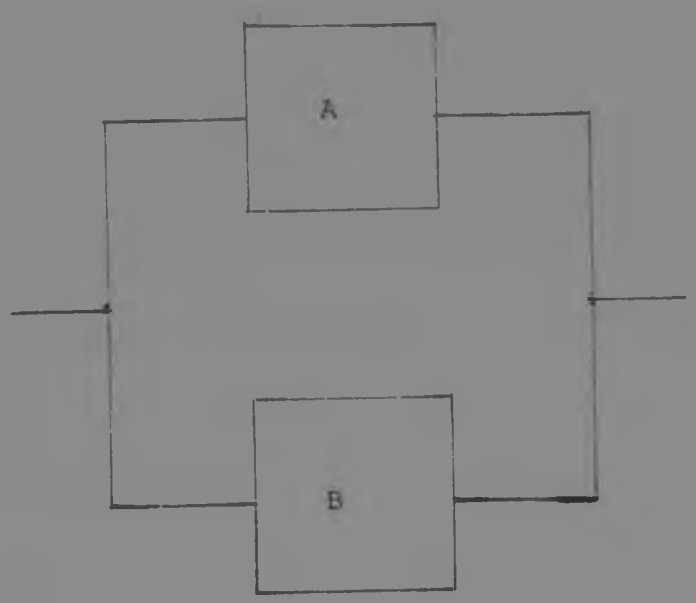


FIGURE J-2 PARALLEL RELIABILITY

A uniprocessor system with L=20000 FITS and 2 boards

now MTBF = $\int_0^\infty R(t)dt$

Where $R = e^{-2Lt}$ = 1/2L = 2.8 years

for a quintiple redundant system

$R_{p5} = R^5-5R^4+10R^3-10R^2+5R$

thus MTBF = $\frac{1}{10L} - \frac{5}{8L} + \frac{10}{6L} - \frac{10}{4L} + \frac{5}{L}$ = 20 years

Now if Repair is introduced then

MTBF = $\frac{u^4}{L^5}$

where u = 1/24 and thus u >> L

therefore

MTBF = 10^{18} years.

However Ramrod has a Reliability of;

$R_r = R.R.R_{p5}^2$

therefore MTBF = 1.3 years

Or if Ramrod is considered as 4 identical units , and an any failure causes a system failure then

MTBF = 1/4L = 1.4 years which is almost 1.3 years.

Thus it is obvious that the master and the ECL bus are the bottlenecks and must be duplicated. Now if these are duplicated then for a double redundant system

$R_{p2} = 2R - R^2$

then the reliability of Ramrod is

$R_{p2}^2 . R_{p5}^2$

therefore

$$MTBF = \left(1 - \frac{1}{28L}\right)\left(1 - \frac{1}{26L}\right)\left(1 - \frac{1}{24L}\right)\left(1 - \frac{1}{22L}\right)\left(1 - \frac{1}{20L}\right)\left(1 - \frac{1}{18L}\right)\left(1 - \frac{1}{16L}\right)\left(1 - \frac{1}{14L}\right)\left(1 - \frac{1}{12L}\right)\left(1 - \frac{1}{10L}\right)\left(1 - \frac{1}{8L}\right)$$

$= 6.7$ years.

however if repair time is now introduced, then Ramrod can be analysed as a quadruple redundant system which requires three units to operate, because if one unit fails then there is still the other identical unit which can now take over operation.

Thus MTBF(Ramrod) $= \frac{7L + u}{12L^2} = 50 \times 10^6$ years

REFERENCES

[AGR 82] AGERWALA, T. and ARVIND, "Data Flow Systems"; IEEE Computer Vol. 15 No. 2, February 1982.

[ALE 81] ALEXANDER, P, "Array Processor Design Concepts"; Computer Design December 1981.

[ANA 80] ANACKER, W, "Josephson Computer Technology: An IBM Research Project"; IBM J. Res. Develop. vol 24 No. 2 March 1980.

[ACR] ACKERMAN, W.B, "Data Flow Languages"; IEEE Computer Vol. 15 No.2 February 1982.

[AGR 76] AGRAWALA, A.K, RAUSCHER, T.G., "Foundations of Microprogramming Architecture, Software and Applications"; Academic Press, Inc 1976

[AMD 1] ADVANCED MICRO DEVICES The AM2900 Family Data Book. 1979

[AMD 2] ADVANCED MICRO DEVICES Build a Microcomputer Series. 1979

[ALD] AL-DABASS, D, "Microprocessor based Parallel Computers and their Application to the solution of Control Algorithms"; Control Systems Centre Report, University of Manchester, Jan 1977.

[ARD] ARDEN, B.W., GINOSAR, M., "MU/2: A Multiprocessor/Computer Architecture"; I.E.E.E. Transactions on Computers, Vol. C-31, No. 5 May 1982.

[BRI 78] BRINCH HANSEN, P., "Distributed Processes: a concurrent programming concept"; Comm A.C.M. Vol 21, No.11 Nov. 1978 pp 934-941

[BLA] BLAKE, R.E., "Advantages to be gained from Process Control by Computer"; Electronics and Power ,March 1977

[BOW] BOWEN, B.A., BUHR, R.J.A., " The Logical Design of Multiple Microprocessor Systems"; Prentice hall Inc. New Jersey 1980

[BAR] BARRON, D.W., "Computer Operating Systems",Chapman and Hall London 1971

[BSO 1] BOSTON SYSTEMS OFFICE BSO Cross Librarian (MLIB) User Manual . 19 Feb. 1981.

[BSO 2] BOSTON SYSTEMS OFFICE BSO Cross-Reference Program (MREF) User Manual . 18 May 1981.

[BSO 3] BOSTON SYSTEMS OFFICE BSO Cross Linkage Editor (MLINK) User Manual . 2 July 1981.

[BSO 4] BOSTON SYSTEMS OFFICE BSO Relocating Cross Assembler (CR8085) User Manual . 7 July 1981.

[BSO 5] BOSTON SYSTEMS OFFICE BSO Object File Conversion Utility (OBJCNV) User Manual . 7 July 1981.

[BSO 6] BOSTON SYSTEMS OFFICE BSO Simulator/Debugger (SI8085) User Manual . 7 July 1981.

[BIS] BISCARRI, J.,GAGO, A., "Low-Cost Multiprocessing System"; Electronics Letters Vol 17 no.24 26 Nov 1981.

[BLO] BLOOD, W.R.Jr., "Mecl System Design Handbook";2nd ed. Motorola Inc,1972

[BRI 73] BRINCH-HANSEN, P., "Operating System Principles"; Prentice-Hall, 1973

[BAK] BAKER, K. "Specifying The System" Microprocessors and Microsystems, Sept 1980 Vol. 4 No. 7

[BAR] BARTEE, T.C."Digital Computer Fundamentals";3rd ed. Tokyo: McGraw-Hill Kogakusha Ltd., 1972.

[BRK] BRINKMAN, B.L., " A Selection of Multi-Microcomputer Systems"; Mini-Micro Systems, JAN 1979.

[BUH] BUHR, R.J.A., ET AL. "Why Multiple Microprocessors"; International Symposium on Mini and Micro Computers Montreal Canada, 1977.

[BERT] BERNHARDT, D,. and SCHMITTER, E,. "Design and Implementation of Fault-Tolerant Multi-Microcomputer Systems"; Microprocessors and Microsystems, Vol.5 No. 4 May 1981.

[BERD] BERNHARD, R,. "The 'no-downtime' computer"; I.E.E.E. Spectrum September 1977 pp 33-37.

[CRA] CRANE, R.C., "Software pack and computer link DEC computers in an Ethernet"; Electronics Dec. 15,1981

[DOY] DOYLE, E.A.Jr., "How Parts Fail"; I.E.E.E. Spectrum October 1981.

[DES] DESIMONE, S.E., "Test Techniques for ECL loaded Boards"; Computer Design,June 1982.

[DIJ] DJIKSTRA, E.W., "Co-operating Sequential Processes", reprinted in "Programming Languages", edited by Genuys, F., NATO Advanced Study Institute, Academic Press, London 1968,pp 43-112.

[DAV 78] DAVIDSON, J., ET AL., "A Generalized Multiprocessor System"; I.E.E.E 1978.

[DAV 82] DAVIS, A.L.,KELLER, R.K, "Data Flow Program Graphs"; IEEE Computer Vol. 15 No. 2, February 1982.

[DAV 80] DAVIS, C.G., COUCH, R.L., "Ballistic Missile Defense: A Supercomputer Challenge"; Computer November 1980

[DEN] DENNIS, J.B., "Data Flow Supercomputers"; Computer November 1980

[DUT] DUTHOIT - "Use of Multi-Microprocessor Architecture for the New SNCF Computer Systems Network" . Paper read at the 8th ORE Colloquium , Madrid , 5 and 6 May 1981.

[ENS 80] ENSLOW, P.H. Jr. "What is a 'Distributed' Data Processing System?" Computer Jan 1978 Vol. 1980 pp 75-96

[ENS 74] ENSLOW, P.H.Jr. Comtre Corporation, " Multiprocessors and Parallel Processing";New York :John Wiley and Sons,1974

[EUR] EUROMICRO JOURNAL; Vol 5:

[FEL] FELDMAN, J.A., "High Level Programming for Distributed Computing"; Comm. A.C.M. Vol. 22 No. 6, June 1979, pp 353-368.

[FAR] FARBER, G., "Principles and Applications of Decentralized Process Control Computer Systems"; Distributed Process Computer Sytstems.

[GIL BEHR] GILOI, W.K., BEHR, P.M., "Making Distributed Multicomputer Systems Safe and Programmable"; Internal report at the Technical University of Berlin, West Germany.

[GAJ] GAJSKI, D.D.,et al, "A Second Opinion On Data Flow Machines and Languages"; IEEE Computer Vol. 15 No. 2, February 1982.

[HOA 78] HOARE, C.A.R., "Communicating Sequential Processes"; Comm.A.C.M. Vol. 21, No. 8, Aug. 1978, pp 666-677.

[HOP] HOPKINS, A.L., et al ,"FTMP A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft"; Proceedings of the I.E.E.E. , vol. 66 No. 10 Oct 1978.

[HOA 72] HOARE, C.A.R., "Towards a Theory of Parallel Programming"; Operating System Techniques, Academic Press, New York, 1972 pp 61-71

[HUD] HUGHES, P., DOONE, T.," Multi-Processor Systems"; Microelectronics and Reliability, vol. 16 pp 281-293, Pergamon Press, 1977.

[HUG] HUGHES, J., "Developement Systems: Ethernet"; Computer Design , May 1982.

[HIR] HIRD, D. J, ELLIOT, D. M .,"Bit-slice Microprocessors- their use and application"; Electronics and Power, vol 25, No. 4, March 1979, pp 179-184

[HOP 80] HOPKINS, R.L., "Meeting the Challenge of Automated ECL Te ing" Computer Design Sept 1980 pp 115-12?.

[INT 1]INTEL CORPORATION MCS 80/85 Family users manual , Oct. 1979.

[INT 2] INTEL CORPORATION Compon nt data catal j , 1980.

[INT 3] INTEL CORPORATION Peripheral design handbook , Aug. 1980.

[INT 4] INTEL CORPORATION Memory Design Handbook , Jan. 1981.

[INT 5] INTEL CORPORATION SDK 5 Kit User's Manual, .

[JOH] JOHNSON, D., "Logic Analyser and mo Development System, Aid in Debugging Multiprocessing Networks"; Digital Design Nov 1980

[KAH] KAHNE, S., ET AL., "Automated Control by Distributed Intelligence"; Scientific American 197?.

[KART] KARTASHEV, S.P. and KARTASHEV, S.I. "Supersystems for the 80's"; I.E.E.E. Computer Nov 1980

[KARP] KARPLUS, W.J., AND COHEN, D., "Architectural and Software Issues in the Design and Application of Array Processors"; I.E.E.E. Computer Sept. 1981

[KER] KERGUELEN R. "Use of Micro-Computers in Distributing Processing on the SNCF" . Paper read at the 8th ORE Colloquium , Madrid , 5 and 6 May 1981.

[KOP 81] KOPETZ H. "Distributed Computer Control Systems" . Course presented by The Continuing Engineering Education Division , University of the Witwatersrand , 4 to 6 Nov. 1981.

[KOP 82] KOPETZ H, ET AL. "The Architecture of MARS"; A Technical University of Berlin Report MA 82/2, April 1982.

[KOY] KOYAMA, S., MIURA, R., "A Multiprocessor System for Fast On-Line Simulation of Dynamical „Systems", reprinted from Simulation of Systems, Delft 1976, North-Holland Amsterdam: 1976

[KOY] KOYAMA, S., MIURA, R., "An all-Digital Dynamical System Simulator using Parallel Processing", reprinted from A link between Science and Applications of Automatic Control, New York and Oxford: Pergammon Press, 1971

[KOY 77] KOYAMA, S., ISURUGI,Y., et al,"A Realization of a DDA System for Continuous Dynamical System simulation with a Universal Multimicroprocessor System 'HARPS' ", Euromicro Newsletter Vol. 3, No. 4, 1977.

[LAM] LAMBRECHS, J.S.D., RODD, M.G., "Highly Reliable Software for use in Fail-Safe Control"; Preprints of the 3rd IFAC/IFIP Symposium on Software for Computer Control , 5-8 Oct. 1982

[LISK] LISKOV, B., "Primitives for Distributed Computing" Proc. 7th Symposium on Operating Systems Principles, Pacific Grove California Dec. 1979 pp 33-42

[LIST] LISTER, A.M., " Fundamentals of Operating Systems"; The Macmillan Press Ltd., 1975

[McD] McDONALD, W.C., WAYNE SMITH, R., "A flexible test-bed for Real-Time Applications"; Computer ,Oct. 1982 pp 25-39

[MET] METCALFE, R.M., BOGGS, D.R., "Ethernet : Distributed Packet Switching for Local Computer Networks"; Communications of the ACM July 1976, Vol. 19, No. 7

[MAC] MACLEOD, I.M., RODD, M.G., "An Evaluation of the applicability of Interprocess Communication Primitive Proposals to Distributed Process Control"; Preprints of the 3rd IFAC/IFIP Symposium on Software for Computer Control , 5-8 Oct. 1982

[KOY 77] KOYAMA, S., ISURUGI,Y., et al,"A Realization of a DDA System for Continuous Dynamical System simulation with a Universal Multimicroprocessor System 'HARPS' ", Euromicro Newsletter Vol. , No. 4, 1977.

[LAM] LAMBRECHS, J.S.D., RODD, M.G., "Highly Reliable Software for use in Fail-Safe Control"; Preprints of the 3rd IFAC/IFIP Symposium on Software for Computer Control , 5-8 Oct. 1982

[LISK] LISKOV, B., "Primitives for Distributed Computing" Proc. 7th Symposium on Operating Systems Principles, Pacific Grove California Dec. 1979 pp 33-42

[LIST] LISTER, A.M., " Fundamentals of Operating Systems"; The Macmillan Press Ltd., 1970

[McD] McDONALD, W.C., WAYNE SMITH, R., "A flexible test-bed for Real-Time Applications"; Computer ,Oct. 1982 pp 25-39

[MET] METCALFE, R.M., BOGGS, D.R., "Ethernet : Distributed Packet Switching for Local Computer Networks"; Communications of the ACM July 1976, Vol. 19, No. 7

[MAC] MACLEOD, I.M., RODD, M.G., "An Evaluation or the applicability of Interprocess Communication Primitive Proposals to Distributed Process Control"; Preprints of the 3rd IFAC/IFIP Symposium on Software for Computer Control , 5-8 Oct. 1982

[MAK] MAKINO, K., KOYAMA, S., et al., "A Multi-Purpose Digital Simulator (HOSS) Using a Hierarchical Distributed Multi-processor Technology", reprinted from Simulation of Systems '79, Sorrento 1979, North-Holland Amsterdam: 1979

[MAI] MAISEY, D., "Distributed Processing for Industry"; New Electronics September 9 198.

[MIC] MICRO NEWS, A Newsletter from L'Electron S.A. Microprocessor Division; Chapter 6.

[MOT B] MOTOROLA INC."Mecl High Speed Integrated Circuits"; Series B.

[MOT 1] MOTOROLA INC."EXOslice User's guide"; Switzerland: 1977

[MOT 2] MOTOROLA INC."M68RC EXORciser User's Guide"; Switzerland: 1975

[MOT 3] MOTOROLA INC."M68MDOS3 EXORdisk II/III Operating System User's Guide"; 1st ed., 1978

[NIS] NISSEN, J.C.D. [illegible], C.V., "Microprocessors for Telecommunications"; Electronics and Instrumentation,vol 11 No.4, April 1980,pp 67-75

[NAD] NADIR J. , McCORNIC B - "Bus Arbiter Streamlines Multiprocessor Design" . Computer Design , June 1980 , pp.103-109.

[NOV] NOVAK M., "Gate Arrays - fabrication, design and economics"; MSc Research Report Dec. 1982, University of the Witwatersrand, Johannesburg

[PAT] PATEL, J.H. "Performance of Processor-Memory Interconnections for Multiprocessors"; I.E.E.E. Transactions on Computers Vol. C-30 No. 10 October 1980.

[POL] POLCZYNSKI, M.H., "Multiple mp Control System raises throughput without bus conflicts"; Electronic Design Jan. 7, 1982.

[PEB] PEBERDY, N. "Digital Electronics-Logic Families"; The Electrical Engineer.Sept 1980 pp 13-20,Thompson South Africa

[RAV] RAVASIO, P.C., et al, "Local Computer Networks"; North-Holland, Amsterdam, 1982.

[RAB] RABINOWITZ, A.E., and RODD, M.G., "Ramrod a Multi-Microprocessor Computer"; Proc. 2nd South African Computer Symposium, OCT. 1981, Pretoria.

[ROD 76] RODD, M.G.,"Organisation of Industrial Control Computers" PhD. Thesis, University of Cape Town, 1976

[ROD 82] RODD, M.G., " The Impact of Microelectronics on Distributed Control Systems "; Inaugral Lecture for the Head of the Dept. of Electrical Engineering , University of the Witwatersrand, Johannesburg 28th October 1982.

[RAT] RATTNER, J., LATTIN, W.W., "ADA determines the Architecture of 32 bit Microprocessor"; Electronics, Feb. 24 1981.

[SUG 80] SUGARMAN, R., " 'Superpower' Computers"; I.E.E.E. Spectrum April 1980.

[SMI] SMITH, ,D,J., "Reliability and Maintainability in Perspective"; MacMillan, 1981.

[SAT] SATYANARAYANN, M., "Commercial Multiprocessing Systems"; Computer May 1980 pp 75-96

[STI] STIFFLER, J,J., "How Computers Fail"; I.E.E.E. Spectrum October 1982.

[SWA] SWARTZLANDER, E,E,Jr., GILBERT, B,K., "Supersystems: Technology and Architecture"; I.E.E.E. Transactions on Computers, Vol. C-31, No. 5 May 1982.

[TOR] TORRERO, E.A. "They said it couldn't be done"; I.E.E.E. Spectrum Sept. 1985.

[TAN] TANAKA, Y., MIYASHITA, K., et al "HARPS (Hokkaido university Array Processor System): A New Hierarchical Array Processor System", 2nd Euromicro Symposium on Micro Architecture, Venice: Oct 1976

[TOO] TOONG, H.D., "Multi-Microprocessor Systems"; Siemens Forsch-u. Entwickl.-Ber Bd 7(1973) nr. 6,Springer-Verlag 1978.

[TRAKH] TRAKHTENGERTS, E.A., SHURAITS, Yu.M., "Software Design for Multiprocessor Systems Computer Control"; Internal report at the Institute of Control Sciences, Moscow, USSR.

[THE] THEIS, D., " Array Processor Architecture"; I.E.E.E. COMPUTER Sept. 1981.

[TEX 1] TEXAS INSTRUMENTS INCORPORATED The TTL Data Book for Design Engineers; 1973

[TEX 2] TEXAS INSTRUMENTS INCORPORATED Supplement to the TTL Data Book 1974

[US DOD] UNITED STATES DEPT. of DEFENCE, "Reference Manual for the ADA programming Language, July 1980

[VIC] VICK, C.R., et al " Adaptable Architectures for Supersystems"; Computer November 1980.

[TOR] TORRERO, E.A. "They said it couldn't be done"; I.E.E.E. Spectrum Sept. 1981.

[TAN] TANAKA, Y., MIYASHITA, K., et al "HARPS (Hokkaido university Array Processor System): A New Hierarchical Array Processor System", 2nd Euromicro Symposium on Micro Architecture, Venice: Oct 1976

[TOO] TOONG, H.D., "Multi-Microprocessor Systems"; Siemens Forsch-u. Entwickl.-Ber Bd 7(1978) nr. 6,Springer-Verlag 1978.

[TRAKH] TRAKHTENGERTS, E.A., SHURAITS, Yu.M., "Software Design for Multiprocessor Sy tems Computer Control"; Internal report at the Institute of Control Sciences, Moscow, USSR.

[THE] THEIS, D., " Array Processor Architecture"; I.E.E.E. COMPUTER Sept. 1981.

[TEX 1] TEXAS INSTRUMENTS INCORPORATED The TTL Data Book for Design Engineers; 1973

[TEX 2] TEXAS INSTRUMENTS INCORPORATED Supplement to the TTL Data Book 1974

[US DOD] UNITED STATES DEPT. of DEFENCE. "Reference Manual for the ADA programming Language, July 1980

[VIC] VICK, C.R., et al " Adaptable Architectures for Supersystems"; Computer November 1980.

[WEI] WEITZMAN, C., "Distributed Micro/Minicomputer Systems, Structure, Implementation and Application"; Prentice-Hall, N.J. 1980

[WAT] WATSON, I., GURD, J., "A practical Data Flow Computer", IEEE Computer,February 1982

[WIL] WILKES, M.V. STRINGER, J.B., "Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer"; reprinted in "Computer Structures:Readings 9

[WILD] WILD, B., "A Support System for Developement of a Microprogrammed Controller", MSc Dissertation (in preparation) Dec. 1982, University of the Witwatersrand, Johannesburg

[WOO] WOOD, A.R., "A Multi-Microcomputer Interface"; Microelectronics and Reliability Vol 19 pp 513-522:Pergamon Press Ltd. 1980

[ZAK] ZAKS, R., WILMINK, J., NICOUD, J.D. "Microcomputer Architectures". Euromicro Symposium. Amsterdam: North Holland, Oct 1977.

[ZOC] ZOCCOLI, M.P., SANDERSON, A.C., "Rapid Bus Multiprocessor System", Computer Design, Nov 1981, pp 189-200

Author Rabinowitz A E
Name of thesis Ramrod: an experimental multi-microprocessor

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.