DESIGN OF A MULTI-PROCESSOR CONTROL SYSTEM

by

ELIAS KARAGIORGAS

Submitted for the Degree of

Master of Philosophy

at

The University of Aston in Birmingham

October 1979

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to his supervisor, Professor H. A. Barker for his guidance and advice throughout his research.

I also wish to thank Mrs. Helen Turner and Miss N. P. Freeman for typing this thesis.

DEDICATION

I wish to dedicate this

thesis to my parents.

DESIGN OF A MULTI-PROCESSOR CONTROL SYSTEM

Elias Karagiorgas

Master of Philosophy 1979

Summary

The aim of the research undertaken is to investigate the aspects of multi-processor systems concerned with their performance, control and architectural design.

Various architectural designs are studied in terms of the tasks which can be performed in accordance with the objectives and goals of the system user, the resulting needs of the system and the types and characteristics of the microprocessors.

The system eventually selected appeared to be architecturally feasible, essentially practical in its uses and takes advantage of the capabilities and flexibilities of the microprocessors in use. This design uses five INTEL 8008 microprocessors, which execute tasks independently or in co-operation, exchanging data and information through common access to a shared memory. The shared memory feature is an essential part of the performance of the system and is the main design characteristic of it. The system was tested and operated in isolation and subsequently operated in conjunction with a 9900/4 Texas Instruments microprocessor system which formed an external communication system. Observations, conclusions and recommendations for improvement are given.

The thesis aims to record observations of the operations of a multi-processor system in order to derive an assessment of its potential uses, particularly for automatic localised control of systems.

MULTI-PROCESSOR SYSTEM, MULTI-PROCESSOR COMMUNICATIONS, TASK ALLOCATION, MEMORY SHARING, MULTI-PROCESSOR CONTROL

CONTENTS

LIST OF DIAGRAMS

INTRODUCTION

Microelectronics is the most influential technology of the twentieth century.

Within this technology the microprocessor is the most influential product, as an agent of radical change that is bringing new industrial methods, producing an evergrowing range of new products, and posing serious questions for society and any national economy.

Dispersion and distribution of information processing functions have been given impetus by recent advances in semi-conductor technology and reduced hardware costs.

Integrated circuit technology matured sufficiently to permit the realisation of the "microprocessor". A microprocessor may be defined as a device which fetches and executes instructions, in a predefined sequence, assumed to be stored in a memory with which the processor interfaces.

With its present, and in the future, increasing capabilities the microprocessor can serve for a system designer as a control element and is responsible for data acquisition, processing, display, setpoint control and communications.

Since the introduction of microprocessors, multiple microprocessor configurations have been the vision of

automatic systems designers as it approaches to higher reliability and higher computational bandwidth.

In particular, multi-processor configurations in which a number of identical processors share a common memory or common memories, have been the subject of intensive study and analysis.

Until recently, however, implementation of computer structures of this type have been limited to special purpose military computers. The two primary reasons for the slow acceptance of multi-processor configurations have been cost and the difficulties associated with the interrelated dynamic properties of systems control and fault tolerance. Although significant advances have been made in the latter two areas, a great deal of effort must still be expended in an attempt to develop an integrated methodology which combines the findings in each of the two areas. On the other hand, the rapidly decreasing cost of hardware which is exemplified by the widespread acceptance of microprocessors has provided the push for multi-processor investigations which are not necessarily bound by some of the more difficult problems of control and reliability. The relationship between total system cost and total system capability can be described by the curves of Fig. 1. Ideally, the relationship between cost and capability should be linear. That is a small increment in system cost should yield a comparable increase

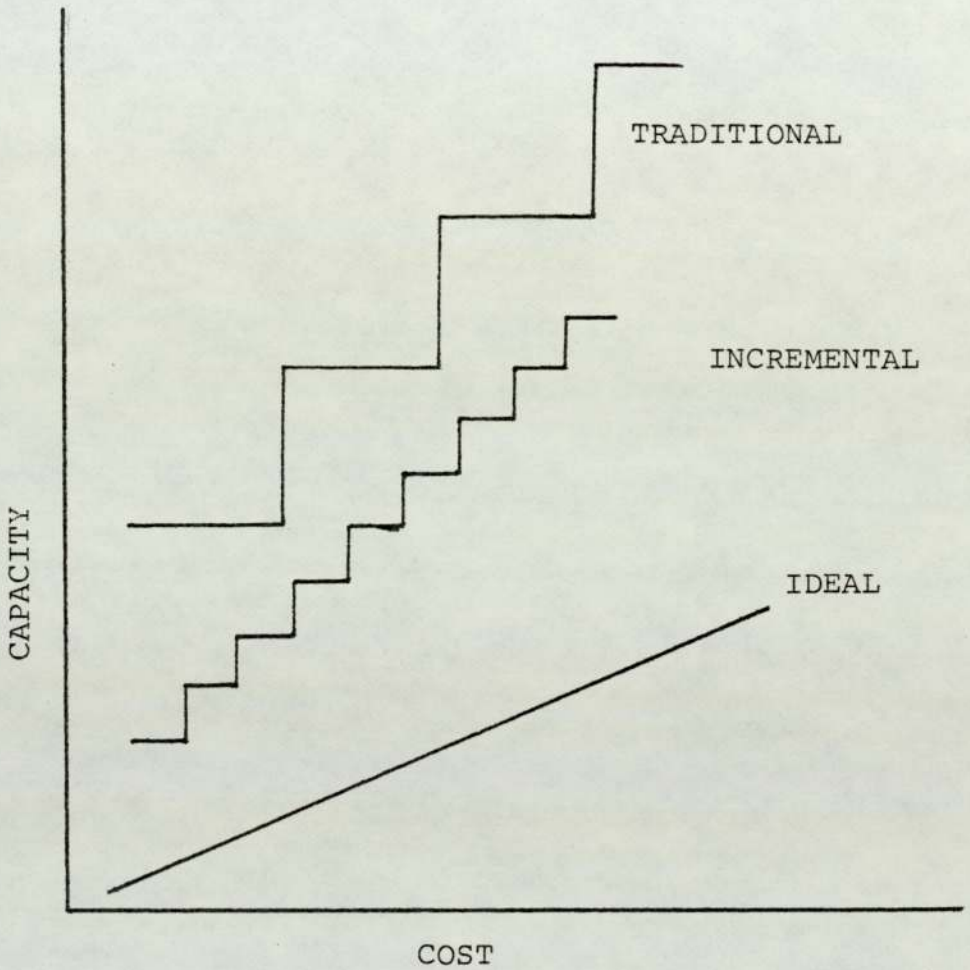


Fig. 1

Approaches to System Growth

in capability. The increase is a function of the slope of the curve: a larger slope is indicative of a more cost-effective investment. An installation however, of a multiprocessor finding itself with a saturated system may have to resort to large investments for more mainframe memory for more (or a different type) of secondary memory, or perhaps to an upgrade to the next more powerful member of an upwards compatible family.

A desirable compromise between these two is one which allows system components to be added incrementally whilst requiring only modest increase in cost. In the past this has been the rationale for the study of systems in which components processors and memory in particular can be added as growth requirements dictated. An added feature of a system of this type has been the ability of the system to withstand failures in a processor, for example, without seriously impairing system performance.

The ever-increasing capabilities of microprocessors coupled with their attractive cost performance in parallel with the hardware advances which brought lower costs and the potential for a variety of physical interconnection possibilities have reduced the limitations for the wide implementation of a multi-processor system. Systems which combine private and shared memories, buses, switches, stand-alone processors etc. are the study of many research institutions, public or private, in an attempt to realize

the problems and stretch the capabilities associated with these systems.

This report tries to examine the problems of designing and controlling a multiprocessor system. The research was conducted here in the Electrical and Electronic Engineering Department of the University of Aston in Birmingham, using the INTEL's 8008 microprocessor. The Texas Instruments latest microprocessor system the 9900/4 was used in the final stages of the research as part of an independent communication system.

Details on the operations of the 8008 and 990/4 are given in Appendices A and B.

CHAPTER 1

MULTIPLE MICROPROCESSOR SYSTEMS - A COMPARISON OF DIFFERENT APPROACHES

CHAPTER 1

MULTIPLE MICROPROCESSOR SYSTEMS - A COMPARISON OF DIFFERENT APPROACHES

1.1 INTRODUCTION

Since the announcement of the first commercial microprocessor in 1971, integrated CPU's have evolved from laboratory curiosities to ubiquitious fundamental system building blocks. Moreover, rapid advances in LSI mere technology during the early 70's have resulted in ever larger RAMS and ROMS.

The advances made by LSI technology have not been applied solely to microprocessors and memories. Complex bit general purpose logic blocks are being integrated on single chips in increasing numbers. Amongst them the development of UART for data communications and single-chip peripheral interfaces. The ability to introduce microprocessor control into many systems currently implemented via hard-wired logic will bring to these systems all the attendant advantages of stored program control. These include greatly improved flexibility, reliability, ease of maintenance and lower cost.

A natural evolution of microprocessor-based system architectures is that of distributed processing, i.e.

multi-micro-computer systems. In distributed intelligence systems, intelligent subsystems, dedicated to specific tasks, communication in an optimal fashion to improve system throughput, increase reliability, and add a new dimension of flexibility.

There is currently no established methodology for interconnecting sets of processors for the purpose of building general-purpose or even special purpose computer systems.

However, there does exist an interesting range of possibilities that span this range: computer networks, multiprocessors, and multiple arithmetic unit processors.

1.2 COMPUTER NETWORKS

Perhaps the most widely known computer network is the minicomputer/multiprocessor for ARPA network[1]. An important attribute of this network is the data transmission bandwidth between computers. The other important attribute of the inter-computer links is the access, or latency time for each unit of information sent between microcomputers. The system contains an expandable number of identical processors, each with some 'private' memory, an expandable amount of 'shared' memory to which all processors have equal access, and an expandable amount of I/O interface

controllable by any processor. The system achieves modularity and reliability by making all processors equivalent, so that any processor may perform any system task, thus systems can be easily configured to meet the throughput requirements of a particular job. The scheme for interconnecting processors, memories and I/O is also modular, permitting interconnection cost to vary smoothly with system size. In considering which minicomputer might be most easily adaptable to a multiprocessor structure, the internal communication between the processor and its memory was of primary concern.

The overall design is represented in Fig. 1.1. Processors make access to shared memory via the switching arrangement. The shared memory of the multiprocessor system is intended to contain a copy of the program as well as considerable storage space for message buffering, global variables, etc. The ARPA network is an example of a loosely coupled network because of the 50 K/bits links between computer in the network and the 100-200 ms latency times associated with cross-network transmissions of packets of information.

1.3 MULTIPROCESSOR SYSTEMS

There are different approaches on the basic structure of a multiprocessor system. Its distinguishing characteristic is that the processors share primary memory.

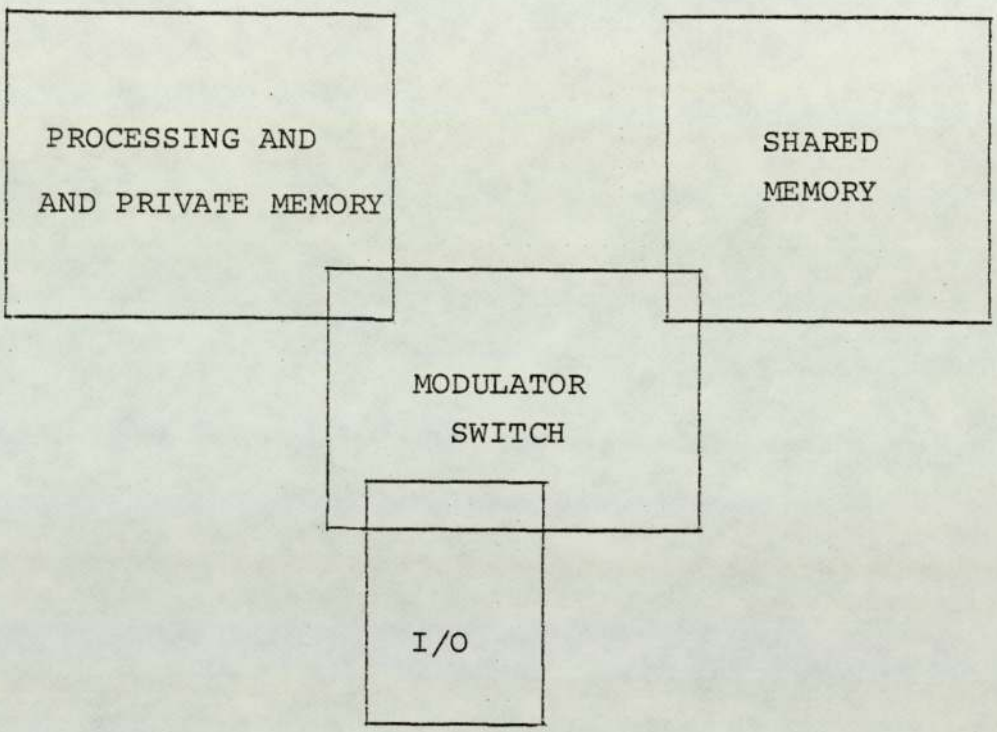


FIG. 1.1 ARPA NETWORK

Depending on the applications some acquire private memory. Main memory and I/O channels are accessible by every CPU. Multiprocessor systems can operate in several modes. In one, the processor may co-operate in solving a problem which requires more computing power than a single processor affords. Each processor might control a position of an overall process, with the necessary co-ordination between the control strategies effected through the processor interconnection means. Both processors are of equal importance in maintaining control of the process, and both must be operating to obtain optimum performance. A more common mode of operation in industrial control is usually called duplexed operation, and its purpose is to increase the reliability of the total system. A primary processor normally performs the control task. In the event of a failure a second, back up processor takes over.

1.3.1 A distributed multi-processor system

The multi-microprocessor system developed by the Central Research Laboratories[2] is essentially a distributed microcomputer system composed of several kinds of sub-systems. Each subsystem is given autonomous control capabilities to facilitate control problems, securing independence between subsystems as well. Instruction execution cycles of typical LSI processors are 5 or 10

times slower than the main memory cycle. Thus it is expected that the main memory can be shared in time and space with several processors to improve memory utilisation.

The virtual memory was chosen in order to allow each user to use more memory capacity than it could if it were restricted by the actual main memory. Furthermore, to avoid operating system complexity, a page detector was developed which notifies a multiprocessor communication adaptor of a page being unused for a long time. Asynchronous ambiters were also used, which handle the simultaneous access to the main memory and I/O devices.

Several advantages are claimed for distributed processing. They offer division of labour, as remove units off-load the processor at the next higher level for enhanced performance of the total system. Although the central operating system must still be a multiprogramming system, the degree of multiprogramming is reduced, since some functions will be handled directly by remove computing units. They also offer a degree of modularity which is difficult to achieve with single centralized computer system.

1.3.2 A single communication bus multi-processor system

In an article on interprocessor communication scheme for multiple microcomputer systems[3], L.Eaton and E. Page advocate the single communication bus as a method of linking several

devices together to allow sharing data with advantages on flexibility, universitality and economy. Despite the advantages of the single bus, it can accommodate only one message at a time, thereby restricting the rate at which transmission may occur.

Alternative concepts for achieving communications between processors have made use of multiple buses, multi-part memories, or cross-bar switches. Typically, the problems of bus arbitration and synchronisation have become increasingly more cumbersome as the number of processors in the system increased. In order to exploit the advantages of the single bus concept and at the same time, minimize its disadvantages, the Pierce loop[4] is being used as a communication bus. Conceptually, a Pierce loop is a set of registers connected in a circular manner that moves a packet of information in a fixed direction from one register to an adjacent register in each unit of time. Each processor has its own memory as well as ancillary circuits for bus interfacing and monitoring. Each processor has a unique name, p or v, and communication between processors takes place by tagging information to be transmitted with either the p or v name of the desired destination and placing it on a loop.

1.3.3 Multi-processor systems with global and local memories

In their article on a Multiple Microprocessor Network, J.E. May and L.J. Krakaver[5] chose a memory system allowing a limited amount of local memory for each processor, with a high bandwidth global memory system accessible by all processors. All global memory accesses as well as inter-processor and I/O controller communications are done over the system bus. Ths bus allows transfer rates of 1 byte every 167 nsec. It was estimated that each Motorola 6800 or Intel 8080 made a memory/bus access on the average of once every 1.5 ms. Thus, the bus access requirements of 8 microprocessors are roughly balance with the bus bandwidth. They also used a bus contention priority system, task dispatching and interrupts. A master controller was also used to perform clock generation, memory refresh, memory control, control panel logic and the I/O controller, and task dispatcher for the processors.

Service requests can be from external sources or from queued internal requests. There are eight priority levels at which these service requests may be present.

The distributed microprocessor system for Avionics by M. Moore[6] consists of identical processing elements interconnected by a network of serial buses. A global bus interconnects all processors in the network and provides a channel for network control and system data. Separate

I/O ports are provided to eliminate the need for real-time command-response interaction on the network buses.

The memory unit serves for both program and data storage. It is asychronous and can therefore be realized with a mixture of technologies. Total size required was 4K and 8K bit words. The processor I/O unit is intended to be the interface between the network and aircraft devices. The device is a single channel that can be set up for autonomous or program controlled transfers. A 16 bit parallel I/O path is used to multiplex command and data information. One interrupt line is provided to the device.

The general configuration of a multiple processor system advocated by C. Reyling[7], uses a common data bus as many microprocessors are able to time-share such system resources as memory, I/O, and peripherals. In this asymmetric structure individual processors have fixed specialized processing functions. It could be used in dedicated applications where type, frequency of occurrence and relative importance of tasks are known in advance. Processors may be specialized to carry out one particular type of task. Once processor, for instance may perform all I/O operations, another perform memory accessing, another provide file maintenance and so on. Specialisation may occur via the software programs executed (local memories), and hardware architectural features (number

of registers, interrupt capabilities, stock processing). Often a side benefit of this partitioning is simplification of programming, since each task can now be treated as an independent module, with no provisions required for execution of other tasks by a given microprocessor.

1.4 MULTIPLE ARITHMETIC-UNIT PROCESSOR SYSTEMS

The third form of computer organisation that incorporates multiple processing elements is the multi-arithmetic and logic unit processor. The fundamental difference between this type of structure and multi-processors is that all the ALU's in the multi-ALU processor support a single instruction stream, while each of the processors in the multiprocessor supports its own instruction stream[8].

If we define a processor to be a unit capable of both decoding and executing instructions, then the multi-ALU processor is not really a multiple processor system. However, multi-ALU organisations are often considered as alternatives to multiprocessors and derive the same benefits from advances in LSI technology as multiprocessors.

1.5 CONCLUSIONS

Networks, multiprocessors, and multi-ALU computers, have been presented as three methods of organizing processors to build highly parallel computer systems. The three classes

can be thought of as as varying along a single dimension, the degree of coupling between processors in the system. In the computer network the minimum access time for a processor is the access time to local memory. In a multiprocessor, each processor has direct access to global data stored in primary memory. Since interprocessor communication occurs by sharing primary memory, the interaction times are on the order of 1-50 μs. In a multi-ALU computer, the analog of interprocessor communication is the transfer of control information that occurs between the control unit and its associated processing elements. The position of multiple processor organisations has a strong influence on its suitability to a particular application. An application consisting of a set of parallel processes that need to interact or share data only every 10-100 s can run on a loosely coupled computer network. At the other end, algorithms that require the parallel execution of arithmetic operations within single expressions force the interaction times between processing elements to occur almost every instruction cycle. Thus the average time between interprocess interaction becomes a critical time constant of an application, and provides a good indication of the type of multiple processor organisation that will be most suitable.

Several advantages may be realised with multiprocessor systems in general. Througput often increases almost

directly with the number of processors while system cost increases by only a small amount. Shared system resources offer an economic advantage by eliminating devices which would need to be duplicated in separate stand-alone systems. Shared resources also provide direct access to data which might otherwise require transmission from one system to another.

The characteristics of LSI processors strongly suggest the multiprocess system as a practical alternative to a multi-task monoprocessor system, since the cost performance will be improved as a consequence of sharing expensive memory and I/O units.

CHAPTER 2

ARCHITECTURE OF MULTIPLE MICROPROCESSOR SYSTEMS

CHAPTER 2

ARCHITECTURE OF MULTIPLE MULT-PROCESSOR SYSTEMS

2.1 INTRODUCTION

Current low cost-scale integrated microprocessors offer the potential of cost-effective multiple microprocessor systems. Advantages that can be gained by these systems include high throughput, improved real-time response, better availability/reliability and modular expansion. Unfortunately, the design techniques, structures and organisations of multi-processor systems are not well defined. A host of problems including process partitioning into parallel tasks, allocating tasks, sequencing and interaction between processors, controlling system resources and overcoming the physical and architectural limits of microprocessors must be thoroughly researched before implementation progress can be made. To provide solutions and design guidelines for multi-processor and distributed processor systems incorporating available large-scale integrated microprocessors, existing microprocessor architectures, organisations and strategies have been analysed to derive those characteristics which are mandatory for workable multiple microprocessor systems.

2.2 DESIGN ISSUES

There are a number of factors that influence the design of a multi-processor system. Emphasis to each one of them

would be given according to the goals, and objectives of the system designer, although these factors are interrelated. These factors are important in making a multiple processor system an effective computing machine and are involved in optimising the architecture to the particular application.

2.2.1 Task Distribution

The logical distribution of tasks is the relationships between the various tasks that the system is expected to perform. In a traditional computer system where the hardware is fixed, the logical distribution of these tasks affects only the structure of the software. In multi-processor systems however, it is possible to allocate processes to different processors. These networks, in effect, replace the multi-processing software of earlier computer systems with hardware. The fact that the jobs can be now done concurrently compensates for the lower performance characteristics of the components. One interrupt free method of dispatching processors to the data communication tasks could be done similarly to the mailbox approach used in Pluribus multi-processor system[9]. In this case processors are allocated to tasks on the basis of pending task priority, with all tasks running to completion on the allocated processor without interruption. At completion, the processor is re-allocated to the highest priority task requiring service. This, of course, eliminates context switching overhead but also puts constraints on

the hardware/software. All tasks must be executed in a time shorter than the overrun time of those tasks requiring service.

The degree of interaction between tasks also defines the organisation of the network. A network can be absolutely represented by a ngraph and if one associates a n control flow with the graph certain logical relationships emerge[10,11,12].

There is a distinction between the meaning of a graph in this context and its meaning in connection with traditional networks. In the latter case, various organisations are postulated in the hope of improving the mechanism for transmitting messages and the graph is used to describe the mechanism for routing messages. Since each processor executes its programs independently of the other tasks being executed, no attempt is made to associate any statement of program control with the graph. For multi-processor networks however, the graph is used to indicate the relationships between processors and hence provides a tool for identifying an isomorphic hardware structure. It is therefore more of a flow graph of program control. For example, the graph of a microprocessor ring network implies a sequential process where completion of a task in one processor initiates the execution of another processor. Alternatively a tree network implies the presence of a hierarchy or processes, where the completion

of several tasks in the terminal processors activates a process in the junction processor.

2.2.2 Relative Bandwidth Between Tasks

The relative bandwidth of the interfaces between these tasks:related to logical distribution of tasks,is the issue of bandwidth. Bandwidth is an important factor in deciding the physical distribution of processors. There are three common ways of handling interprocess communication:

(a) Serial Communication

(b) Parallel Data Bus

(c) Multiport Shared Memory.

Only serial communication allows any degree of physical separation. The other approaches imply the use of several processors combined in a single chasis. A recently advocated approach is a system in which several program processors are clustered around a central service facility. The cluster acts as a contemporary multi-programming system in hardware. The microprocessors in a service centre handle system processes while the program processors handle user requests. The service centre performs four functions: memory management, process management, file management and monitoring and protection. Each of these tasks could be maintained by a separate processor [3, 8, 11, 13, 14, 15].

2.2.3 Real Time Response

The specified response time associated with each task is the third issue. There are two components to this:

(a) The maximum amount of time allowed to recognise a condition (latency),and

(b) the total time allowed for a response.

The real time response of a mutli-processor system depends on the computational power of the individual microprocessor and specifically on their instruction speed and I/O capability. In the microprocessor used in this research, the 8-bit Intel 8008, there were limitations on both counts. The particular 8-bit microprocessor provides one interrupt to the CPU and,in this case,was not used for reasons explained later in the thesis. Although this may be extended by using additional peripheral chips, the cost of these chips must be weighed against the use of an additional microprocessor module. Moreover, the relatively slow execution times of these units caused by their 8-bit data path, limits their total response for multiple interrupts. Accordingly, a common mechanism for many high speed applications is to allocate one processor to each real time process.

It must be pointed out that newly introduced 16-bit microprocessors (TI 9900) would change the scope, approach and capabilities of these systems. However, we are concentrating on the 8-bit processors mainly and although

the design issues would remain the same for the advanced 16-bit processors, the system response would alter[3, 16 17, 18].

2.2.4 Reliability

The reliability of a network depends on the reliability of the nodes and the reliability of the communication system. In larger networks, work on reliability has centred on insuring the integrity of the network even in the event of a failure of one of the processors. This work was focused on hardware mechanisms that minimise the coupling between a processor and the network,and on the design of software which detects improper transmission by a faulty processor. In addition these networks should incorporate a number of encoding rules and network protocols which are intended to insure the validity of the data. The reliability of the nodes is generally a separate issue and is usually not considered in the design of the network. On the other hand, on a multi-processor system one would like to ensure the reliability of the total system. The most obvious solution to that (given the low cost of microprocessors) is to provide a "backup" microprocessor to every microprocessor in the system. This backup monitors the operation of the primary unit and in the case of failure either replaces it in the system or reports the failure. If that sounds suspicious as the backup can cause the system to fail, another approach associates a monitoring

function with one (or several) of the regular checks on the status of the other processor in the network and automatically detects nodes that are functioning improperly and removes them from the system. These nodes are then replaced with previously inactive nodes which have been included in the system with the specific purpose of acting as spares. A combination of the two above mentioned approaches can be also considered.

The security and reliability of the system is very important as in current practice in microprocessor environments long down-times cannot be tolerated[7].

2.2.5 Cost

Cost is clearly a significant issue in any design. Multi-processor systems differ from traditional networks not only in the total cost being dropped, but also in that the relative cost of the processor vs communications has shifted dramatically. This affects not only the range of applications but also the configuration of the network since we are dealing in an environment where in most cases the interface to any network will be a significant portion of module cost. In most cases though, it is felt that there will be a tendency to localise the network; to avoid any long distance method of communications and to reduce the bandwidth of any remove communication.

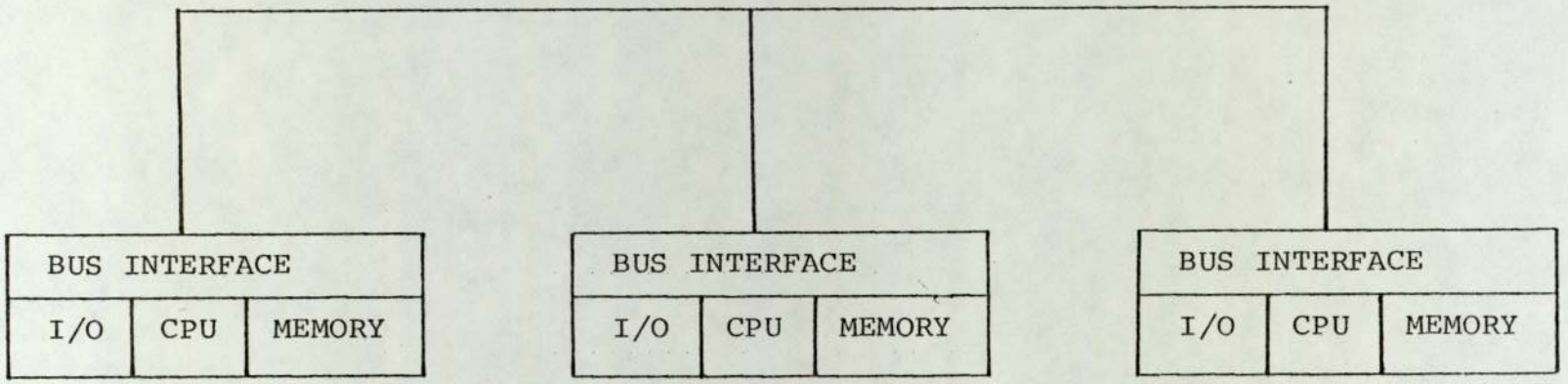
2.3 ANALYSIS OF THE ORGANISATIONAL STRUCTURE OF MULTI-PROCESSOR SYSTEMS

In the previous section we examined the design factors that influence the architecture of a multi-processor system. It was stressed that the flexibility, simplicity and capability of the microprocessors can provide almost any architectural design (within the limitations of microprocessors) to suit the objectives and goals of the system designer. In this section we examine some of the existing conventional architectures of multi-processor systems.

Distributed, parallel and pipeline architectures have been recognised as feasible approaches to provide high throughput systems[15, 19]

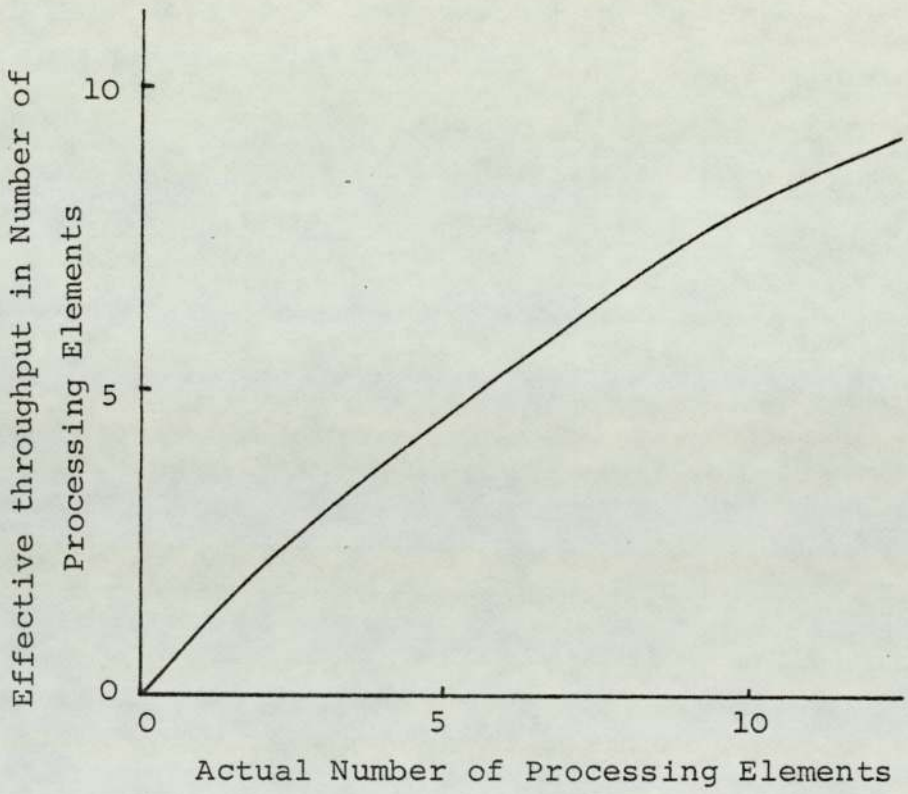
2.3.1 Distributed Processor Architecture

Distributed processing refers to a specific technique for interconnecting a number of processors. It utilises a Bus Interface Unit (BIU) to connect each processor to a single bus. There is little CPU involvement in the communication function. Addition of a processor will not affect the interface of those processors already in the system (Fig. 2.1). The primary advantage of this architecture is thought to be physical distribution and incremental expandability. An additional potential advantage of microprocessors in a distributed system is improved cost/performance (Fig. 2.2). The most obvious critical design issue is to determine whether or not a



Distributed Multi-Processor System

Fig. 2.1

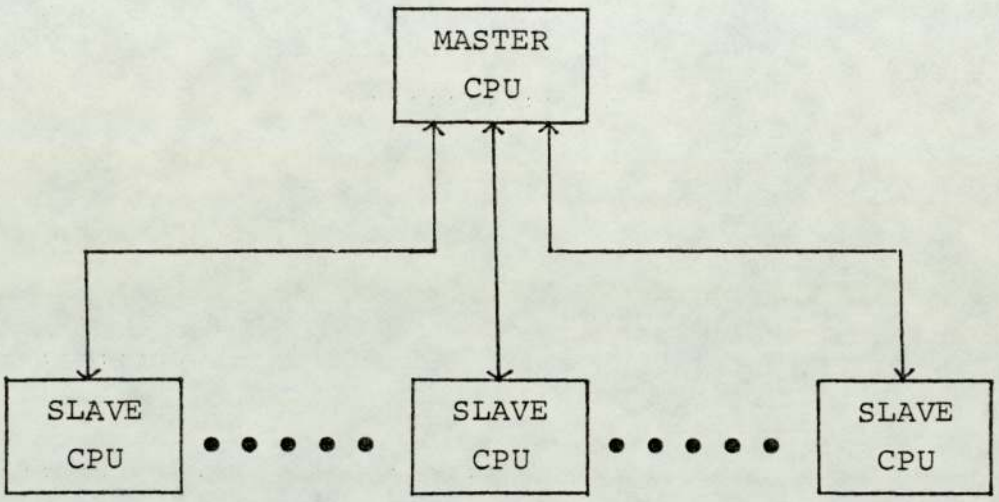


Distributed Processing Performance

Fig. 2.2

given application can be partitioned and executed concurrently. Tasks and their actions must be known in advance so that the system functions can be subdivided among the individual processing elements. This includes segmentation of software into dedicated program segments for each processor and assignment to controlled variables and devices to each processor. Such static allocation of tasks minimises program interaction which permits simplified development and debugging of individual program segments. Interprocessor communication is usually restricted to passing messages or data blocks through shared peripherals or serial communication links as opposed to a shared main memory. Failure though,of any processing element (CPU or I/O channels), may seriously degrade system performance as the system cannot dynamically shift tasks that have been assigned to the defective element.

A myriad of possible distributed intelligence microprocessor systems (DIMS) structures exists. The master-slave organisation[20] (Fig. 2.3), offers many advantages to multi-microprocessor systems. This system employs a single integrated operating system to dynamically allocate tasks as they are received. A resource allocation processor (master) can allocate tasks to processing modules (slaves) through a resource request table. PMs may be identical and capable of executing any task (symmetric) or may be pre-assigned to handle special functions (asymmetric). The symmetric multi-processor is used in



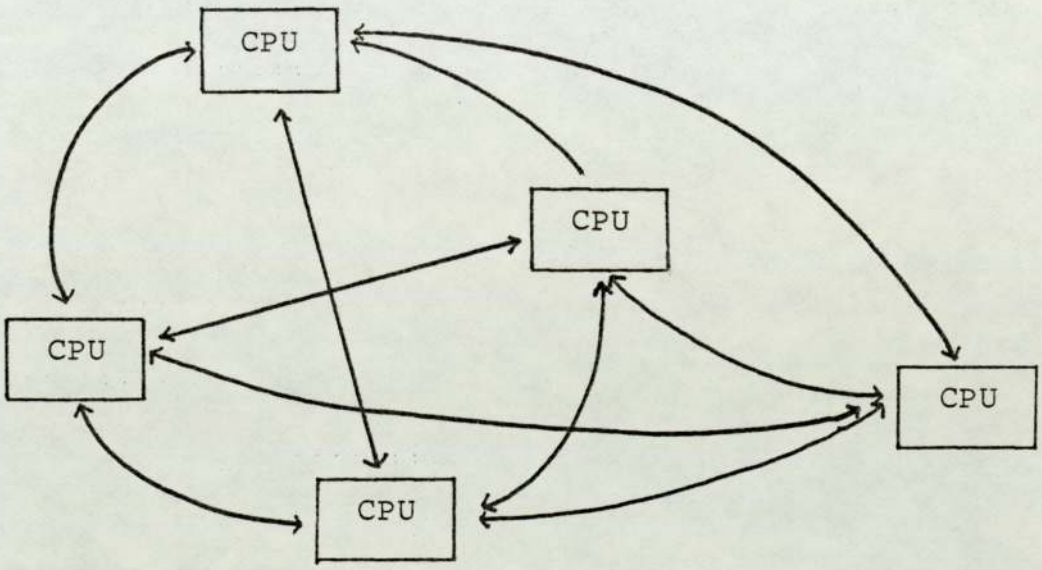
Master-Slave Multi-Processor Organisation

Fig. 2.3

a general purpose environment where processing requirements are constantly changing. Since PMs are equivalent, a given task can be re-assigned in the event of PM failure. By contrast the asymmetric multi-processor is composed of PMs specially configured for a set number of tasks. Incoming tasks must be queued to assigned PMs even though other PMs may be idle. Although this may decrease throughput it simplifies the operating system, which becomes a task scheduler and is relieved from the identification and allocation of parallel tasks. As a system becomes more asymmetric more tasks must be allocated to specific PMs and portions of the operating system becomes more individualistic.

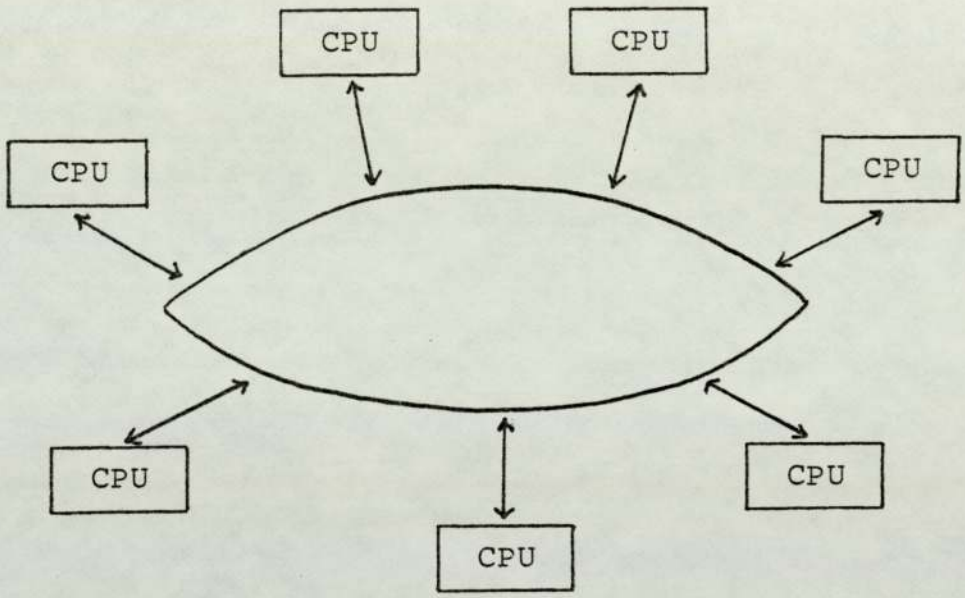
Another possible formation is illustrated in Fig. 2.4. of a master-master structure in which any CPU can communicate with any other CPU. In this organisation all the CPU's must support compatible interprocessor interfaces and I/O instructions. This organisation may well be effective for large communication networks. However, it may not be suitable for multi-microprocessor systems where the tasks to be performed by specific CPU's may vary drastically.

Yet another possible organisation is the ring structure illustrated in Fig. 2.5. In this organisation though, if the information bus is also needed by the individual CPU's for their own processing, severe contention problems will occur with a resulting degradation in the performance of



Master-Master Multi-Processor Organisation

Fig. 2.4



Multi-Microprocessor Ring Structure

Fig. 2.5

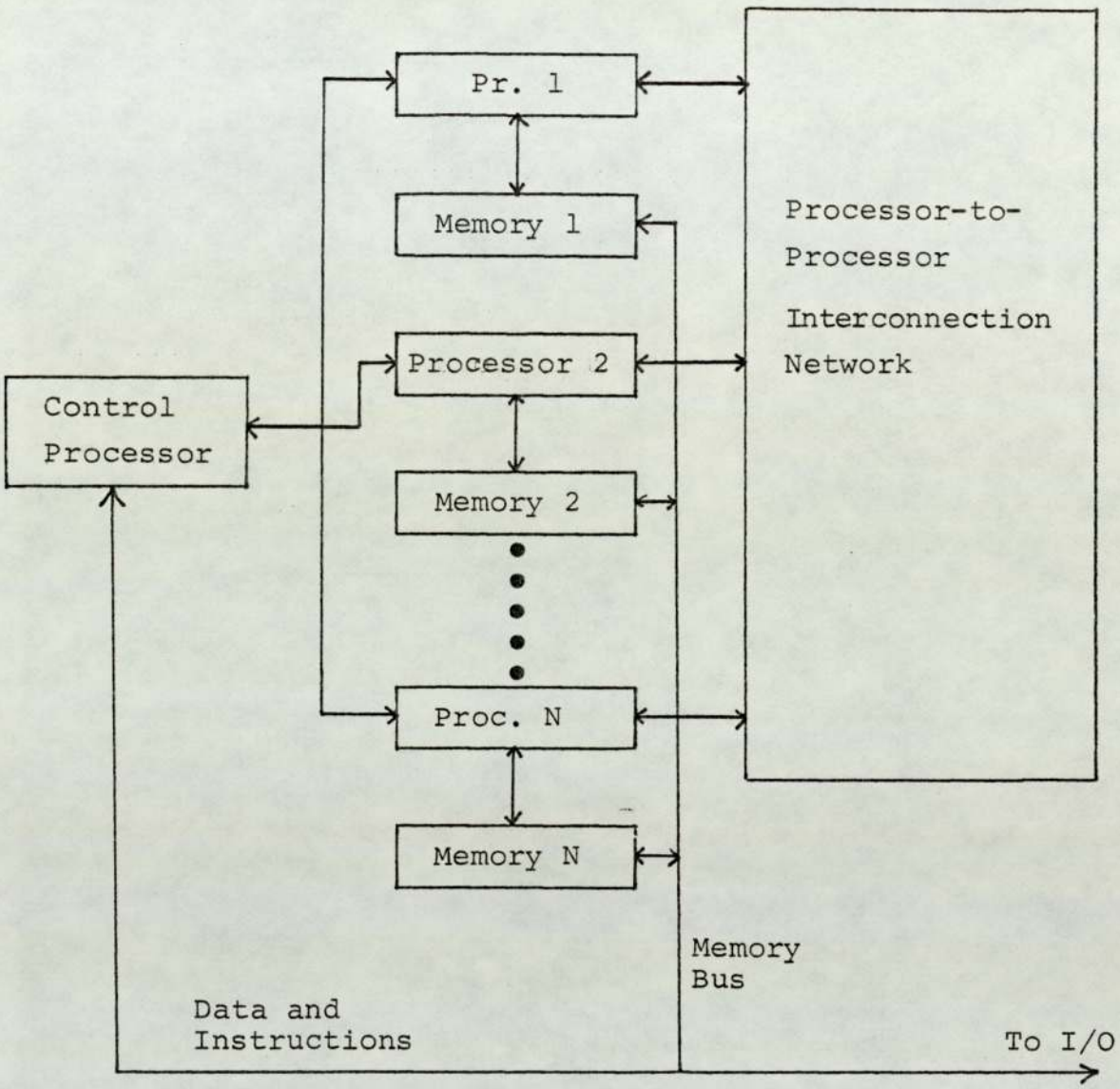
the overall system.

Distributed multi-processor organisations could be applicable to avionics slip-board control command and control and weapon control functions as have been considered as being amenable to partitioning.

2.3.2 Parallel Processor Architecture

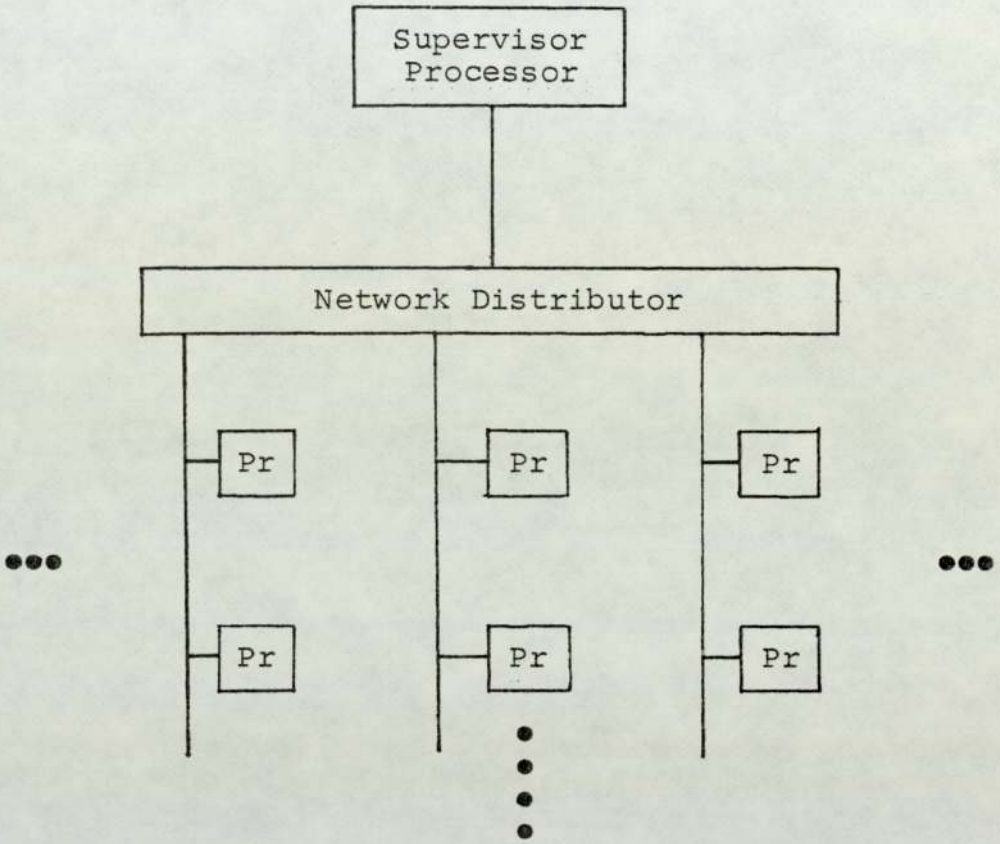
Parallel processing is the concurrent processing of two or more portions of the same system algorithm by two or more processing units. This can occur at the task, sub task, instruction stream or data set level. Two organisations of parallel multiple processors can be identified; the single instruction multiple data (SIMD) and the multiple instruction multiple data (MIMD).

In SIMD architectures, a single control unit fetches and decodes instructions. The instruction is executed in the control unit itself or its broadcast to other processing elements (Fig. 2.6). One subclass of the SIMD architecture the 'array processors', where instructions manipulate vectors of data simultaneously and the control unit has limited capability,appears to be cost-effective for very specific applications(Fig.2.7). Rationale for this organisation is the high throughput obtained by simultaneous (parallel) operations of processors on different data streams. Computations must be describable by vector instructions with many identical operations in



Block Diagram of an SIMD Micro-computer

Fig. 2.6



Block Diagram of an Array Processor

Fig. 2.7

action simultaneously on different data; high speed data routing between processors is necessary; and operands that are manipulated simultaneously must be fetched simultaneously. Applications can be in weather predication, air traffic control, radar signal processing or any high speed vector computation.

The MIMD architecture achieves parallelism by performing independent tasks on separate data sets concurrently and combining results of the execution of the independent tasks. To attain high efficiency, proper synchronisation of processors and allocation of tasks is necessary to balance the processing load.

2.3.3 Miscellaneous Architectures

The pipeline architecture consists of a number of ALUs, cascaded and inter-connected based on a specific algorithm. The architecture is discussed because it demonstrates one of the primary advantages of microprocessors: simplification of the design process itself. The application for which the pipeline was designed is signal processing. The real advantage found in this architecture is the ease of the design, because of the use of assembly language.

Multi-processor systems could be further classified according to the degree of interaction between processing modules (CPU's or I/O channels), often called coupling[15, 21]. We should add at this point that processors passing

data through shared memory are sometimes considered to be tightly coupled, although some designers disagree with this classification. Usually there is no program interaction between loosely coupled processors although they may share read/write memory to pass information. When the system is to consist of numerous small modules for general purpose applications, the connections are necessarily of the loose type. In this case the processors should be able to access memory with as much sharing as possible, communicate with one another through shared resources with minimum contention problems, and be capable of dynamic configuration in the event of PM failure. This type of system requires high throughput and/or high availability. The author feels that these systems will eventually replace many minicomputers in data based inquiry/response and in real time control applications.

This production is advocated by reasoning that:

(a) Inherent flexibility permits small increments of growth with minimum system redesign (extensibility).

(b) Ability to dynamically allocate tasks to balance the processing load improves throughput and real-time response.

(c) Ability to dynamically reconfigure PMs in the event of failure.

(d) LSI is cost-effective and a multi-processor system can overcome the ultimate physical limits of LSI

for high performance applications[15,21,22].

One or more of these items must be chosen as design goals for a microprocessor based multi-processor system.

2.4 CONCLUSIONS

The microprocessor revolution has made possible the economical de-centralisation of computing power. This has been achieved not necessarily by making systems with improved cost/performance, but by making microprocessor control of many functions economical and practical. The microprocessor has made it economically possible to introduce processor control to a host of new applications, and thus to the diversity of architectural designs. Multi-microprocessor structures are largely effective in situations where the tasks to be performed can be effectively and efficiently partitioned. This would give rise to further diversity of architectural designs as I/O processing capability is improved coupled with improved reliability and fail-safe features. An additional benefit resulting from the effective partitioning of tasks in a multi-CPU system is that the software, by being partitioned into several relatively independent packages, is much simpler and runs more efficiently. This is especially effective in a system supporting many interrupting devices.

CHAPTER 3

CONTROL OF SYSTEM RESOURCES IN A MULTIPROCESSOR SYSTEM

CHAPTER 3

CONTROL OF SYSTEM RESOURCES IN A MULTIPROCESSOR SYSTEM

3.1 INTRODUCTION

Allocation of tasks and sychronisation of micro-processors are the most serious problems in the design and operation of multiple-microprocessor systems. They involve identification of a parallel process, partitioning the process into subprocesses or tasks, establishing a priority scheme for the tasks, assigning tasks among various microprocessors, sychronising them, and providing some means to dynamically reassign a task in the event of PM failure. Although solutions to these problems have been proposed the implementation is very difficult. One of them is by expressing potential parallelism in the coding via a WAIT-SIGNAL. This approach was used in this research project. It is obvious that the smaller the number of microprocessors, the easier the process.

Processes or tasks operating in a micro-multi-processor system share a number of resources to improve performance. Resources include hardware (processors, memory, I/O channels, registers, buses) and software (programs, data files, buffers, variables). The more shared resources that are available, the greater the control required for the allocation and resolution of tasks. Too much sharing results in complex control structures and task conflicts. This can produce

low throughput or deadlock as two or more tasks are waiting for resources that have already been assigned to each other.

3.2 HARDWARE RESOURCE CONTROL

Arbitration, flag test and set and interrupts are the most common methods of hardware resource control.

Arbiters

An arbiter accepts requests from PMs (active elements), resolves contention and alerts the elements of its decision.

A centralized arbiter consists of a single self-contained hardware unit. Intel has designed a custom 'bus controller' chip for this function on the SBC 80/CO microcomputer board[23], Widdows has developed a 'lBus arbiter' for the MINERVA system[24] and Reyling has proposed a resource allocation microprocessor[7].

A decentralized aribter is one in which control logic is distributed throughout the active elements connected to a shared resource. The arbitration method includes daisy chaining, priority encoding, and polling asynchronous requests, (flags and interrupts). Choice of method depends on simplicity, device servicing requirements, expandability, susceptibility to failure control line limitations and controller speed[25]. Arbiter speed should be such that the

overhead to access a device is only a fraction of the time spent using the device. For example, a high speed cycle-shared memory requires a hardware arbiter, while a block of shared memory can be allocated by a microprocessor arbiter.

Status Flags

Conflicts over shared memory and I/O can be resolved via the flag-and-set procedure. The requesting processor tests the states of the flag, which is a resource busy indicator. If busy, the microprocessor must wait before obtaining the resource. If not busy, the flag is set to busy during resource access, and then reset when the microprocessor is finished with the resource. Simultaneous requests for a resource must result in only one processor gaining access. Since requests for a shared resource occur asynchronously, care must be taken that more than one processor does not gain control of the resource. For this reason the test and set operation must be indivisible.

If memory is used for the status flag, it must be capable of a read-modify-write cycle before permitting further accesses. This requires a lock on the memory address. Although it is easier to lock a block of memory than a specific address[25] in that case the remainder of the module is unavailable to other processors. For this reason status bits are sometimes implemented as a set of dedicated external registers that perform the

read-modify-write cycle themselves.

Interrupts

Interrupts can be used to service internal processor errors, clock signals, external devices or to sychronise interprocessor communications (shared memory). Servicing interrupts in a micro-multi-processor system is usually assigned to the originating microprocessor. External devices may be preassigned to individual processors or dynamically directed to whatever processor is best equipped for service. This real-time assignment can be done through a centralized hardware arbiter, a dedicated high speed processor or by individual microprocessors. Assignment is made on the basis of servicing capability, availability, task allocations, and software priorities of each processor. Interprocessor communications can be sychronised by passing an interrupt request signal from one processor to another, such that requests for each can be wire-ORed onto one interrupt level dedicated to interprocessor communications. A prioritized vector, corresponding to the highest priority processor requesting the bus, can be placed on the data bus when the interrupt has been acknowledged. Handling external device interrupts is perhaps the most critical decision in implementing the device interface to the micro-multi-processor systems. Microprocessors with highly advanced interrupts facilities and priority levels are requested.

The ability of the microprocessor to do useful work while waiting for availability of system resources is very valuable. The microprocessor could then perform background processing of tasks not requiring system resources. This is particularly important in applications where the microprocessor must be able to respond in real time to local interrupts.

3.3 CHARACTERISTICS OF PROCESSING MODULES

The composition of Processing Modules (PMs) depends to a large extent on system bus structure (interconnection topology), interprocessor communications, and the number and type of shared resources. In general, the PM includes a CPU (microprocessor chip, clock, bus control, buffers,) local or private memory for instructions and data storage, system bus interface circuitry, memory map hardware, interrupt handling logic, and I/O device controllers for private I/O. It is also possible for each PM to share memory with other PMs. In cases where system bus width is significantly greater than microprocessor word length, an instruction stack can be used to lighten system bus loading[24]. One critical area of PM is the system bus interface logic. Flexibility and performance of the interconnection topology are directly proportional to the complexity and expense of this interface logic. If heterogeneous microprocessors are used, interface logic

will be unique for each processor. In particular 8 and 16-bit processors may communicate over the same bus, requiring distinct interfaces for each microprocessor.

Memory map hardware is used to translate addresses provided by the microprocessor into addresses in physical memory (both prive and shared). In a parallel computation environment, precise memory requirements for a group of concurrently executing programs cannot be predicted ahead of time. As a result, programs and data must be moved and/or compacted to make room for additional items. The memory map facilitates dynamic variation of physical locations during program execution without actually moving the locations. If data are to be moved, the map function is changed to reflect a new physical address assignment. The memory map also provides a convenient means for two or more programs to share data. References to shared data are mapped onto the same physical addresses, while references to private data are mapped into distinct locations for each PM. It is sometimes possible to use software to achieve the memory map function. To do this, extensive use is made of indirect addressing through indirect pointers in microprocessor register or in read/write memory, and indexed addressing.

3.4 INTERCONNECTION OF FUNCTIONAL MODULES

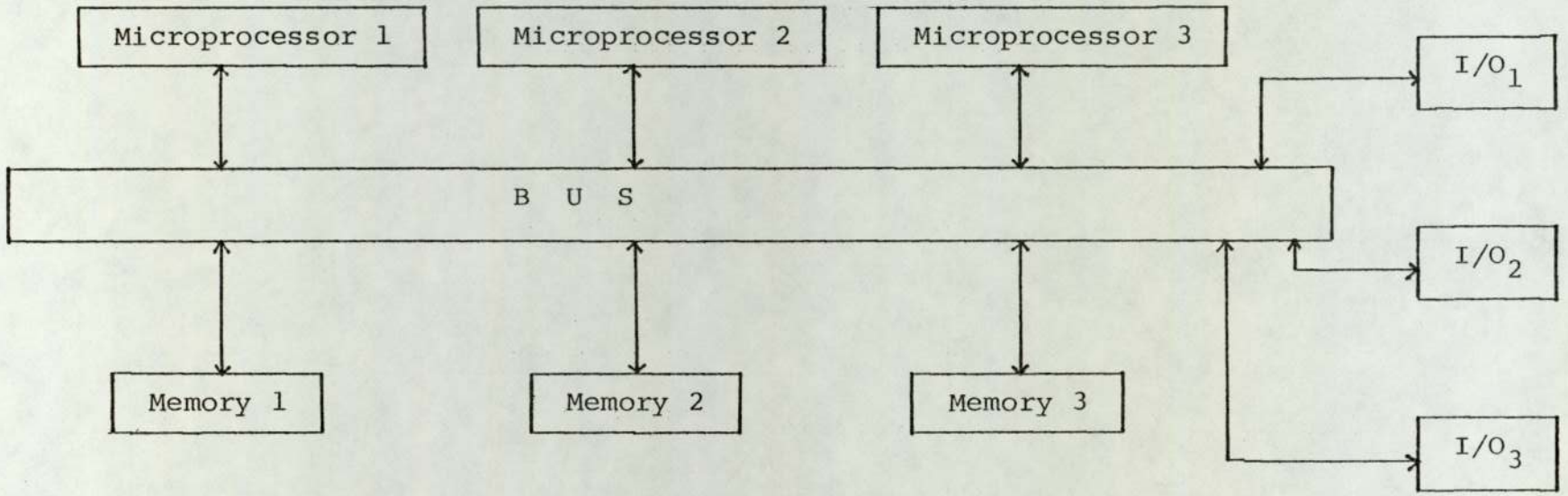
Some organisations for interconnection of PMs, shared memory, and shared I/O are the time-shared common bus, multi-

bus/multiport memory and crossbar switch[2,1,26].

The single time-shared system bus shown in Fig. 3.1 is a shared resource, therefore a means must be provided to resolve contention (fixed priorities, first-in, first-out, queues). Interference between PMs requesting the bus depends on the length and frequency of PM bus cycles, memory and I/O cycle times and the number of PMs that share the busy (system capacity). The lower the ratio of bus cycles required by an individual PM to the total number of cycles available, the higher the system throughput. For this reason, private memory and private I/O are highly advantageous. Total system capacity is limited by the bus transfer rate. Disadvantages of this bus structure are that system expansion increases contention which degrades throughput and increases logic.

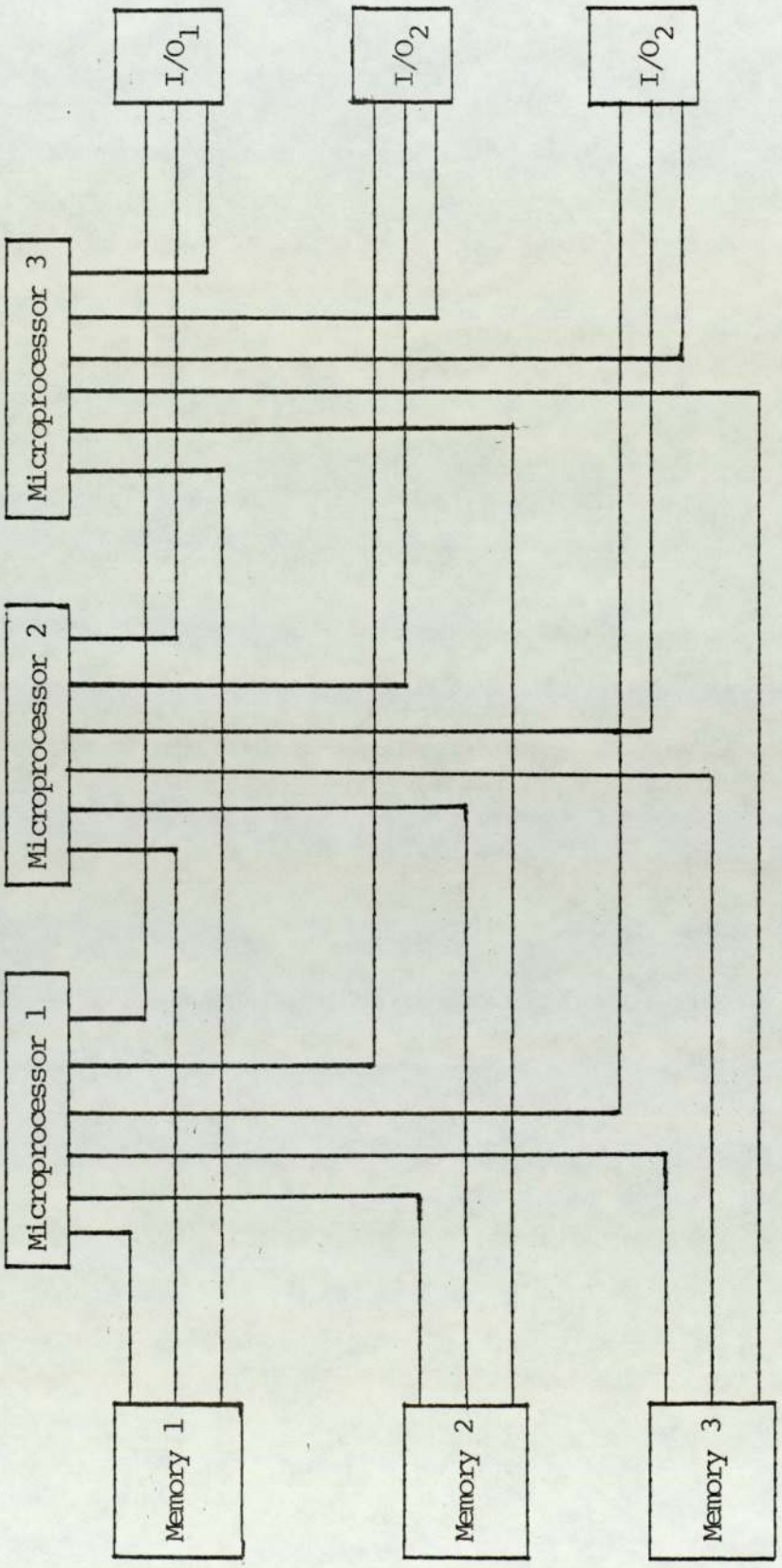
Multiported systems (Fig. 3.2) employ multiple dedicated buses that are connected between PMs, shared memory and shared I/O. Each of the latter two (passive) elements have multiple ports which provide excellent throughput, bus contention logic must be built into each passive element to acknowledge or hold PMs competing for the resource.

In the crossbar switch organisation any passive element can be connected to any PM for the complete duration of a data transfer through the crossbar matrix. (Fig. 3.3). This scheme can produce high system throughput.



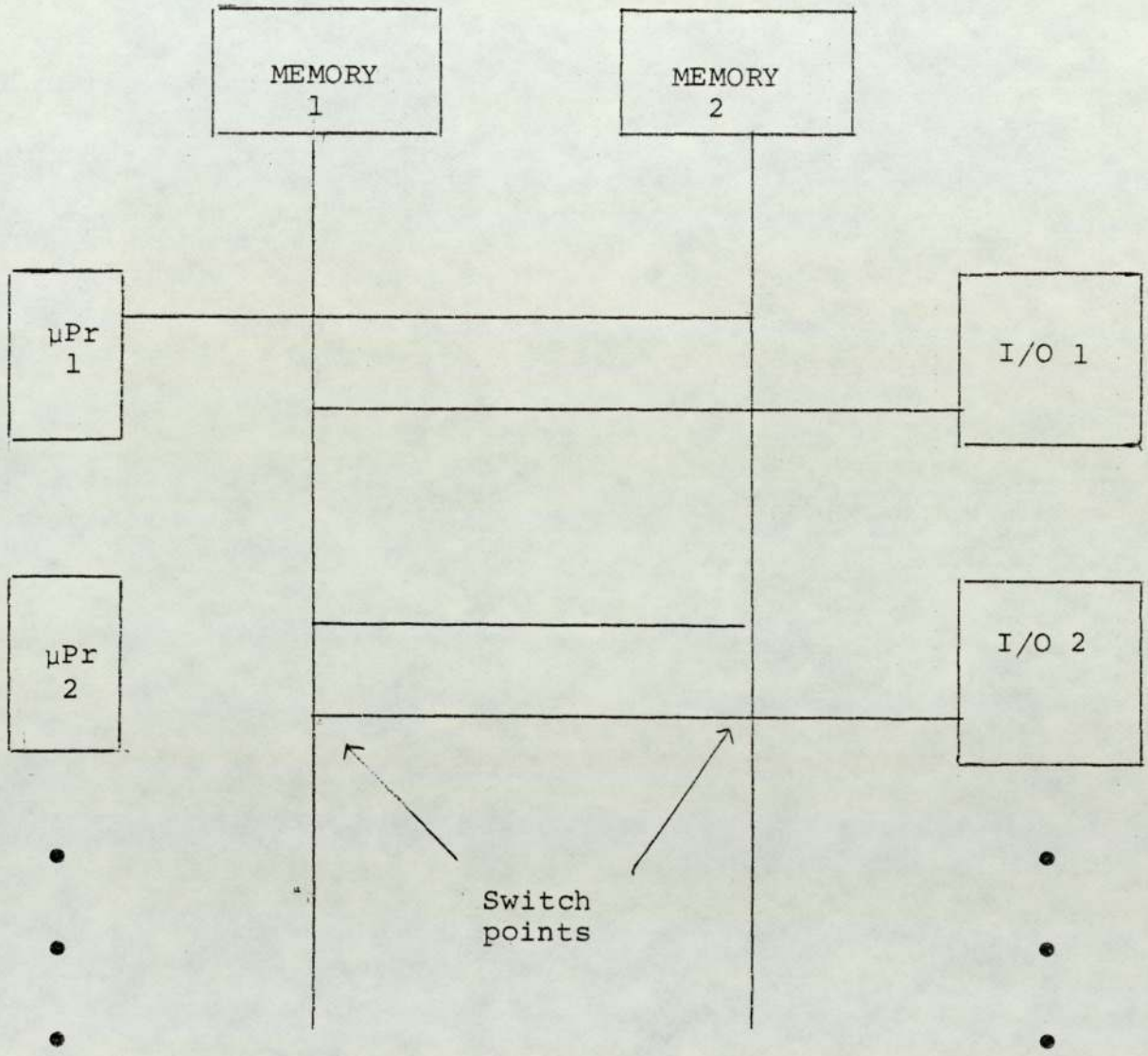
Single Time-Shared Bus

Fig. 3.1



Multiport System Bus Configuration

Fig. 3.2



Crossbar Switch Organisation

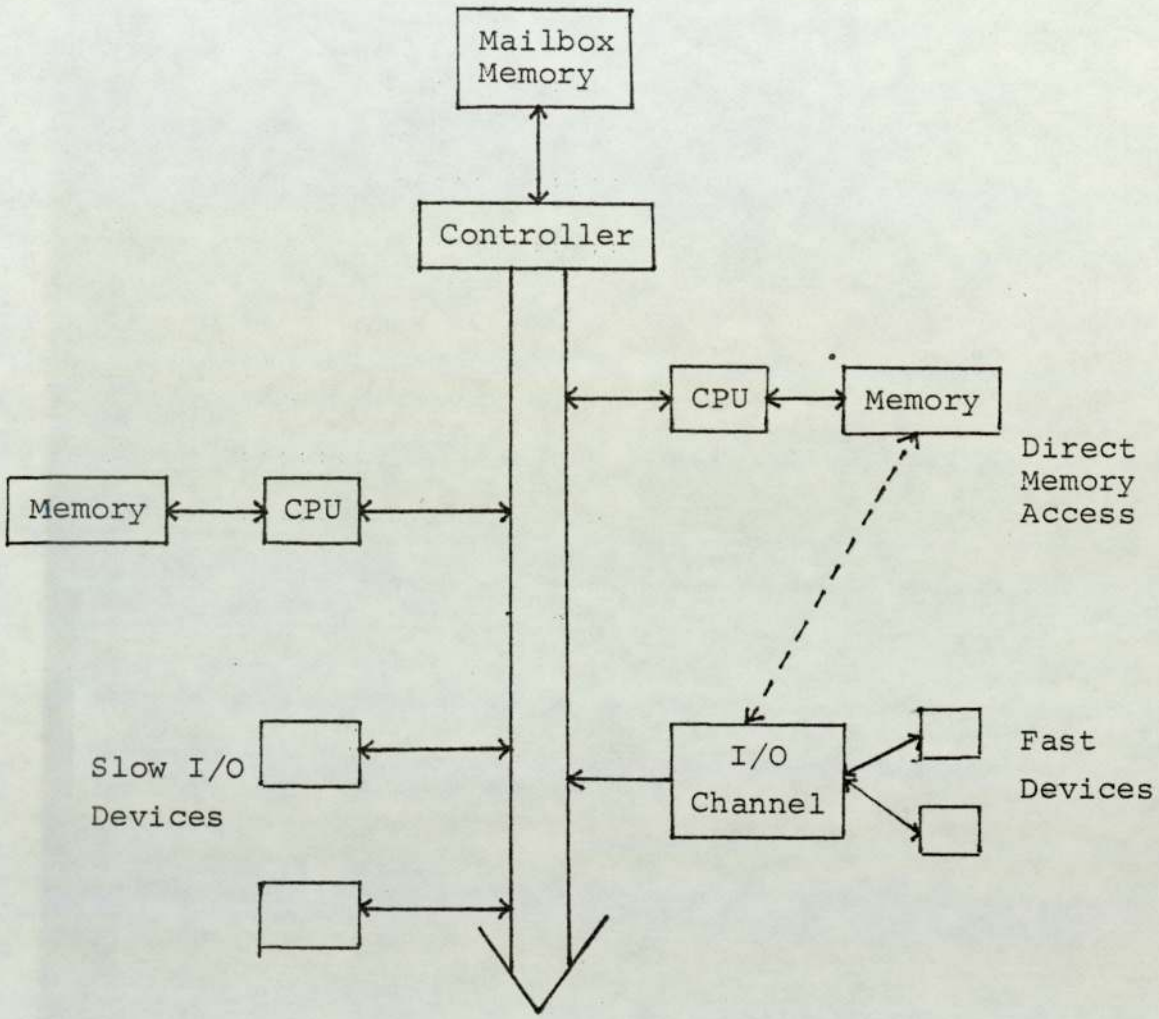
Fig. 3.3

Other bus structures are possible, either as a combination of the above mentioned, or others depending on the tasks the system performs.

3.5 INTERPROCESSOR COMMUNICATIONS

In the multiple instruction multiple data organisations, a major interprocessor communication media is the mailbox memory (Fig. 3.4). Processors in this context are central processing units (CPUs), or I/O direct memory access (DMA) channels. Mailbox memory is a shared resource consisting of messages, data files, request blocks or queues. The sending processor structures information and places it in the mailbox. The receiving processor 'looks' in the mailbox, to indicate that there is something in the mailbox for the recipient processor[20]. Flags, jump conditions, interrupts or special instructions can be used to alert the receiving processor of information to be taken. A status bit, residing in memory or external hardware, is generally used to indicate the condition of the mailbox.

Ford[27] has proposed PUT and GET instructions for a mailbox controller with BLOCK and WAKEUP signals from the controller to processors on the system bus. This controller can process only one PUT or GET signal at any time. This competition for mailbox memory must be resolved by arbitration.



Mailbox Memory Organisation

Fig. 3.4

3.6 CONCLUSIONS

Ability to identify and isolate failures to achieve fail-safe capabilities is often a prime motivation for a multiprocessor design. The degree of fault detection, task reassignment, and duplexing of functional units depends on the applications requirements. Failures can be detected using either parity on the system bus for both address and data, or local diagnosis for each PM in private read-only (ROM) or PROM memories, or by protected memory to detect address out of range, or by invalid op code, or other invalid condition detection.

The architecture most likely to be employed in the very near future will consist of asymmetric PMs, a single bus with a centralized arbiter, or a multiport bus, dedicated assignment of interrupts, interprocessor communication through shared memories, hardware flags, and limited failure recovery. The operating system will structure tasks into request blocks and queue them to a preassigned Processing Module. Two to four microprocessors with the same number of I/O channels are, one feels, a reasonable number for prototype multi-processor system.

CHAPTER 4

PERFORMANCE AND COST OF MULTIPROCESSOR SYSTEMS

CHAPTER 4

PERFORMANCE AND COST OF MULTI-PROCESSOR SYSTEMS

4.1 SYSTEM THROUGHPUT

Determination of multi-processor systems throughput as a function of the number of microprocessors required, is a primary concern. In the general configuration of a multiprocessor system (Fig. 4.1) system throughput (T_s) is defined as the number of instructions executed per second by the system. Maximum value of T_s would be equal to the number of processors times the maximum throughput per processor, if not for bus interference (all memories and peripheral devices are accessed over a single bus). System throughput is determined by the number of processors (N) in the system, throughput of an individual processor when there is no bus interference (T_p) and the amount of bus interference that actually exists in the system. (The effects of interference only when the bus is used for making single-word transfers are considered here, contention for multiple-word transfer units or I/O devices that effect throughput may be considered in a analogous to the following manner).

When bus interferencence occurs, one or more processors must wait for the bus to become free, reducing

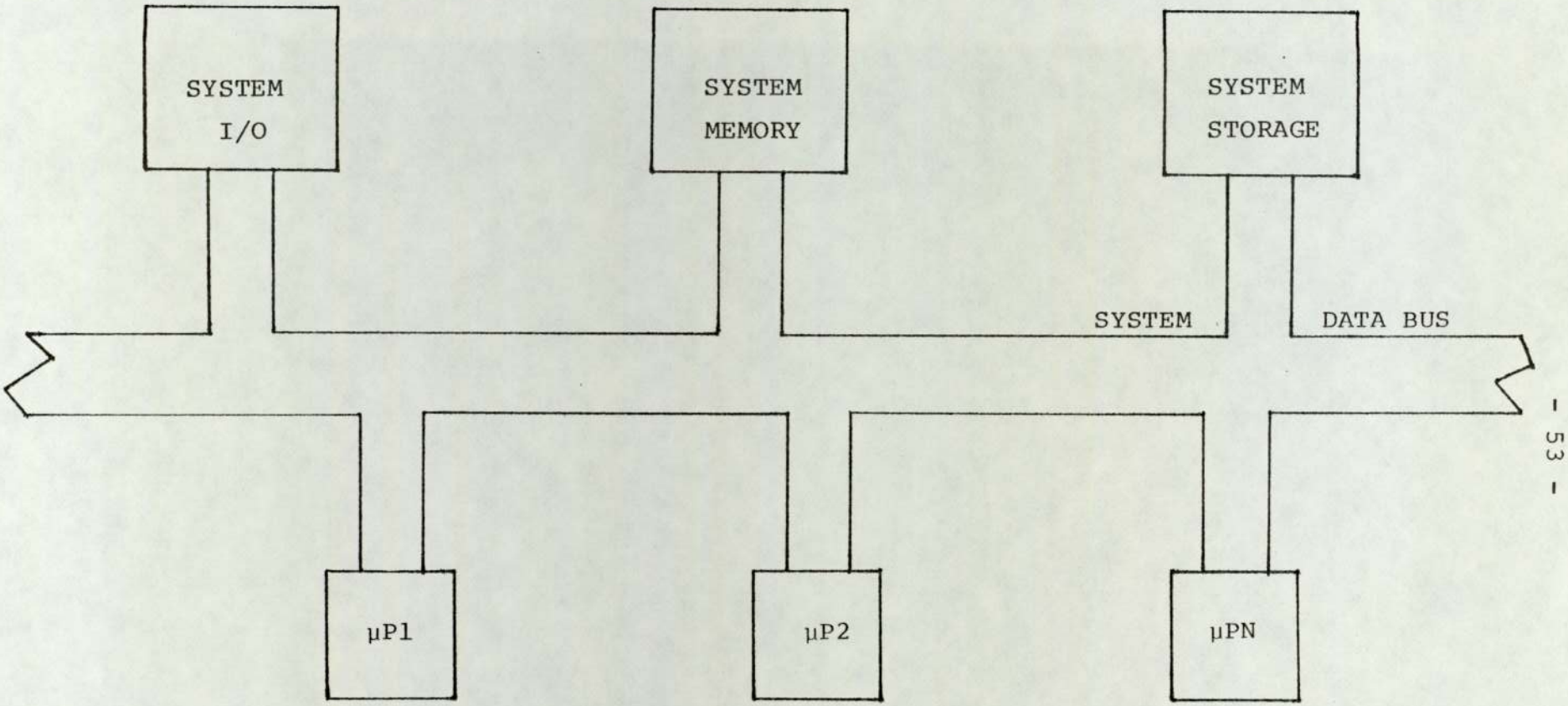


Fig. 4.1 GENERAL CONFIGURATION OF MULTIPLE PROCESSOR SYSTEM.

throughput of individual processors and therefore of the entire system. The amount of bus interference in a system is a function of the bus utilization requirements of individual processors. Bus utilization (β) is defined as the fraction of available bus cycles required by an individual processor. The value of β for a given system is determined primarily by μP instruction execution time and memory read cycle time. For the system used in the research and memories, β is likely to be in the range of 0.1 to 0.5[7].

For given values of N, T_p and β the max., average and min. values of system throughput T_s may be found. Consider β = 1/3 and three (N=3) processors in the system, then max. throughput will occur if each processor uses only every third cycle. This can occur only if the probability distribution of bus reference intervals is P_o, 1, 2=0, P_3=1, $P_{4-\alpha}$ = 0 (P_i is the probability of a bus reference every i bus cycles). The average bus utilization is determined from the probability distribution by the formula[7,3,11]

$$\beta = \frac{1}{\sum_{i=1}^{\infty} ixP_i}$$

For the case of P_3=1 N=3, the processors will sychronize with each other and no interference will occur. If N is greater than 3, throughput will be limited by

the bus capacity to $3T_p$. For N less than or equal to 3 there will be no interference (after sychronization) and T_s will equal NT_p. (Fig. 4.2a). By extending this reasoning it may be said that if $\beta = 1/I$, max. T_s occurs for $P_I=1$, $P_i(i\neq I)=0$.

The min. value of T_s can be considered also. The worst case possible would be if all processors accessing the bus always had to wait for N-1 other processors before gaining access to the bus (assuming the processor waiting longest has highest priority). In this case all bus reference intervals would be increased in length by N-1 bus cycle intervals. The decrease in throughput could be derived as

Ratio = R = ration of throughput with maximum interference to throughput with no interference.

$$= \frac{\sum_{i=1}^{\infty} iPi}{\sum_{}^{\infty} (i+N-1)P_i} = \frac{1/\beta}{1/\beta+N-1} = \frac{1}{1+\beta(N-1)}$$

minimum T_s = (throughput with no interference) x R.

$$= NTpR = \frac{NTp}{1+\beta(N-1)}$$

This minimum value of throughput may be used to determine the range of possible throughputs and is shown in Fig. 4.2a for $\beta= 1/3$ and 4.2b for $\beta= 1/10$. It could be noticed that

even in the case of maximum interference there is an increase in T_s with N. It should be said that the throughput data in Fig. 4.2a and b are not necessarily a direct indication of the useful work throughput for a system. Expanding the number of microprocessors in a system will increase its processing overhead if partitioning of that system into a larger number of functions augments the supervisory problems of indicial function co-ordination, or if the individual microprocessors are not efficiently utilized and therefore load down the bus when not performing useful tasks. In other cases overheads per processor may decrease as N increases, when the number of microprocessors available allows a more natural partitioning of work functions.

4.2 SYSTEM COST

Having examined the potential increase in multiprocessor systems throughput, it is now possible to find out how much this added throughput will cost. This is a strong function of the ratio of the cost of system resources, designated C_R, which includes memory and peripheral devices, to the cost of an individual microprocessor, designated C_P, which includes CPUs, data bus interface circuitry, power supply cost, buffers. Total cost is then $C_R + NC_P$. Let us consider two systems, one with $C_P = C_{R/5}$ and the other $C_P = C_R/30$. (T_p assumed to be the same in both cases). Considering costs vs throughput (Fig. 4.3) it is clear the advantage of a high ratio of C_R to C_P as well as low value for β. As N approaches $1/\beta$

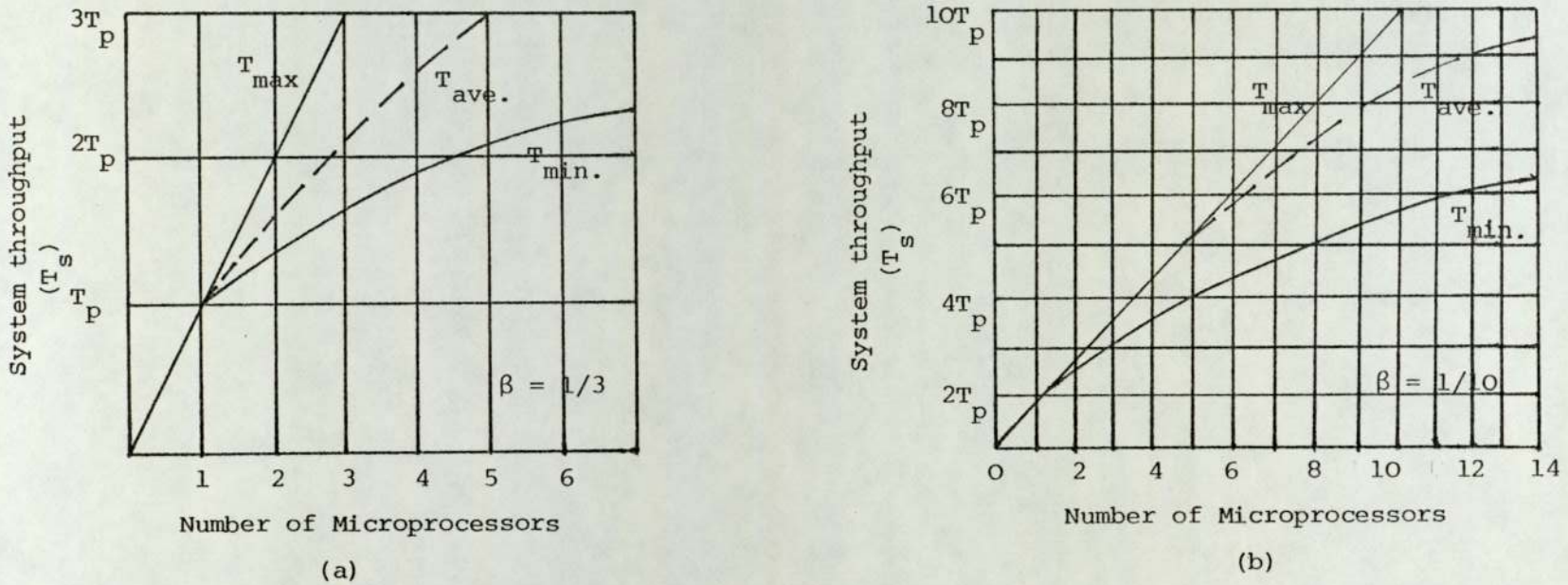
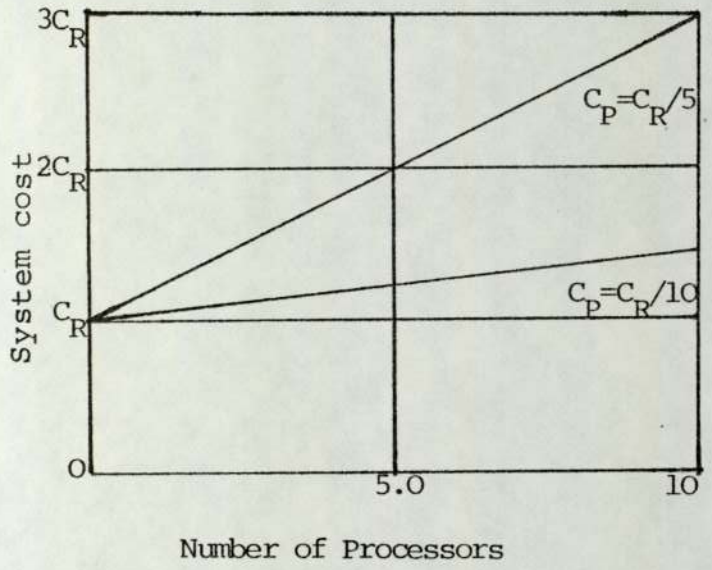
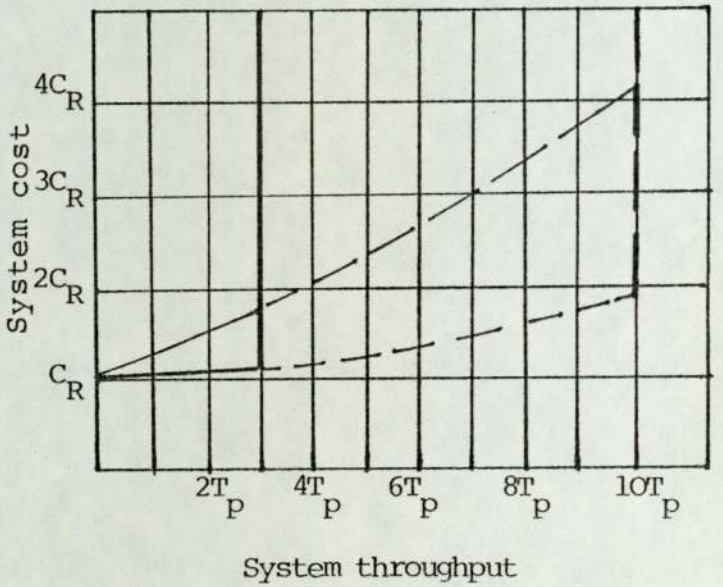


Fig. 4.2 PLOT OF MINIMUM, MAXIMUM, and AVERAGE MULTI-PROCESSOR SYSTEM THROUGHPUT (T_p = throughput of a single microprocessor vs number of processors).
In (a) each processor requires one third of the available bus cycles, while in (b) each processor requires only one-tenth of the available cycles.





——— = 1/3
----- = 1/10
C_p = $C_R/5$ x
C_p = $C_R/30$ o

Fig. 4.3 RELATIVE SYSTEM COST vs NUMBER OF PROCESSORS (a) AND SYSTEM THROUGHPUT (b)

system cost increases rapidly. The minimum value of C_s/T_s can be calculated in order to determine the optimum number of microprocessors in a system. For microprocessors the ratio of C_P to C_R is typically low since the current cost of a complete microprocessor is low, compared with memory, mass storage, peripherals[7]. A means of decreasing the apparent value of β is to provide some memory local to each microprocessor. This memory is accessible without utilizing the system data bus and contains data utilized only by that particular microprocessor.

4.3 SYSTEM CONTROL

The use of microprocessors in a multi-processor system organisation requires the provision of special control functions in addition to those of a stand-alone system. Control mechanisms are required for interprocessor communication and resource allocation. One way to handle resource allocation is to provide a flag which indicates whether a resource is available or in use by a microprocessor. A microprocessor requiring the use of a resource checks the status flag. If the flag is in the 'not busy' state the microprocessor sets it to the 'busy' state and uses the resource resetting the status flag when done; if the flag is 'busy' the microprocessor waits until the 'not busy' state is indicated.

The status flag approach thus requires a test and set

(TAS)[28] and a reset operation. One restriction on TAS is that it be indivisible with respect to other microprocessors; that is, if two microprocessor simultaneously execute TAS, only one should gain access to the resource. If the system data bus allows only read or write transactions with memory, and sequence of successive bus cycles cannot be dedicated to a single microprocessor, it will not be possible to provide an indivisible TAS routine. If the system bus allows a read/modify/write operation in a single bus cycle, TAS will be indivisible if implemented in a single bus cycle. For most microprocessors a standard instruction with this capability is not supplied. An additional property highly desirable for the TAS operation is a 'non-busy wait' capability, which allows the microprocessor to go into an idle state that is interrupted when a requested resource becomes available. This prevents the system data bus from becoming overloaded when several microprocessors are repeatedly testing a bus status flag. Often features may be desired for TAS operation. If a microprocessor has several different tasks it could work on, and each task requires a different set of resources, the ability to request one of several groups is useful. Also valuable in some applications is the ability of the microprocessor to perform background processing while waiting for availability of system resources. This is particularly important in applications where the microprocessor must

be able to respond in real time to local interrupts. When a microprocessor is initialized it executes the first instruction from a fixed location in memory. If this location is in system memory, all microprocessors execute the same instruction, and a method to direct each processor to its appropriate task must be devised. This problem is solved if local memories containing these addresses are used.

4.4 CONCLUSIONS

Multi-microprocessor networks can be configured economically to boost the service capabilities and reduce resource overheads of a system. However, it is clear that there is a limit to the number of microprocessors that can be connected to share a resource, without degenerating individual throughput and response. An assessment has been given on the system's behaviour. This should provide sufficient information to predetermine the relationship between resource utilization, the processor throughput and system's control.

CHAPTER 5

A MULTI-PROCESSOR SYSTEM USING THE INTEL 8008 MICROPROCESSOR

5.1 Introduction

5.2 Design Considerations

5.3 Design and Construction of the 8008 CPU System

5.4 Design and Contruction of the Shared Memory

CHAPTER 5

A MULTI-PROCESSOR SYSTEM USING THE INTEL 8008 MICROPROCESSOR

5.1 INTRODUCTION

The prime objective in this research is to study the effect of a processor to memory performance and in particular a number of microprocessors accessing a shared memory through a common bus. The microprocessors used were the INTEL 8008. The 8008 is characterised by a five state processor cycle, with each state requiring 2.8 μs. When the 8008 reads from memory to get the next instruction, it presents on its data bus during state T1 the lower eight bits of the desired address. It proceeds to state T2, where the upper six address bits and an indication that the 8008 wants to read appears on the bus. If by the end of T2 memory has not responded READY, indicating that the desired byte is being presented on the bus, the 8008 goes into the WAIT state, where it remains for as many four μsec periods as are necessary for memory to respond READY. In the event that memory responds READY before the end of T2, T3 is entered and the byte is brought into the CPU. If necessary states T4 and possibly T5 are used to execute the instructions. The maximum memory bandwidth capable of being utilised by the processor is eight bits every twelve μsecs.

5.2 DESIGN CONSIDERATIONS

In the design stage the capabilities, advantages and disadvantages of the 8008 were considered, as at the time of the research more advanced microprocessors were developed thus changing and improving the design constraints posed by the 8008 and offering greater design flexibility. Expandability of I/O features was considered in the design as it was desirable in a multi-processor system to permit connections with computers and terminals. Expandability of memory was another design feature that was considered as the requirements of the whole system could demand expansion of memory to provide greater buffer storage for any future additional links to the system. The question of local or global memories was also considered. In order for multiple microprocessors to operate effectively together it appeared at least two pre-requisites has to be met:

(1) minimal interference and dependence among the processors, and

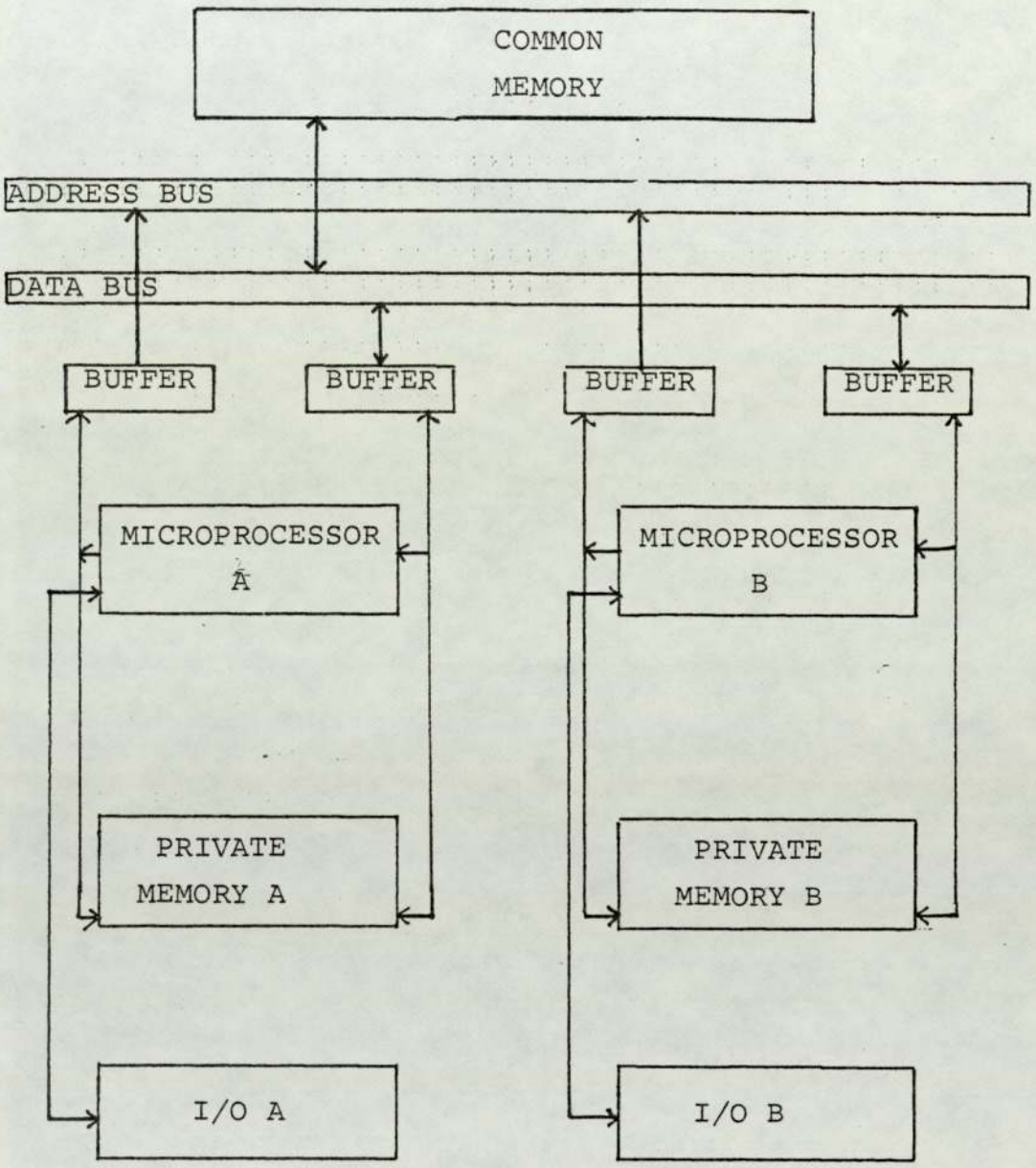
(2) a convenient low overhead interprocessor communications facility.

If a global memory was to be used it would require a large bandwidth to support simultaneous accesses by the 2, 3, 4 or more microprocessors[17,20].

Eventually, the design shown in Fig. 5.1 was prevailed. Every two processors to share a common memory. Each processor had its own I/O facilities and the memory system chosen allowed a considerable amount of local memory for each processor with a common memory shared by every other two microprocessors. The concept was to keep the core software programs (those executed most frequently) in the local memory to reduce the amount of common memory access contention among the processors. It was thought that the common memory should be used as a storage of information common to both processors.

The processor buses were designed to support the communication of the two processors through the common memory and the I/O buses were independent for the two processors. When both processors are trying to access the common memory, one has access to it and the other is forced to a WAIT state, until the bus is cleared for access. The microprocessors are identical and could perform any task. If 4 or 6 microprocessors were to be used with 2 or 3 common memories respectively, the distribution of tasks and software should be considered to improve reliability of the system and compensate for any break down of any of the microprocessors within the system.

One important factor that influenced this particular design was the concept of the priority resolution and



A Multi-processor System using the INTEL

8008 Microprocessor

Fig. 5.1

interrupt priority levels. Since the performance of the 8008 interrupts was not reliable, interrupts were not used and the priority logic in the common memory configuration was designed as explained in a latter section.

Thus the design with two processors sharing the common memory was preferred and in the case of a simultaneous access of the shared memory there is an arbitrary priority order. In the case that three processors were using the common memory without priority level accessing, the design would have been more complicated and the hardware connections would have caused considerable problems.

5.3 DESIGN AND CONSTRUCTION OF THE 8008 CPU SYSTEM

The CPU module is the Central Processor for the system. In this capacity the following control requirements are performed.

(a) The execution of program instruction, the control signal to RAM, PROM and I/O modules.

(b) All the necessary arithmetic, logical and data manipulation operations needed for program operation.

(c) Overall system timing.

The module itself contains an INTEL 8008 CPU chip, logic that supports the chip, 2K bytes of PROM and 1K bytes

of RAM memory. In addition to the processor chip, the module contains the following logical blocks.

(a) Timing generator

(b) Cycle decoder

(c) State decoder

(d) Bus logic

(e) Address latches

(f) I/O latches

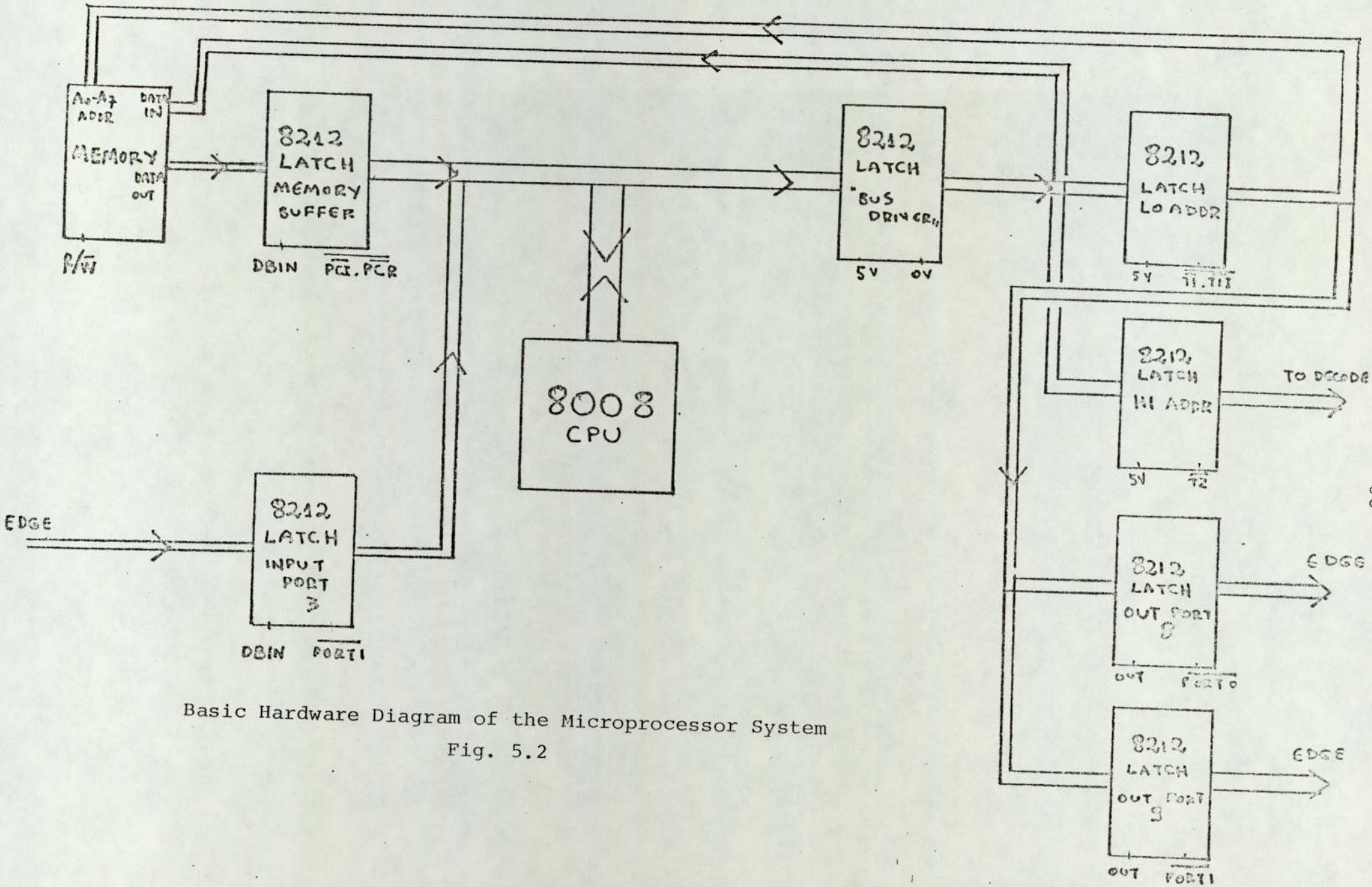
(g) Read/Write control.

Figures 5.2, 5.3 and 5.4 show the functional relationships between these blocks. Complete control over the rest of the logic on the module, according to the instruction it received from memory, are exercised by the 8008 CPU chip. The timing generator consists of a crystal controlled clock oscillator, a state decoder, logic on the CPU chip itself and auxilliary timing logic.

The complete circuit diagram of the CPU module is shown in Fig. 5.5.

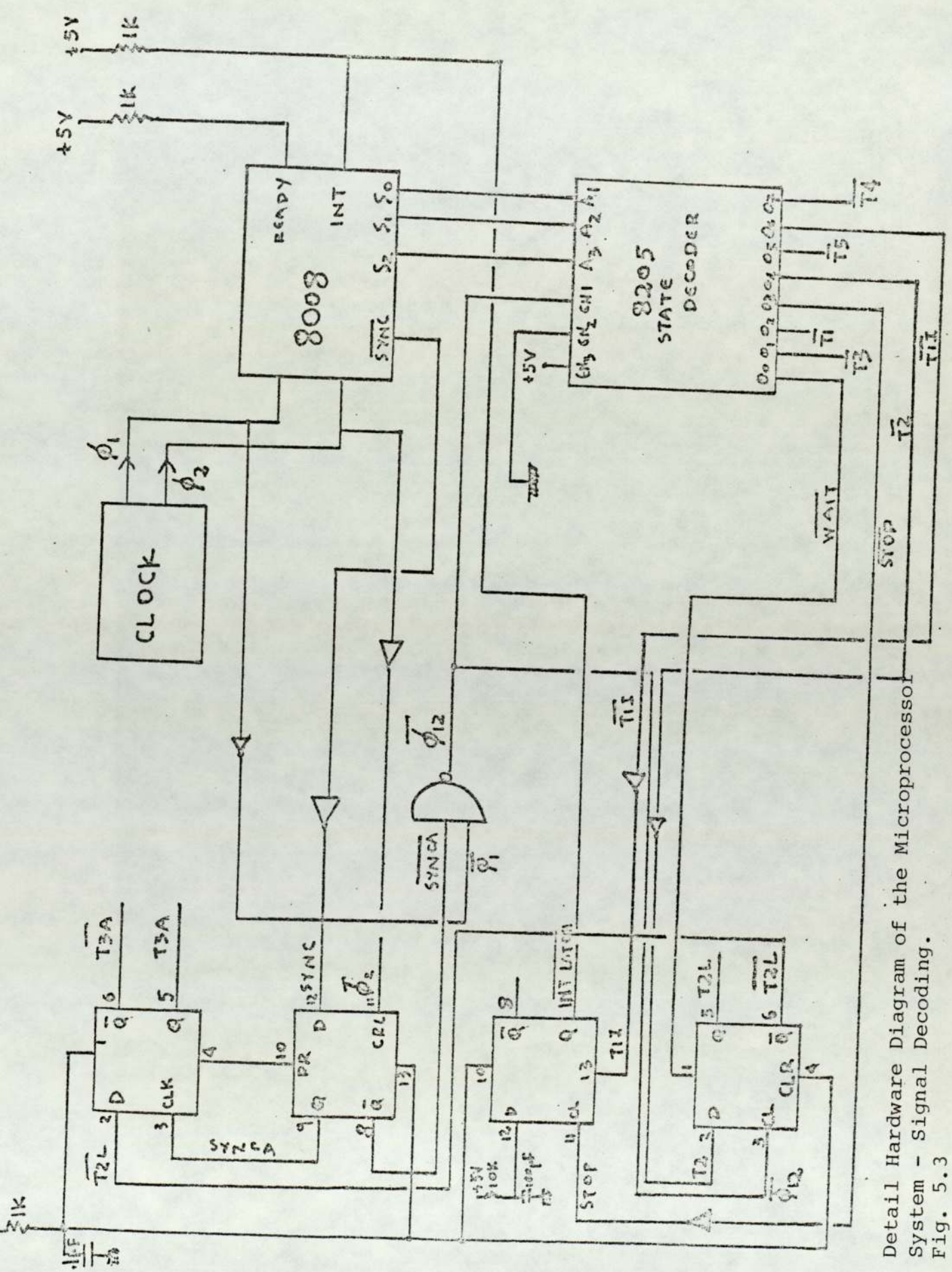
The non-overlapping 500 KHz clock phases ϕ_1 and ϕ_2 which drive the CPU chip as well as other timing circuitry on the board are generated from the clock generator.

The 3MHz quartz crystal is the basis frequency reference. A portion of the crystal's signal output is developed across a capacitor and applied to the clock

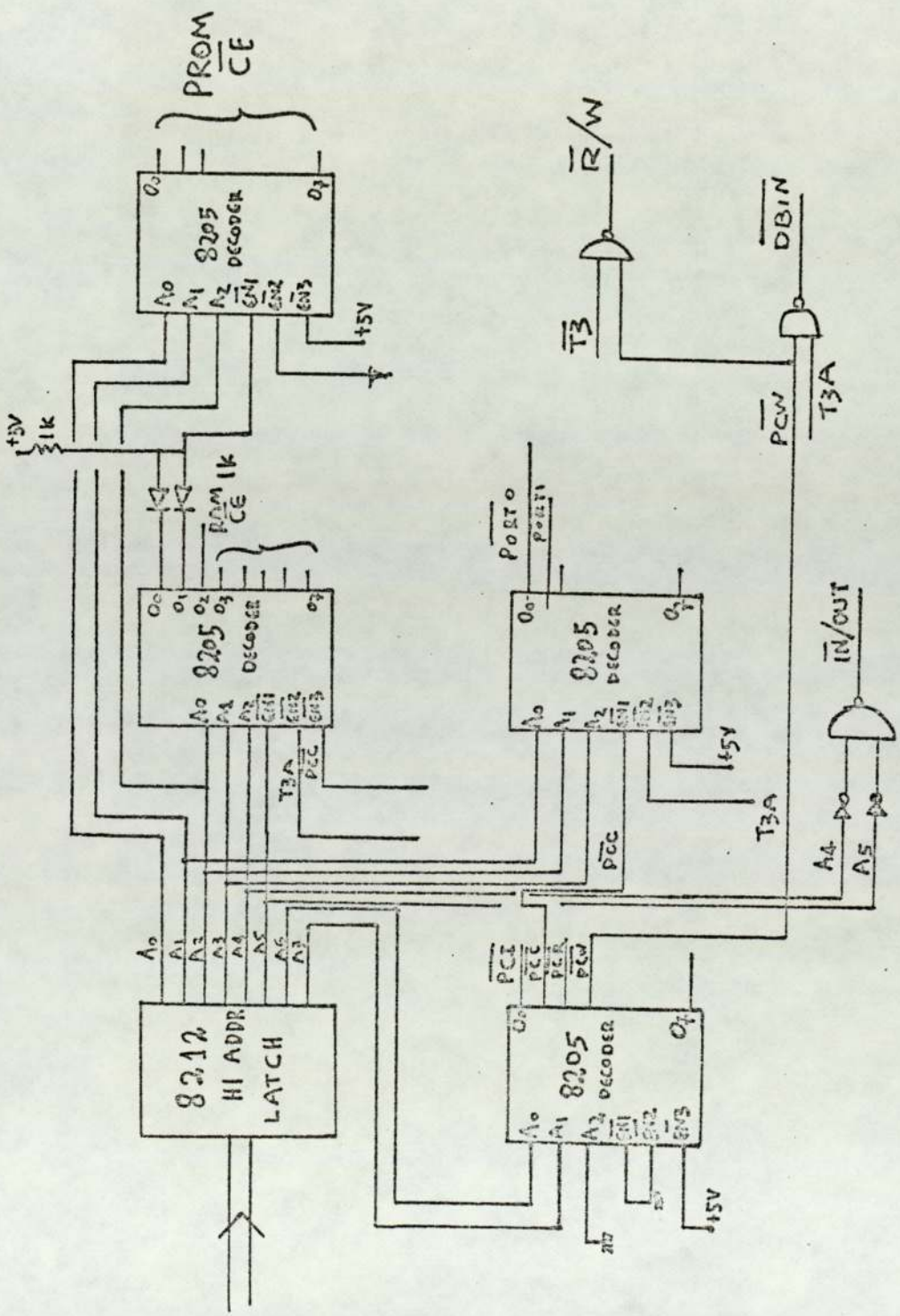


Basic Hardware Diagram of the Microprocessor System

Fig. 5.2



Detail Hardware Diagram of the Microprocessor System - Signal Decoding.
Fig. 5.3



Detail Hardware Diagram of the Microprocessor System - Memory and I/O Decoding

Fig. 5.4

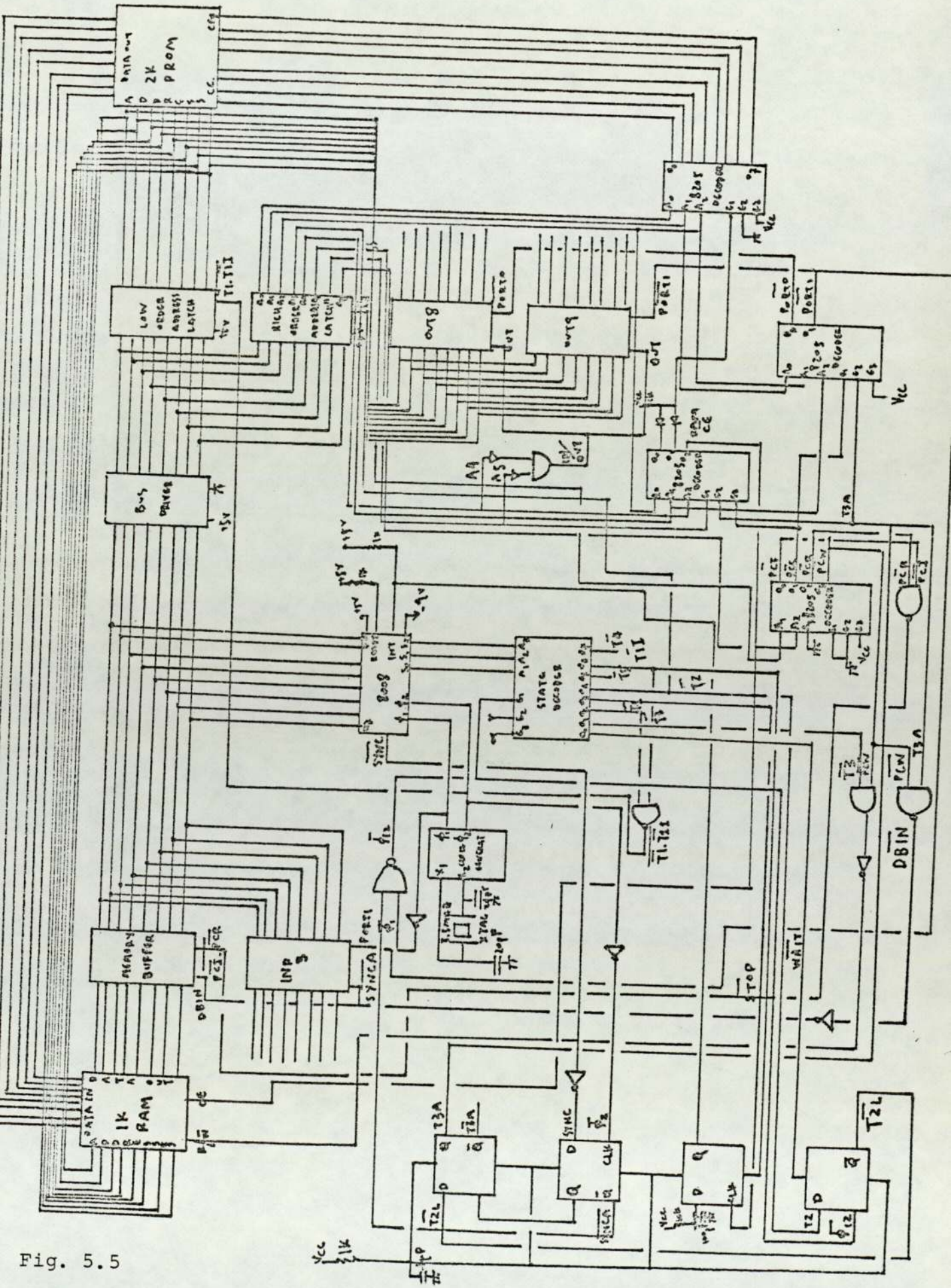


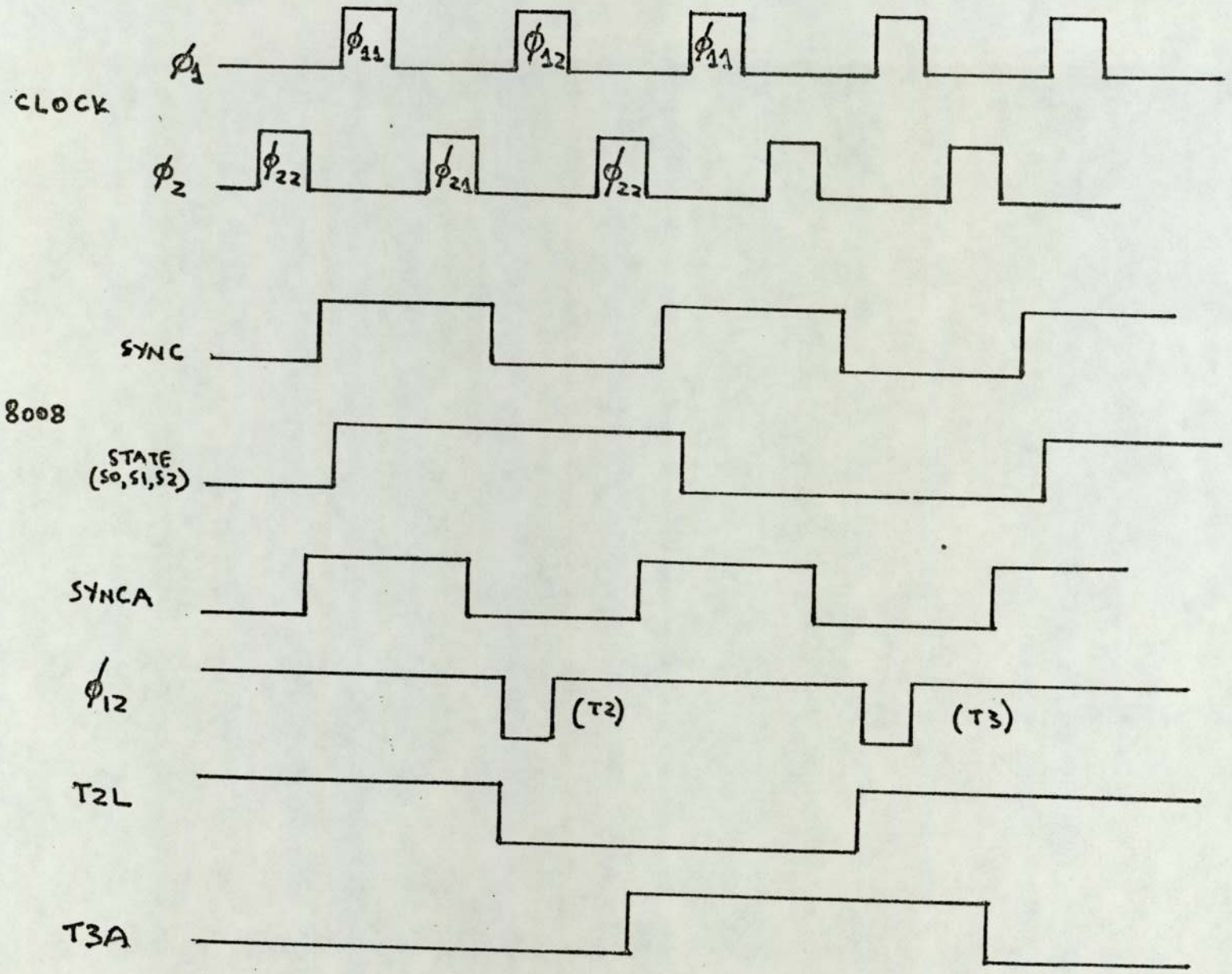
Fig. 5.5

Circuit Diagram of the 8008 CPU Module

generator to which the two signals at 500 KHz frequency are produced. The ϕ_1 and ϕ_2 clock phases are applied to the clock inputs of the CPU chip. The SYNC and clock signals are then fed to the auxilliary timing logic.

In the auxilliary logic system the $\overline{SYNC}$ is applied to a D-type flip-flop (74L74N) which is clocked by the low-to-high transition of ϕ_2. This produces the SYNCA signal which is shown with relation to clock signals ϕ_1, ϕ_2 in Fig. 5.6. SYNCA is used to derive other timing singals on the modules (Fig. 5.3). $\overline{SYNCA}$ and $\overline{\phi_1}$ produce a half frequency clock ϕ_{12} an intermediate signal, applied to a 8205 decoder. Outputs from the state decoder include $\overline{T1}$, $\overline{T1I}$, $\overline{T2}$, $\overline{T3}$, $\overline{T4}$, $\overline{T5}$, $\overline{WAIT}$ and $\overline{STOP}$ The T2 and $\overline{\phi_{12}}$ signals connected to a D-type produce $\overline{T2L}$ signal which is used to generate T3A and $\overline{T3A}$ signals. The T3A signal gated is used during I/O operations.

The first part of every machine cycle is an instruction ferch cycle (PCI). Memory address requires 14 bits, two passes are needed to output memory address. The lower eight bits of the referenced location are transmitted during T1. This byte is sent out on the eight lines of the main data bus and presented to the low order address latch. During T2 the CPU sends out the six high order bits of the referenced address, plus the two cycle bits, in similar fashion. The high order address latch (Intel 8212) is strobed by $\overline{T2}$ output of the state



Timing Diagram of the Microprocessor System
Fig. 5.6

decoder. The fourteen low order bits held in the address latches indicate the location of the instruction that the processor intends to fetch. The two remaining bits indicate that a PCI sub-cycle is in progress.

The $\overline{PCC}$, $\overline{PCW}$, $\overline{PCI}$ outputs are furnished to circuitry on the CPU module itself permitting the modules control logic to generate R/$\overline{W}$ and $\overline{DBIN}$ control signals. The 8212 latches are also used to multiplex the data from memory used as a memory buffer. During T3 the CPU reads this bus and the information on this bus is transferred to the CPU's instruction register.

A $\overline{PCR}$ or a $\overline{PCW}$ signal will be broadcasted by the CPU chip during T2. If a PCR sub-cycle is indicated, external conditions are exactly the same as for an instruction fetch from memory. If a PCW is indicated the cycle decoder activates the $\overline{PCW}$ line which, with $\overline{T3}$ generates the R/$\overline{W}$ command line. The write signal indicates to the memory that data are to be stored in an address location.

If the instruction that the processor fetches from memory is an I/O instruction. That instruction contains a five bit field which specifies one of the 32 peripherals. In order to distinguish an input from an output instruction, the lower eight addresses are reserved for input devices and the upper 24 for outputs. The address

of the object I/O device is sent in the SELECT inputs of a 8205 decoder.and the two signals produced in the outputs of the decoder. $\overline{\text{PORT0}}$ and $\overline{\text{PORT1}}$ are used to strobe the Input/Output and control port latches.

A 'bus driver' latch 8212 was used as a 'follower' between the CPU data and bus and the i/p's of L0 and H1 address latches. For reasons explained in another section INTERRUPTS were not used. One input and two output ports were used.

One part of the memory included on the CPU module is made up of eight Intel C8702A erasable programmable lead-only-memory chips, each having a capacity of 256 eight-bit words. The outputs of the PROMS are connected to the DATA OUT lines of the RAM memory module to the DATA bus of the memory latch on through that to the 8008 data bus. An 8205 decoder is used to enable the selected block of the PROM addressed by the CPU.

In order to obtain data from a memory location, it is necessary to perform a Memory Read Operation. This operation includes two phases. The Address phase in which the desired memory address is sent to the PROM section, where it is decoded and used to enable the specific memory location which is accessed. The Data phase, where the data is sent out from the PROM. A memory (PROM-RAM) read operation is initiated by the CPU chip which sends 14 bit

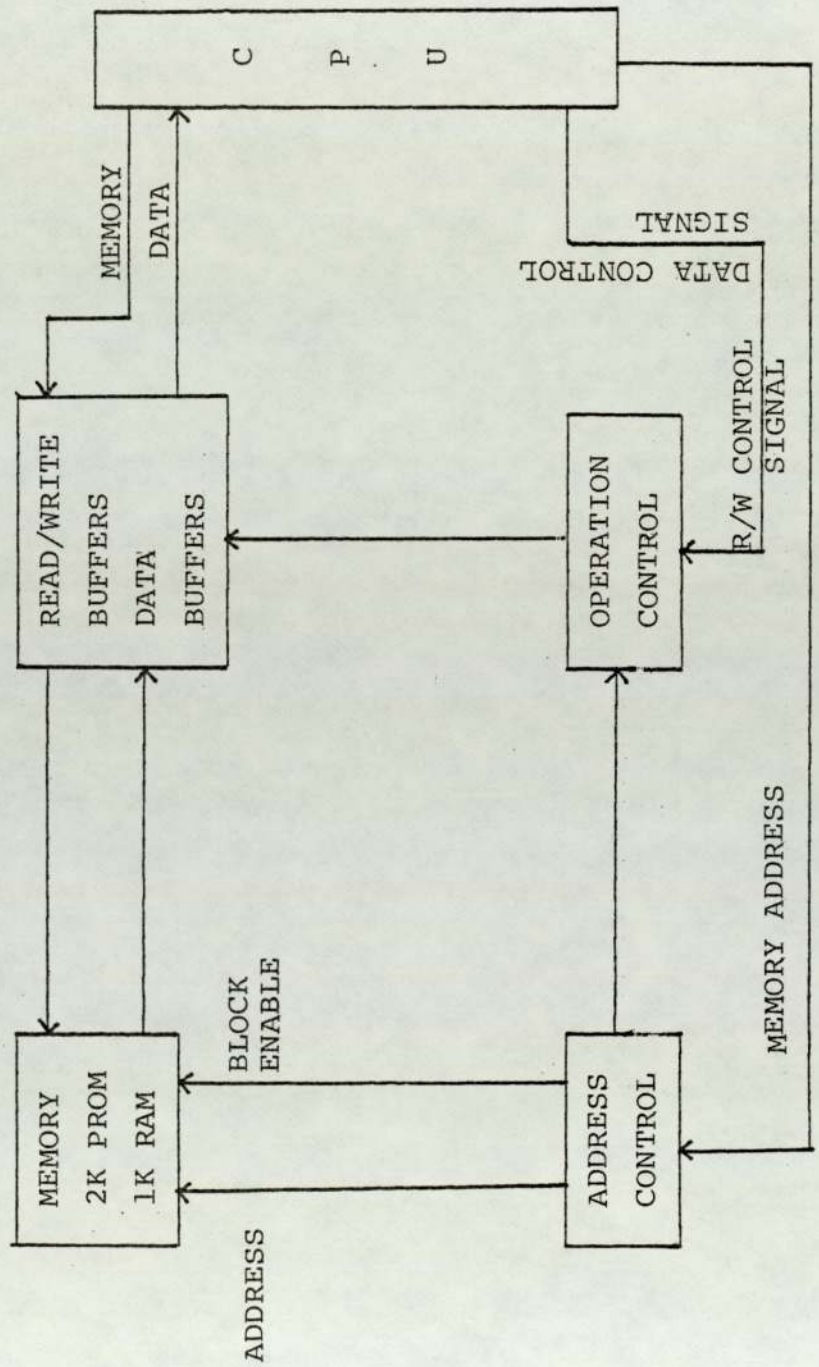
memory address to the address decoding circuits to select one particular memory location. The contents of the specific memory location are then available to the memory data latch whence sent on the CPU chip.

A memory module functional diagram is shown in Fig. 5.7.

The other part of the memory consists of eight INTEL P2102 chips having a capacity of 1024 one bit words. The DATA IN lines on the RAM memory are connected to the inputs of the LO and H1 order address latches.

RAM's DATA OUT lines connected to the PROM's outputs lines and to the bus driver data bus to be processed to the CPU data bus. The RAM memory is selected by one output of an 8205 decoder. The addressing for the PROMs module is designed to have an order address which starts from the location 00:000 up to 07:256 (decimal - 2K) or 00:000 to 07:377 (octal - 2K). The RAM module starts from location 08:000 up to 11:256 (decimal - 1K) or 10:000 up to 13:377 (octal - 1K).

The processor also sends signal $R/\overline{W}$ to the RAm memory. In its false states this signal dictates a write operation, therefore, during a read operation it will be true. If the signal is a write operation signal, data available on the input lines will be written into the selected memory location by the write pulse. If the signal is



MEMORY MODULE FUNCTIONAL DIAGRAM

FIG. 5.7

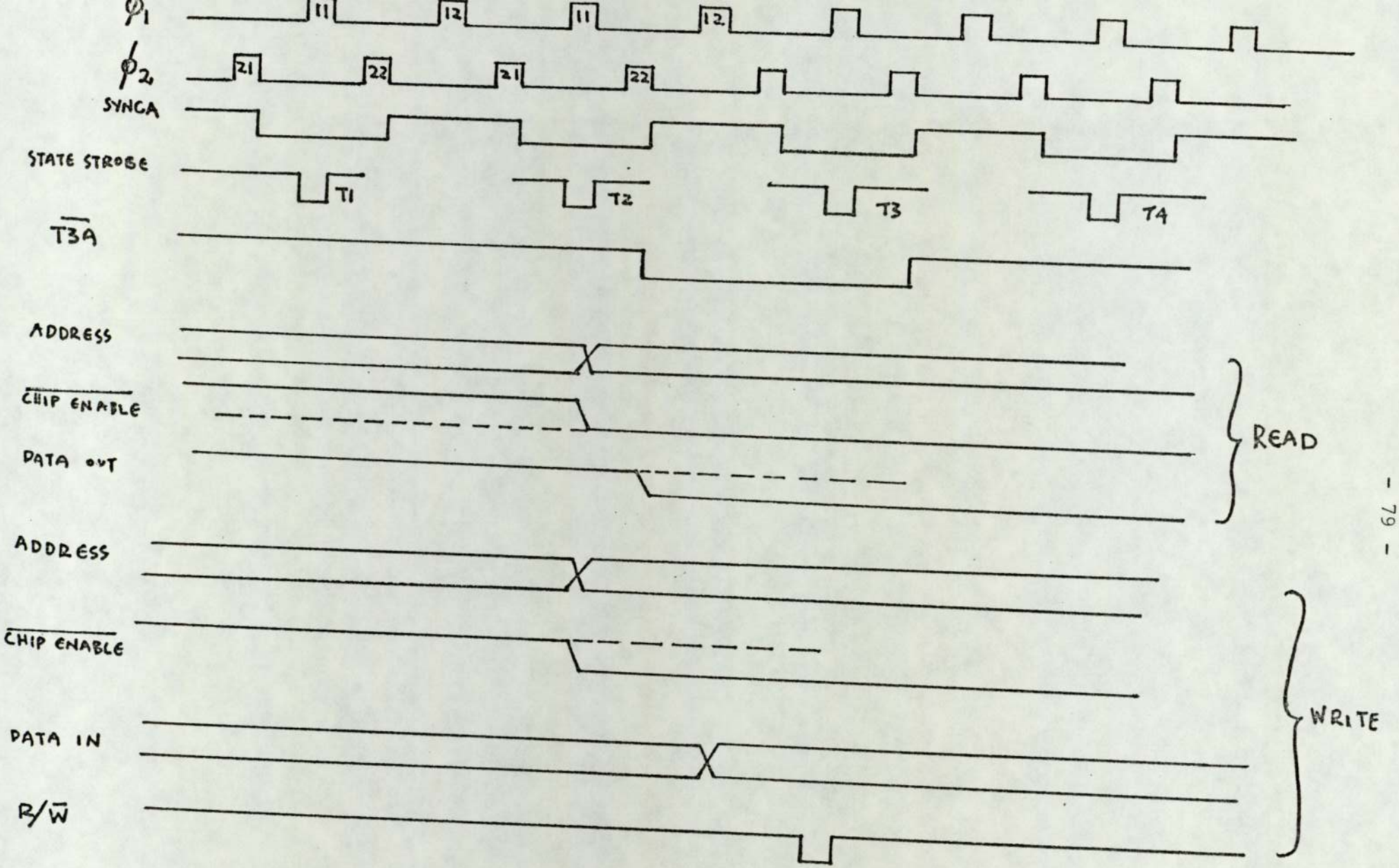
read operation signal, data of the selected location will be appeared on the data output lines.

The timing diagram for the Read/Write operations is shown in Fig. 5.8.

The operation and decoding for an I/O instruction has been explained earlier in this section. An input operation after the selection of the input port has been obtained, is performed in order to get data from an external source and to present it back to the CPU. In an output operation data from memory and the CPU are sent for use by an external device.

After the construction of the two CPU boards checking on its operations, I/O memory, was performed with different peripheral devices. Peripheral devices may differ in transmission characteristics. The VDU and teletype asynchronous serial bit stream, consist of data bits that are preceded by a start bit and followed by one stop bit. The start and stop elements do not contain information, but they do establish bit and character synchronisation at the receiving device.

In the transmission of data, a clock signal is not transmitted along with the data, and gaps (idling) between the characters may result. Therefore, the receiving device must generate a clock that is synchronised to the data for the purpose of data sampling.



Timing Diagram of the Read/Write Operations

Fig. 5.8

The interface board between the microprocessors and the VDU used, a MOS/LSI data communication device, the UART (Universal Asynchronous Receiver/Transmitter) which performs serial/parallel data conversions, timing and synchronising circuitry.

Different programs used for checking the operations of the CPU boards are given in APPENDIX C .

The PROM chips were programmed on the INTELLECT 8 development system.

The microprocessor CPUs board, the separate memory board, the VDU interface and power supplies were housed in a rack. Layouts of the two CPU boards and the separate shared memory board are given in APPENDIX E. Microprocessors are shared memory back connections on the rack are given in APPENDIX F .

5.4 DESIGN AND CONSTRUCTION OF THE SHARED MEMORY

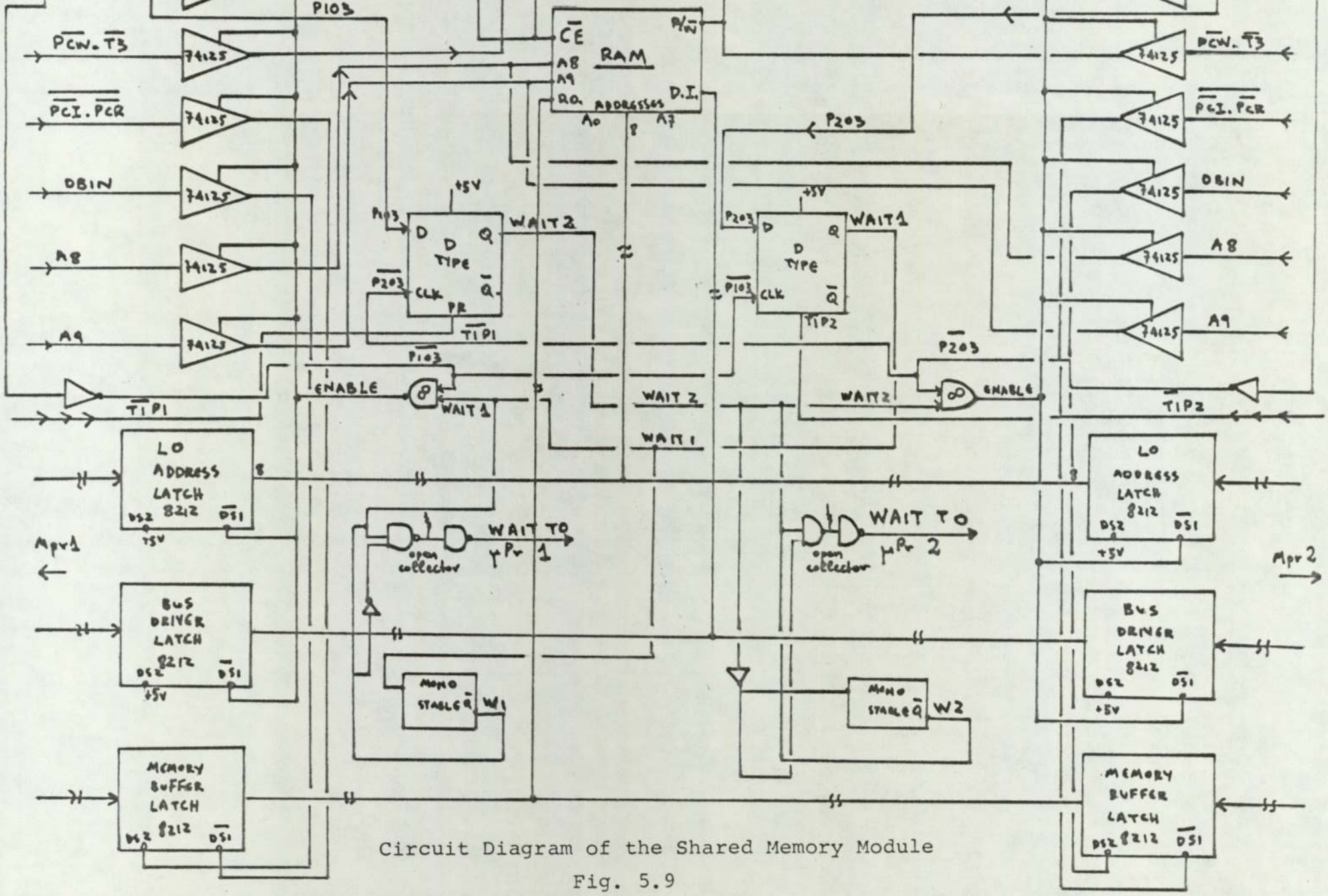
The shared memory board module consists of 1K RAM memory, accessible by both microprocessors and all the necessary control circuitry for both microprocessors to ensure equal accessibility and servicing. The design is a symmetrical one as each processor uses its own control circuitry to access the memory. An important design issue was to ensure accessibility to the memory of one

processor at a time on the first requested first served basis. If one of the processors is requesting access to the shared memory while the second processor is using it, the first goes to a WAIT state (idles) until the second one has been served and then the first one can access it. A WAIT state signal may be of indefinite length but the actual WAIT interval is always an even multiple of the processors clock period. In order to guarantee an exit from the WAIT state the processors READY line must go high at least 350 ns prior to the trailing edge of ϕ_{22}. When this condition is fulfilled the processor proceeds to the T3 state, begining with the next ϕ_1 clock pulse.

The complete circuit diagram of the shared memory module is shown in Fig. 5.9.

In the previous section we mentioned that output of the 8205 H1 address decoder can decode 1K of RAM. Thus the O_3 output of this decoder from both processor is connected to the $\overline{CE}$ of the shared memory. The $R/\overline{W}$ signal of the memory is connected to the $\overline{PCW}$. $\overline{T3}$ signal for both processors.

The RAM addresses A_0 ... A_7 are connected with the output of the low address latches (8212) of each processor. through a low address latch (8212) one for each processor and when enabled it accesses memory. The DATA IN signals



Circuit Diagram of the Shared Memory Module

Fig. 5.9

of the RAM, memory are connected to each processor's BUS DRIVER through a BUS DRIVER LATCH (8212). The RAM DATA OUT are connected through memory buffer latches to the outputs of the corresponding memory buffers for each processor. The A_8, A_9 addresses of the RAM are connected to the A_8, A_9 high latch addresses of the two processors. The enable signal for the low address latches in that separate board is the same with the enable of the BUS DRIVER latch. In fact signals from each processor PO_3, $\overline{PCW}$. $\overline{T3}$, $\overline{PCI}.\overline{PCR}$, $\overline{DBIN}$ the last two enabling the memory buffer latch, and A_8, A_9 address lines, pass through a three-state buffer (74125). The output of this gate is disabled when the enable of the buffer is high. The enables of all the above mentioned signals are connected to the output of a NAND gate with inputs the WAIT of the individual processor, trying to access the shared memory and the $\overline{PO_3}$ signal. If the WAIT signal is high the processor is not waiting, and when $\overline{PO_3}$ goes high, that particular low processor requests access to the memory. Thus when both go Hl the output of the NAND goes LOW and all the signals are then enabled allowing the processor to access memory. If at the time that the first processor is requesting access to the shared memory, the other processor is accessing it, then the first processors WAIT line goes LOW thus disabling access.

A D-type flip-flop was used to produce the WAIT

signal for the processor that is trying to access memory, while the other one is already accessing.

The D input of the flip-flop is connected to the PO_3 signals of the processors. The clock input is connected to the $\overline{PO_3}$ signal of the other processor (D input $P1O_3$, CLK input $\overline{P2O_3}$).

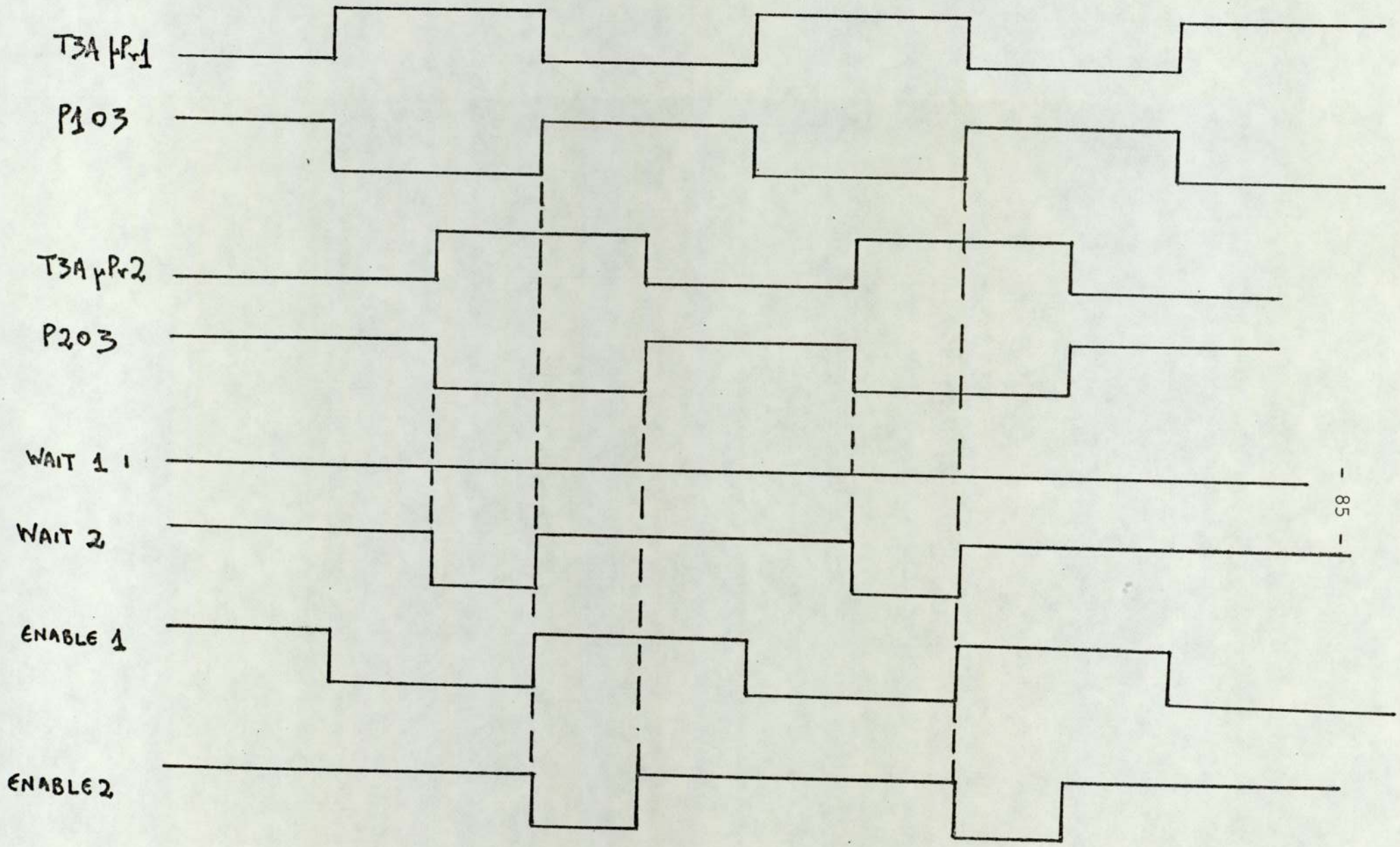
The output Q of the flip-flop will follow the data input D while the clock is high. Latching will occur when the clock returns to low. Thus the Q output is connected to the WAIT line of the other processor. The PR line for each flip-flop is connected to the $\overline{T1}$ signal of each processor. This logic provides the idling of one of the two processors when they both try to access memory at the same time.

The timing diagram is given in Fig. 5.10.

The layout of the shared memory board is given in APPENDIX E .

Programs on the operation of the two microprocessors with the shared memory are written in INTEL 8008 microprocessor language. Debugging the programs was made easier by presenting and programming the language instructions set in HEXADECIMAL machine codes.

Two sets of different programs for both microprocessors examine the capabilities of communication between the two



Timing Diagram of the Multi-processor System

Fig. 5.10

microprocessors through the shared memory.

These programs produce a slow and a fast count with very fast access from both microprocessors to a common location in the shared memory. The purpose is to establish the performance of the two microprocessors, when executing individual tasks, both attempt to access memory simultaneously, APPENDIX C.

As new components such as memory devices and microprocessor support chips are continuously appearing on the market, no detailed circuit specifications of such items will be given in this report. For information on such devices one should consult the latest microprocessor support components literature, and new devices specifications.

CHAPTER 6

OPERATION OF THE INTEL 8008 MULTI-PROCESSOR SYSTEM WITH EXTERNAL COMPUTING DEVICES

CHAPTER 6

OPERATION OF THE INTEL 8008 MULTI-PROCESSOR SYSTEM WITH EXTERNAL COMPUTING DEVICES

6.1 INTRODUCTION

Typical multi-processor architectures implement a parallel or symmetric multi-processor architecture in which every processor is equally capable of picking up any task. The advantage of this approach is that any number of processors from 1 to a maximum physical limit can be used, thus yielding a nodular machine whose power can be tailored to nodal requirements. We chose instead a multi-processor architecture which dedicates each processor to a certain subset of the tasks that have to be performed. By logical distribution of tasks we mean the relationships between the various tasks that the processor system is expected to perofrm. In our case where the hardware is fixed, the distribution of these tasks affects only the structure of the software. Specifically it affects the execution sequence of the various processes. In this system, in effect, we replace the multi-processing software of earlier computer systems with hardware. The fact that the tasks now can be done at the same time compensates for the lower performance characteristics of the components.

6.2 TASK ALLOCATION

Several approaches have been suggested for allocating processors to processes. On one hand, the low cost of microprocessors makes it feasible to assign a processor to each task. For example, a multi-processor network for machine tool control has been proposed[29] consisting of three microprocessors. Two of the processors control for axes of the machine tool while the third acts as an executive. In addition to this an alternative approach allocates processors dynamically. This requires that the processors are in close proximity and that they have common mode of access to I/O signals[30].

The nature of these tasks in our approach is that they are short or at least divisible into short segments and they are mostly independent of one another. Our system is mainly dedicated to localised monitoring control. In that capacity local memory for each processor was essential to keep the programs executed more frequently and since the processor configuration was on a master-master basis the concept of the shared memory was extremely useful on interprocessor communication and data exchange.

6.3 COMMUNICATION FUNCTIONS

The existence of separate I/O facilities by each processor offers flexibility to our approach as it maintains

the independence of each processor to communicate with external devices and at the same time offers a rigid modularity to the whole system. Each processor controls a portion of the overall process, with the necessary co-ordination between the control strategies effected through the processor intercommunication means. Both processors are of equal importance in maintaining control of the process and both must be operating to obtain optimum performance.

The reliability of the network depends on the reliability of the nodes and the reliability of the communications system.

Thus the use of local memories to each processor as well as shared memory, although not unique in conception, we felt that it is superior in our system from the concept of global memory as many memory accesses could be serviced by the local memories and thus accesses to shared memory could be reduced and bus bandwidth requirements could be eased.

All interprocessor communications via the shared memory are done over the system bus. The bus operates synchronously with processors, memories and I/O. There is no priority system on accessing the shared memory and when one processor is trying to access and the other one is accessing,the first processor goes to a WAIT state

until the job of the second one terminates. In case of a tie, there is an arbitrary priority order. We found that this non-sophisticated approach for our applications guarantees independence and flexibility of the processor and there is no bus contention problems as well as speed.

Our system could easily be used as an ON-LINE control or as a REAL-TIME system, Fig. 6.1.

6.4 INTEL 8008 MULTI-PROCESSOR SYSTEM AND 9900/4 MICROPROCESSOR INTERFACE: DESIGN AND CONSTRUCTION

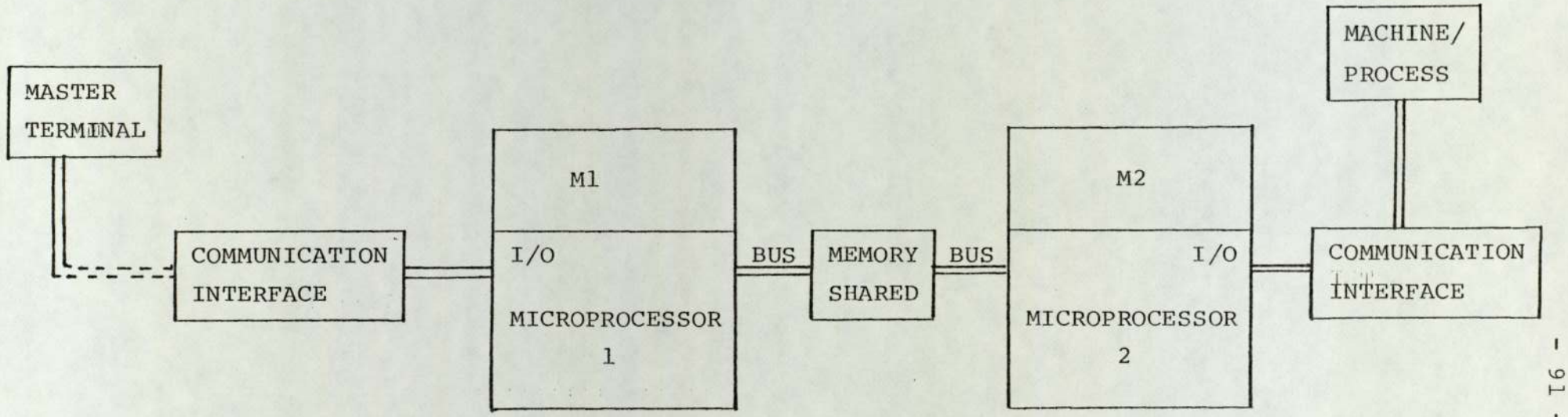
The system designed and constructed, as described in the previous chapter, was tested and operated on its own using the VDU terminal to verify the results of the simulation and operation of different programs applied to the system.

The sysetm was then connected to the new T.I. 9900 microprocessor available in the department to perform a particular operational task.

An interface between the 8008 system and the T.I. microprocessor system was designed and built.

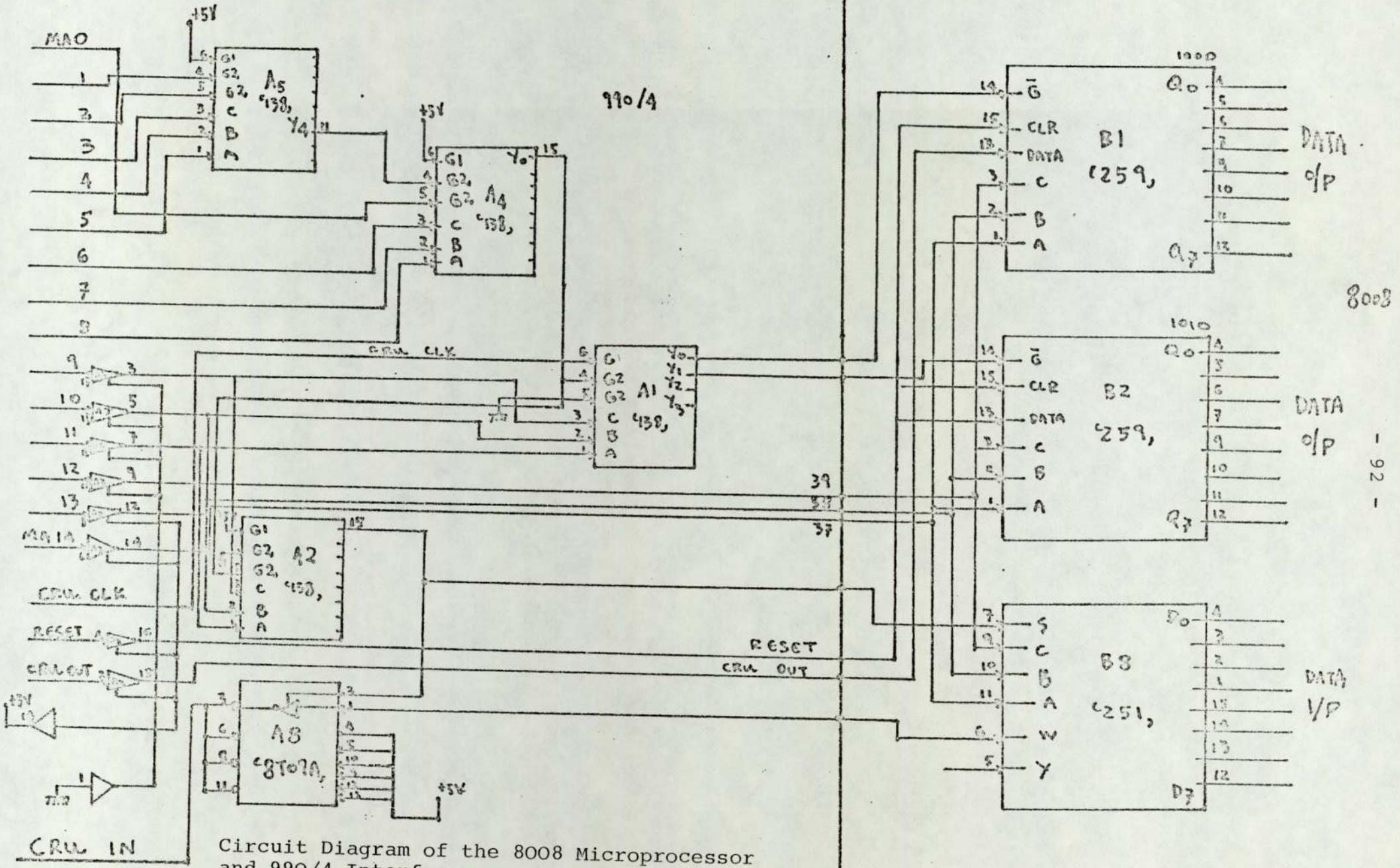
The circuit diagram for that interface is shown in Fig. 6.2.

The design of that interface was concentrated on the communication of the 8-bit 8008 with the 16-bit



A Multi-Microprocessor Control System

Fig. 6.1



Circuit Diagram of the 8008 Microprocessor and 990/4 Interface

Fig. 6.2

T. I. Microprocessor.

Four LSI38 memory decoders were used to decode the 16 memory address lines of the 9900 and to ensure data transfers via the CRU.

Octal buffers and line drivers SL-241 were connected to memory address lines M9 to M14 of the 9900 and to the RESET and CRU OUT signals. A tri-state quad bus driver "8 TO 9" was used to buffer the CRU IN signal. The Y_0, Y_1 outputs of one decoder were used to enable two 8-bit addressable latches (LS259). The DATA IN inputs of these latches were connected to the CRU OUT signal from the 9900. Inputs of these latches were connected to the Memory Address of the 9900. The 8-bit outputs of these two latches were connected to the two output ports of the 8008 system (port 8 - port 9). Specific address locations for these two latches with respect to the 9900 were designed and handwired at 1000, and 1010. The DATA SELECT inputs of the DATA selector SL251 chip were connected to the 9900 memory address lines. The strobe of the same chip was connected to the decoded CRU or signal. The 8-bit output of this chip were connected to the INPUT port of the 8008 system.

6.5 MULTI-TASK SYSTEM: OPERATIONAL SIMULATION

Having established and tested the communication between the 8008 system and the 9900 a problem task was

implemented. The task has a military application. The idea of this particular problem was presented to the author when he was doing his military service in the Navy.

The problem is to compute quickly and accurately the 'RANGE' and 'ELEVATION' settings of a gun, given the distance of a particular target.

Different values of distances between targets are fed by an operator via the key board of a VDU to the T.I. 9900 microprocessor system. The 9900 transfers the data to the 8008 system which analyses it, with intercommunications between the two 8008 microprocessors and their shared memory and transfer back to the 9900 the values for the 'RANGE' and 'ELEVATION' and from there to the screen of the VDU.

The I/O facilities of the 8008 microprocessor 1, were connected to the 9900 system and the task of this particular microprocessor was to compute the 'RANGE' given the distance value, transferred from the CRU OUT of the 9900. When the 'RANGE' is computed it is stored in a location in the shared memory. The 8008 microprocessor 2 is accessing this particular memory location fetches the 'RANGE' value and from that computes the corresponding value for 'ELEVATION'. Elevation values is the task allocated to microprocessor 2. When the appropriate 'ELEVATION' value is calculated it is being stored in

another location in the shared memory. Microprocessor 1 fetches the 'ELEVATION' value and outputs both the 'RANGE' and 'ELEVATION' values to the 9900 and from there to the VDU screen.

The execution programs of the 8008 microprocessors were in PROMs and the different values of 'RANGE' and 'ELEVATION' were stored in the private RAM memories of these microprocessors.

The programs are written in INTEL 8008 microprocessor language and the other one in T.I. 9900 microprocessor language. The 8008 programs were programmed in PROM on the INTELLEC 8 system available in the department. The mnemonics code was presented in HEXADECIMAL instruction machine code. Board connections between the two systems are given in APPENDIX G .

In this particular application the I/O facilities of one of the 8008 microprocessors were used, but I/O facilities of both 8008 microprocessors could be connected to one terminal and a machine to complete the purpose of the localised control system we built. Other peripheral devices could also be connected as well as facility for scanning memory banks from a disc or floppy disc. The system also could be expanded by connecting to it another set(s) of microprocessor system identical to the one we built for different control purposes and applications.

In this example, the message paths are direct. In a more complex application the message paths need not be sequential. As more processors or functions are added to the system, the number of routing paths increases,but does not become more complex for any given processor, in the system. Therefore, each processing module may be viewed as a distinct building block, and modular development of the system is practical.

Basic flow diagrams of the multi-processor communication are given in Figs. 6.3, 6.4 and 6.5.

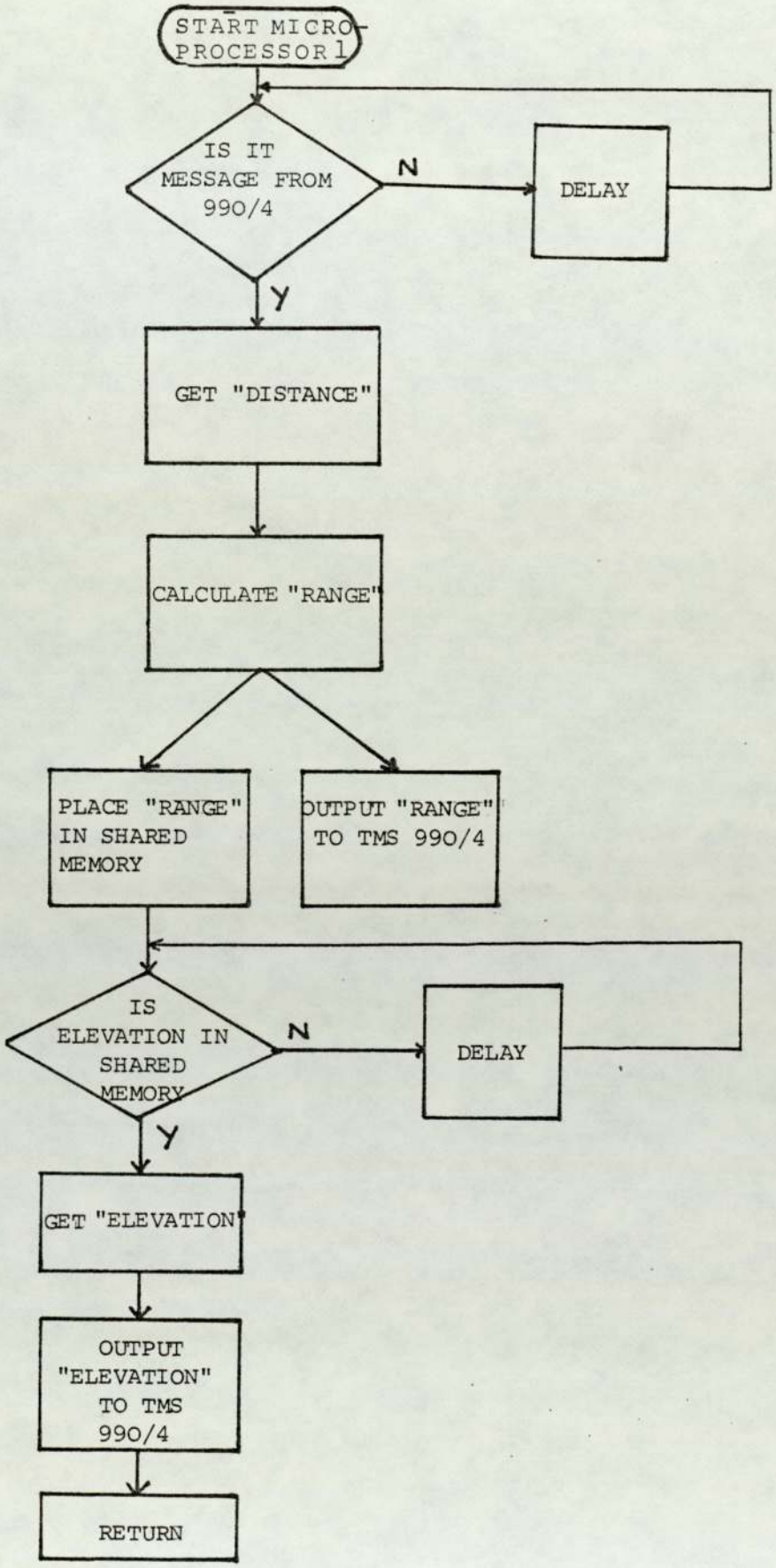


Fig. 6.3 BASIC FLOW DIAGRAM OF MICROPROCESSOR 1 TASK

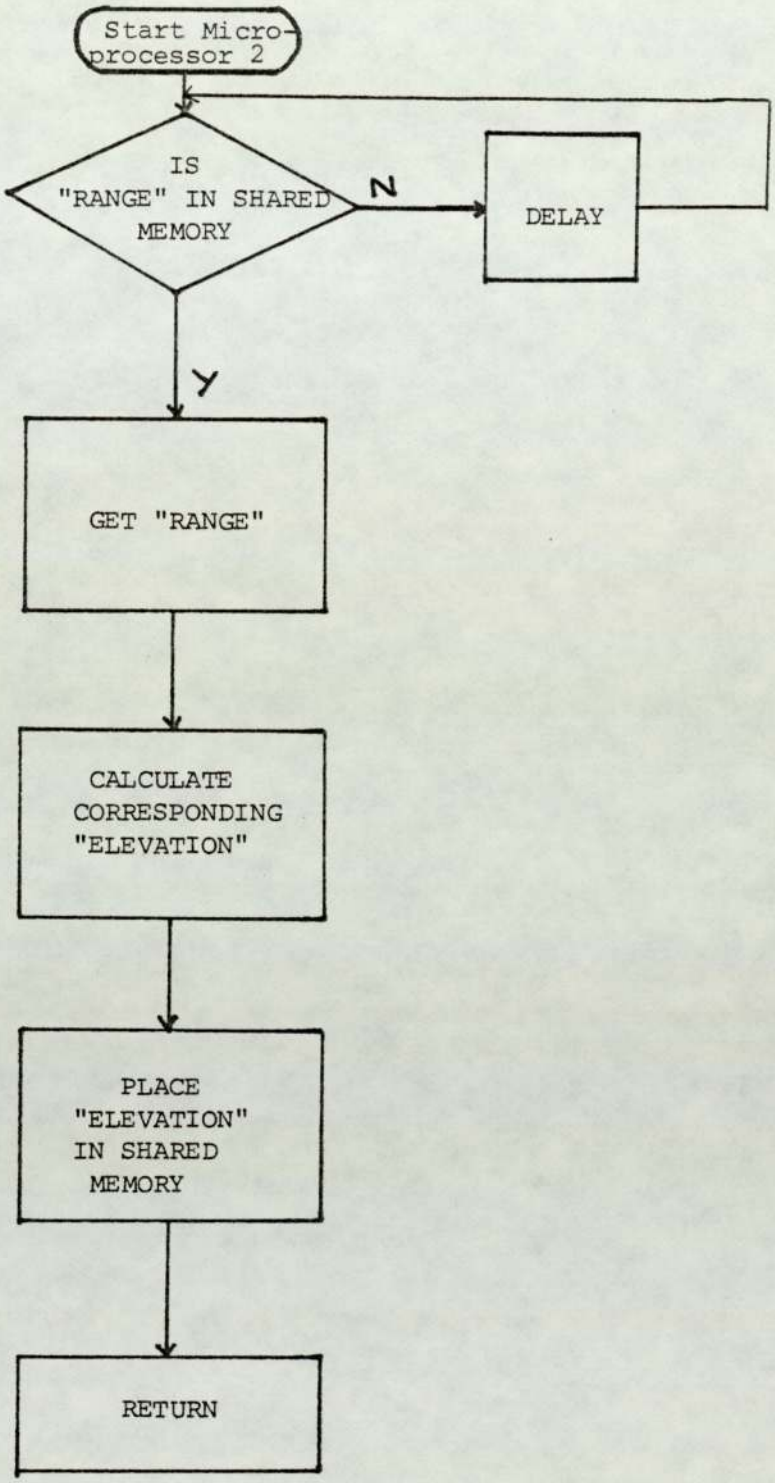


Fig. 6.4 BASIC FLOW DIAGRAM OF MICROPROCESSOR 2 TASK

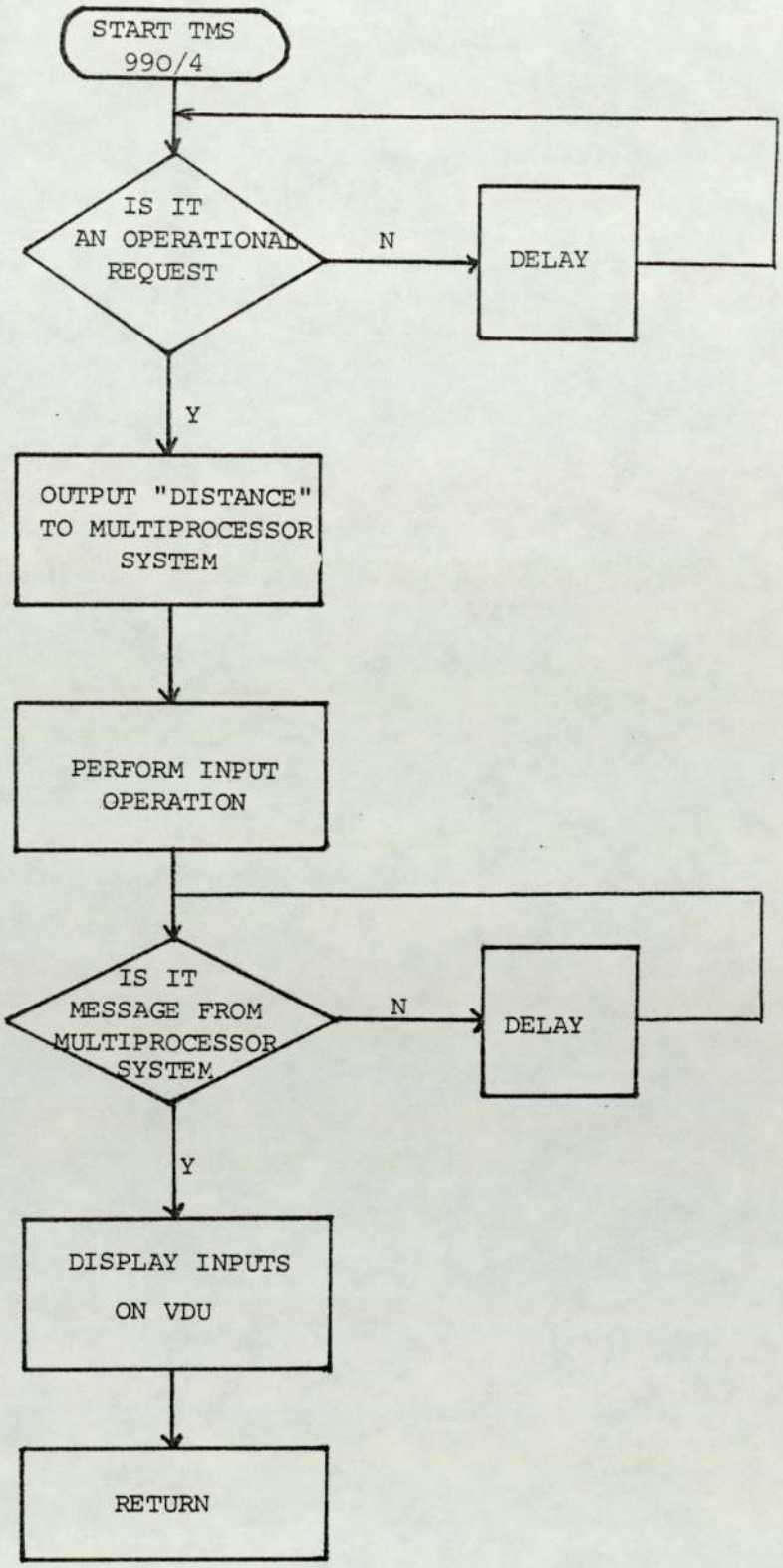


Fig. 6.5 BASIC FLOW DIAGRAM OF TMS 990/4 TASK

OPERATIONAL SIMULATION PROGRAMS

INTEL 8008 MICROPROCESSOR 1

Task Allocation - RANGE

```
        XRA
        LBA
        LEI     1
        LCA
        LDA
        LAI     50D
        LHI     10B
        LLI     100B
        LMA
        LAI     100D
        LHI     10B
        LLI     110B
        LMA
        LAI     150D        RANGES values
        LHI     10B         are stored in
        LLI     120B        RAM
        LMA
        LAI     200D
        LHI     10B
        LLI     130B
        LMA
        LAI     250D
        LHI     10B
        LLI     140B
        LMA
        LAI     0
KEY     INP     3B          Value of the
        CPI     0           DISTANCE in
        JTZ     KEY         microprocessor 1
        LHI     10B
        LLI     100B
```

```
        LBM
        CPB
        JTC     +2
        JFC     LET
        LHI     10B
        LLI     110B
        LBM
        CPB
        JTC     +2
        JFC     LET         Establishment
        LHI     10B         of the corresponding
        LLI     120B        RANGE values
        LBM
        CPB
        JTC     +2
        JFC     LET
        LHI     10B
        LLI     130B
        LBM
        CPB
        JTC     +2
        JFC     LET
        LHI     10B
        LLI     140B
        LBM
        CPB
        JFC     LET
        JTC     LET

LET     LEA
        LAB                 Output the RANGE
        OUT     10B         value to 9900/4
        LCB                 system
        LBA
        LAB
        LAA!LAA
```

```
        LAI     0
        OUT     10B
        LHI     14B
        LLI     100B
        LMC                 Value of the
        CALL    DELAY       RANGE in shared
STAR    LHI     14B         memory
        LLI     200B
        LAM                 Value of ELEVATION
        CPI     0           in ACC from the
        CTZ     DELAY       shared memory
        CPI     0
        JTZ     STAR
        OUT     10B         Output the ELEVATION
        LAA!LAA             value to 9900/4
        LAA!LAA             system
        LAI     0
        OUT     10B
        LHI     14B
        LLI     200B
        LMA
        JMP     KEY
DELAY   LDI     300B
        DCD
        JFZ     DELAY
        RET
```

INTEL 8008 MICROPROCESSOR 2

Task Allocation - ELEVATION

```
        XRA
        LBA
        LCA
        LDA
        LEI     1
        LAI     70D
LET1    LHI     10B
        LLI     100B
        LMA
        LAI     50D
LET2    LHI     10B
        LLI     110B        ELEVATION values
        LMA                 are stored in
        LAI     40D         RAM
LET3    LHI     10B
        LLI     120B
        LMA
        LAI     20D
LET4    LHI     10B
        LLI     130B
        LMA
        LAI     20D
LET5    LHI     10B
        LLI     140B
        LMA
        LAI     0
KEY     LHI     14B
        LLI     100B        Value of the RANGE
        LAM                 in Acc. of Micro-
        CPI     0           processor 2 from
        CTZ     DELAY       shared memory
        CPI     0           Acc.=Accumulator
```

```
                JTZ     KEY
                LEA
                LBA
                LAI     0
                LHI     14B
                LLI     100B
                LMA
                LAB
                LAE
                CPI     50
                JTZ     ELEV1
                CPI     100
                JTZ     ELEV2       Comparison with
                CPI     150         RANGE values
                JTZ     ELEV3
                CPI     200
                JTZ     ELEV4
                CPI     250
                JTZ     ELEV5
ELEV1           LHI     10B
                LLI     100B
                LAM
                CAL     SHARE
                JMP     KEY
ELEV2           LHI     10B
                LLI     110B
                LAM
                CAL     SHARE
                JMP     KEY
ELEV3           LHI     10B
                LLI     120B        Establishment of
                LAM                 the corresponding
                CAL     SHARE       ELEVATION value
                JMP     KEY
ELEV4           LHI     10B
```

```
        LLI     140B
        LAM
        CAL     SHARE
        JMP     KEY
SHARE   LHI     14B
        LLI     200B        Value of ELEVATION
        LMA                 in shared memory
        LAA!LAA
        RET
DELAY   LDI     300B
        DCD
        JFZ     DELAY
        RET
```

```
        TITL 'TASK ALLOCATION "DISTANCE" '
        IDT  'TAD'
START   LWPI WS
        BL   @INIVDU        INITIALISE UART
NEXT0   EQU  $
        BL   @OUTXT
        DATA >0D0A
        TEXT 'DISTANCE='
        BYTE ' '+>80
        BL   @READNO        INPUT NUMBER
        LI   R12,>1000      ; OUTPUT
        SWPB R8             ;  DISTANCE
        LDCR R8,8           ;  TO 8008
        CLR  R0                 ;WAIT FOR READY
TEST1   STCR R0,8               ; SIGNAL FROM
        JEQ  TEST1              ;  8008
        NOP                 DELAY
        NOP
        STCR R2,8          RANGE FROM 8008
        CLR  R0
        LDCR R0,8           O/P READY TO 8008
TEST2   STCR R0,8          ;WAIT FOR
        JNE  TEST2         ;ACKNOWLEGE
TEST3   STCR R0,8               ;WAIT READY
        JEQ  TEST3              ; SIGNAL
        NOP                     ;
        NOP                     ;
        STCR R3,8          ELEVATION FROM 8008
        BL   @OUTXT
        DATA >0D0A
        TEXT 'RANGE='
        BYTE ' '+>80
        MOV  R2,R8
        BL   @WRITNO        DISPLAY RANGE
        BL   @OUTXT
        DATA >0D0A
        TEXT 'ELEVATION='
        BYTE ' '+>80
        MOV  R3,R8
        BL   @WRITNO        DISPLAY ELEVATION
        JMP  NEXT0          ENDLESS LOOP
*
**************************
*
* ROUTINE TO INITIALISE UART
*
INIVDU LI    R12,>1B40      CRU BASE
       SBO   31             RESET
       LI    R0,>6200
       LDCR  R0,8           CTRL
       SBZ   13             NO TIMER
       LI    R0,>34         SPEED=9600
       LDCR  R0,12
       B     *R11
```

```
*
* ROUTINE TO OUTPUT TEXT STRING
*
OUTXT   MOV   R11,R10       SAVE RET.
OUTXT1  MOVB  *R10+,R9      GET CHAR
        BL    @OUTCHR       O/P
        MOVB  R9,R9         LAST?
        JGT   OUTXT1        NO,LOOP
```

```
        NEG   R10           ; YES,
        ANDI  R10,>FFFE     ;  EVEN
        NEG   R10           ;   ADDRESS
        B     *R10          ; & RETURN
*
* ROUTINE TO INPUT ONE CHAR
*
INCHR   LI    R12,>1B40     CRU BASE
        CLR   R9
INCHR1  TB    21            CHAR REC'D ?
        JNE   INCHR1        NO, LOOP
        STCR  R9,7          YES, PUT IN R9
        SBZ   18
        B     *R11
*
* ROUTINE TO OUTPUT ONE CHAR
*
OUTCHR  LI    R12,>1B40     CRU BASE
        SBO   16            RTSON
OUTCH1  TB    22            TX BUFFER EMPTY?
        JNE   OUTCH1        NO, LOOP
        LDCR  R9,8          YES, LOAD
        SBZ   16
        B     *R11
*
* ROUTINE TO INPUT NUMBER
*
READNO  MOV   R11,R5        SAVE RETURN
READ0   LI    R6,10         BASE
        CLR   R7
READ1   BL    @INCHR        INPUT CHAR
        CI    R9,>0D00
        JEQ   READ0         END IF CR
        SRL   R9,8          MOVE TO RHB
        AI    R9,->30
        JLT   ERROR         TOO SMALL
        C     R9,R6
        JHE   ERROR         TOO BIG
        CLR   R10
        MPY   R6,R7         SUM=SUM*10
        MOV   R7,R7
        JNE   ERROR         OVERFLOW
        MOV   R8,R7
        A     R9,R7         SUM=SUM+DIGIT
        JOC   ERROR         OVERFLOW
        JMP   READ1         GET NEXT DIGIT
READ0   MOV   R7,R8         ANSWER IN R8
        B     *R5           RETURN
```

```
*
ERROR   BL    @OUTXT                ; ERROR--
        DATA  >0D0A                 ; OUTPUT
        TEXT  'ERROR,TRY AGAIN'     ;  MESSAGE
        BYTE  >0D,>0A+>80           ;
        JMP   READ0                 ; THAN LOOP
*
* ROUTINE TO OUTPUT NUMBER
*
WRITNO  LI    R5,3                  MAX.3 DIGITS
        LI    R6,10
WRITN1  DEC   R5
        JLT   ANSWER
        CLR   R7


        DIV   R6,R7                 DIV BY 10
        AI    R8,>30                ASCII
        SWPB  R8
        MOVB  R8,@BUF(R5)           STORE IN BUFFER
        MOV   R7,R8                 REMAINDER
        JMP   WRITN1                LOOP
ANSWER  LI    R5,3
        LI    R6,BUF                POINT TO BUFFER
NEXTCH  MOVB  *R6+,R9               GET CHAR
        BL    @OUTCHR               O/P
        DEC   R5                    LOOP 3 TIMES
        JNE   NEXTCH
        B     *R11
*
* DATA AREA
*
WS      BSS   32                    WORKSPACE
BUF     BSS   6                     DECIMAL NO. BUFFER
*
        END   START
```

CHAPTER 7

SYSTEM PERFORMANCE

7.1 System Feasibility Assessment

7.2 System Communication and Task Execution Performance

CHAPTER 7

SYSTEM PERFORMANCE

7.1 SYSTEM FEASIBILITY ASSESSMENT

Based on our investigative study, in order to evaluate the feasibility of a multi-processor system the most important parameters that we studied were:

(a) Bus utilisation as a function of the number of processors in the system and the average processor task time.

(b) Independent I/O facilities.

(c) The probability of simultaneous conflicting actions (concurrent access of shared memory).

(d) Synchronising the actions of the various controllers in the system.

(e) The impact of interrupt control, and

(f) The problem of deadlocks or infinite cycles within the system.

Having studied these parameters we designed and constructed the system as already explained. Processors are given equal fixed tasks. Upon completion of the task, the processor transmits a single data item to the shared memory, for the other processor to select. Each processor monitors the shared memory for each data that has been assigned for.

Each processor makes sure that data assigned for the other processor have been selected before producing new ones. Thus the microprocessor chip is desirable for microprocessor control should provide machine cycle states information, a READY control unit which allows the microprocessor to enter a WAIT state, a HOLD input, good I/O facilities and interrupt facilities. The 8008 is characterised by a five state processor cycle, with each state requiring 2.8 μs. When the 8008 reads from memory, to get the next instruction, it presents on its data bus during state T1 the lower eight bits of the desired address. It proceeds to state T2 where the upper six address bits an indication that the 8008 wants to read appear on the bus. If by the end of T2, the memory has not responded READY, indicating that the desired byte is being presented on the bus, the 8008 goes into the WAIT state, where it remains for as many as four microsecond periods as are needed for memory to respond READY. In the event that memory responds READY before the end of T2, T3 is entered and the byte brought into the CPU. If necessary states T4 and possibly T5 are used to execute the instructions. The maximum memory bandwidth capable of being utilised by the processor is eight bits every twelve (12)μsecs . The cycle time of the memory is about 700 n.s. that is one fourth of the time required for a single processor state.

7.2 SYSTEM COMMUNICATION AND TASK EXECUTION PERFORMANCE

The communication and program execution between the 8008's system and the T.I. 9900 was satisfactory. The program execution in particular of the task orientated 8008's with the more powerful 9900 was good. The only problem appeared in the switching on the 8008's which caused jamming of the programs and the breakdown of the systems operation. On the otherhand I/O operations, private memory accessing, program execution, logic operations were quite satisfactory. Operations for accessing the shared memory were as expected. It appears for this particular application, that the waiting of one processor, when the other one was accessing the shared was negligible. The processor's request for shared memory access is completed in one memory cycle. The processor remained in the same processor state for one more memory time during which the other processor can access the shared memory.

Generally the system operated as it was predicted from the design stages.

Analysing the performance of our approach, that is every other two processors per common memory, we feel is ideal for localised control systems with efficient I/O facilities, task decentralisation becomes easy, utilisation of individual processor performance is fully

exploited, and an easy monitor system for central control.

Our approach also provide system expandability as an addition of 1 or 2 more processors to share the same memory is feasible. We must though stress at this point the hardware problems associated with it. At the same time it must be realised that in a design of that form some form of priority level to processors accessing the shared memory is required.

Clearly our system provides the mechanism whereby programs (the specialised application programs or any of the stand-alone programming support aids) can be loaded and run on the actual microprocessor hardware. Object programs are normally written onto PROM, loaded and run into RAM and the shared memory serves for inter-processor communications of common data and information for individual processor task completion.

The system also provides small physical size and power consumption, as well as reliability.

CHAPTER 8

CONCLUSIONS

8.1 General

8.2 Suggestions for Future Development

8.3 Final Remarks

CHAPTER 8

CONCLUSIONS

8.1 GENERAL

The microprocessor revolution has made possible the economical decentralization of computing power. This has been achieved not necessarily by making system with improved price/performance, but by making computer control of many functions economical and practical, relative to their previous implementations. Multi-microprocessor structures, we feel, are effective in situations where the tasks to be performed can be effectively and efficiently partitioned. This will give rise to improved I/O processing capability, improved reliability, and a fail-soft feature where the bulk of the system can keep operating, should any subsystem fail. An additional benefit resulting from the effective partitioning of tasks in a multi-CPU system is that the software by being partitioned into several independent packages, is much simpler and runs more efficiently.

Thus a step forward, we feel, in the direction of releasing more of the potential capability of the microprocessor is to provide processors with distributed multiple tasks. When the control area of the microprocessor increases, the engineer at present turns to the greater sophistication of the minicomputer with its real time,

multi-programming executive. The difficulty in using a microprocessor is that the single program may not have sufficient processor time constraints when it has wasted most of it in completing one part before proceeding to the next. A multi-processor multi-task system, like the one we advocate simplifies the problem considerably by allowing the total job to be broken down into separately identifiable activities. The non-time critical I/O can then be placed under task control rather than interrupt control, obviating the need for a priority interrupt structure in many cases.

One of the fundamental design decisions in a multi-tank system, we found, is the determination of the distribution of tasks. Generally there are two ways:

1. Task distribution is determined by external interrupts, and
2. Task distribution is determined only by the tasks themselves.

We adopted the second method, as it is suitable for localised control in the manner in which it handles data structure relationships, simplicity on interprocessor communications, and I/O processing flexibility.

Our design was based on the application task that some parts of a program are run more frequently than other parts that run less frequently. This fact allows a

significant advantage to be gained by the use of private memory. With this configuration the ratio of accesses to local versus shared memory, could be as high as 3-4 to 1. This not only reduces contention delays for access to the shared memory, but also cuts the number of accesses which suffer the delays. We designed all processors to be identical and equal. As a consequence, no single processor is of vital importance and a processor could be changed easily in case of failure. Apart from system flexibility an additional advantage of multiple copies is reliability.

Until the processors interact, a multi-processor is a number of independent single processor system. It is the interaction wich poses the conceptual as well as the practical challenge. If the various processors spend their time waiting for each other, the system degrades to a single processor equivalent. If they can usefully run concurrently, maximizing at the same time shared memory utilization, then the system's power is being multiplied.

In any practical application of a multi-processor system, we feel that we must keep the system running in the case of module failure. The first problem in doing this is making the processors run independently

by allocation of runnable task to processors, so that the full requisite power can be quickly brought to bear on high priority tasks. We propose four ways of doing this to help manage tasks queues:

(a) Break the job into small tasks

(b) Make the processors identical

(c) Keep a priority on tasks and

(d) Use interrupts where necessary

Critical to our approach is the fact that the private memory of its processor could be used as a retreat to local operation in the face of systems' problems.

Our system has a great application in localized system control. The author visited the British Leyland Longbridge plant, where they are trying to design a similar system on localized monitor control on the production lines, with each multiprocessor unit communicating with a central control terminal, and they express great interest in the possibilities of using our approach.

Our system offers flexibility of I/O processing, it is easy to expand, easy to install and offers a reliable localized control. A set of identical processors sharing every other two a common memory, executing individually an identified task, intercommunicating and at the same time communicating with a central control terminal, would have great industrial control applications.

It is also feasible to expand the system by connecting a third or fourth processor to the system still accessing the same common memory. It must be realized that in a design of that form some form of priority level to processor accessing memory is required. On that basis the processor would wait for access after the completion of access by the microprocessor with higher priority. The microprocessor with the higher priority does not wait for memory access. Thus the number of memory cycles that a microprocess could have to wait to get hold of the memory access bus, depends on the number of priority that particular processor has, and the total number of microprocessors in the system.

It is thought that the incremental growth objective is realizable at least in terms of minimizing the effects of memory contention in a network of microprocessors sharing a memory.

8.2 SUGGESTIONS FOR FUTURE DEVELOPMENT

It is apparent that there is a need for further research and development to be performed to assess and develop a multi-processor, multi-task control system.

At this point we must stress the role of interrupts in a multi-control system. It was explained in a previous chapter why we did not use interrupts in our particular system.

The problem here exists if any particular task, event or data from external device has to be handled by the system. This event or data could require attention by the system at unpredictable times (asynchronous). The problem is that, unless the system 'looks for' the event that requires attention (our approach), then it could quite easily 'miss' the event, particularly if it only lasted for a short time. Interrupts is the answer to this problem and could be either hardware or software.

Manufacturers of latest designs of microprocessors have provided their micros with extended interrupts facilities and that helps greatly the system designer and user.

This is an area which must be investigated further in the different applications of our system.

It must be said, without underestimating the capabilities and flexibility of the 8008, and having realized its weak points, a system with more powerful microprocessor (Motorola 6800, Intel 8080), would be more suitable for complicated applications, as tasks and problems in practical industrial applications, tend to be more demanding.

As we have already pointed out, the problems to be considered in the design of multiprocessor systems, would depend on the applications, designer and user. For example should the individual processors be dedicated to totally independent programs or should they work

co-operatively on a single large problem to reduce execution time and promote reliability. In the former case considerations are not limited to production environments. One can visualize a situation in which program development is performed by a user accessing a set(s) of multiprocessors from a dedicated terminal. In either case the problem of executive control to implement inter-processor communication, memory protection, memory mapping and shared data memories and buses are significant. Should a single executive, control the total system, or should the executive be partitioned in to global and local executives ?

We feel that in less dynamic systems, in which the same sets of code are executed repeatedly, private memories should be dedicated to each processor to contain procedure segments and the shared memory can be used to contain data of both private and shared nature.

8.3 FINAL REMARKS

At present multiprocessor systems have started appearing in all areas of applications in industrial and organizational fields. Different specialized multi-user systems and high bandwidth signal processing are being used.

As improvements in integrated circuit technology continue and processors and memories become cheaper, and smaller multi-processor systems designs offer a

revolutionary challenge to the foundations of industrial production and organizational change, structure and innovation, and to society as a whole.

I trust that the system we have designed and the approach we have followed will be given the opportunity to demonstrate its uses, and that it will contribute to the future and further development and understanding of multi-processor control systems.

LIST OF SYMBOLS

T_s	System throughput
T_p	Bus interference
N	Number of processors
β	Bus utilization
P	Distribution probability of bus reference every i bus cycles
R	Ratio of throughput with maximum interference to throughput with no interference
C_R	Ratio of the cost of the system
C_P	Cost of an individual microprocessor

APPENDICES

APPENDIX A

INTEL'S 8008 MICROPROCESSOR

Intel is a USA-based component manufacturer and the 8008 processor is the central component of the MCS-8 Microprocessor Set. The 8008 can include up to 16K 8-bit words of RAM or ROM and its a parallel processor with an 8-bit external bus for communication with memories and I/O devices. It is manufactured using Silicon Gate MOS technology.

The 8008 processor is shown in Fig. A1. Two independent dynamic memories are used to implement a stack of 8 14-bit address registers and 7 8-bit scratch pad registers. The address stack consists of the program counter and 7 address registers for subroutine nesting to 7 levels. The CALL instruction is used to store automatically the program counter in the stack, and RETURN is used to restore the program counter. The 14-bit program counter allows direct addressing of 16K words of memory for program instructions. Each 14-bit address is transmitted over the I/O bus in two cycles, consisting of the 8 lower order bits followed by the 6 higher order bits.

The scratch pad memory contains the accumulator used for mathematical and logical operations and used as the destination for data operations and never to store data.

The next 4 registers are used for temporary storage and to transfer data between program modules. The last two registers (H , and L) are normally dedicated to addressing external memory for data.

The arithmetic/logic unit performs full-parallel 8-bit operations. Four single bit indicators are set as a result of each operation. These are carry, zero, sign, and parity.

When the processor supply (V_{DD}) and clocks are started, a HALT instruction is automatically stored in the instruction register and the system registers are reset in the following 16 clcock periods. Normal operation commences when the INTERRUPT line is set from a source external to the processor chip.

All communication between functional units in the 8008 processor occurs via a single-8-bit internal bus. The processor controls the bus and sets the 3 status lines, S, O:2, according to the action occurring on the bus. The status lines are available as o/p's to peripheral circuitry. A typical cycle of processor operation consists of 5 states: 2 for addressing memor; 1 for fetching an instruction or data, and 2 for instructing execution. For multiple cycle instructions, which do not require the 2 execution states, the processor operates asynchronously. One instruction cycle takes 12.5 μs to be executed. (See Fig.A2).

The 8008 uses, 1, 2, and 3 byte formats for its instructions. The 2 byte instructions perform the operation specified by the first byte on the data specified in the second byte (immediate mode). The 3-byte instructions use 2 bytes to specify a 14-bit memory address for jumps and calls. There are 5 basic groups of instructions, the index register group, the accumulator group, the instruction program counter and stack control group, I/O group and 2 HALT instructions.

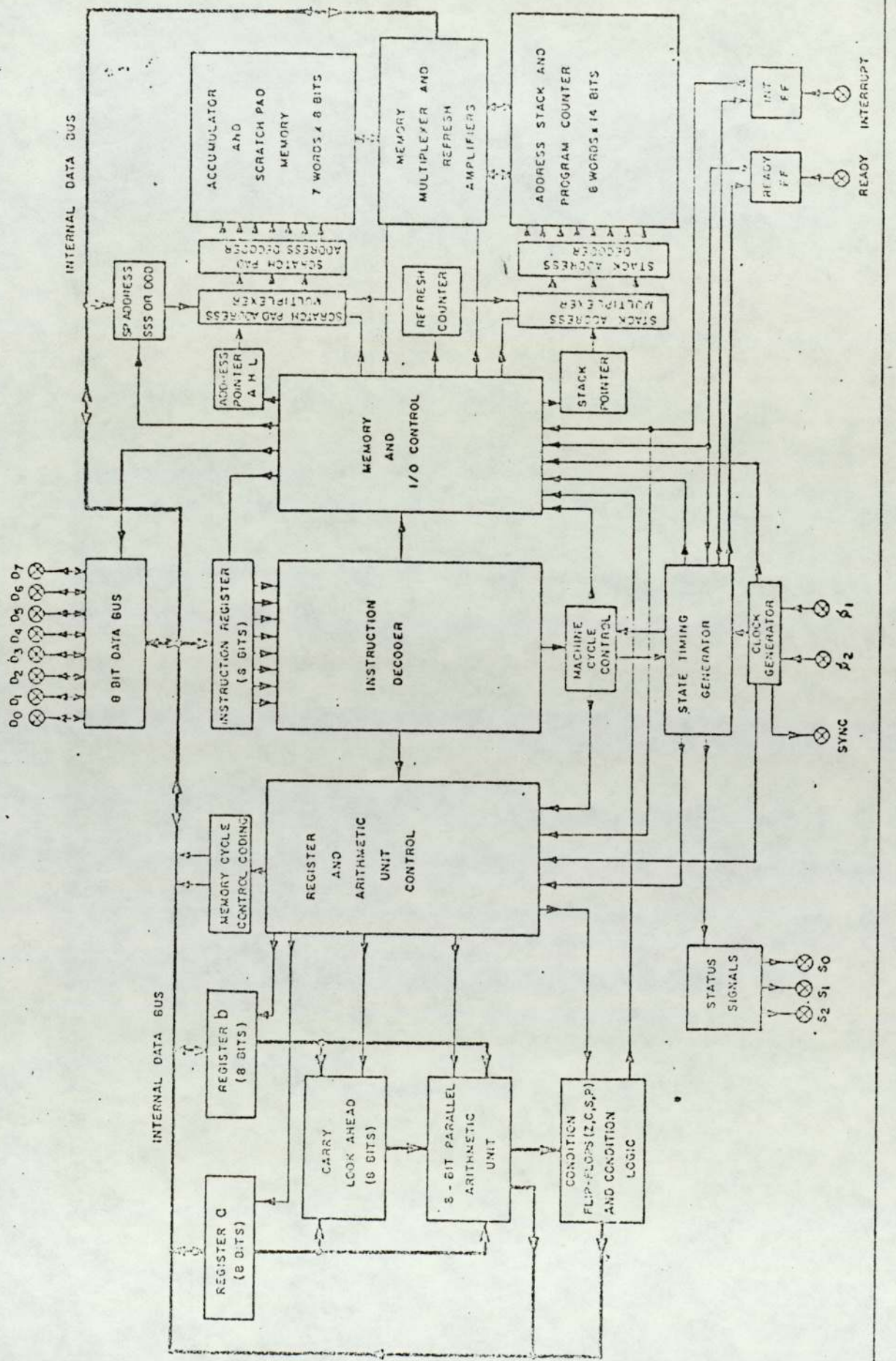
The 8008 communicates with external memories and I/O device controllers via its 8-bit data bus, the 3 status line, the SYNC line and the READY and INTERRUPT lines. The READY line allows the processor to operate with any speed of semiconductor memory. This is achieved by the processor waiting on the READY line during an instruction cycle.

The I/O data buffer on the 8008 chip is bi-directional with low power TTL compatibility on the o/p and TTL compatibility on the input.

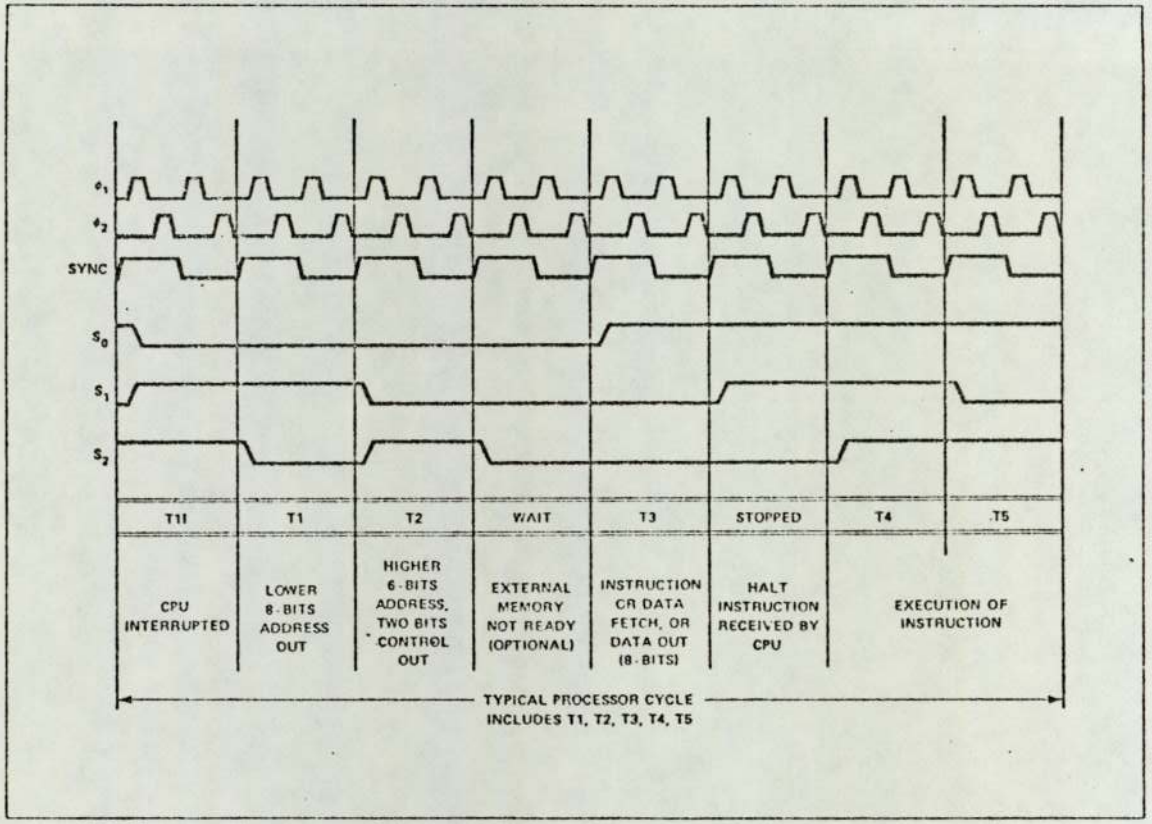
General purpose software had been developed for the 8008 by Intel including loaders, teletype input, different routines and FORTRAN IV assembler and simulator which allowed the generation and testing of 8008 programs on a large off-line computer.

As has been stressed before the 8008 belongs to the first generation microprocessors and since then in the last three years, the technology and capabilities of the 3rd generation microprocessor have overpowered

the uses of the 8008, but it is still used in some acpplications.



A.1 8008 CPU Block Diagram



A.2 8008 Instruction Cycle

APPENDIX B

THE TMS 9900 MICROPROCESSOR

The TMS 9900 microprocessor is a single-chip 16-bit central processing unit (CPU), produced using N-channel silicon-gate MOS technology. The CPU comes in a 64 pin chip, and is driven by a 3 MHz four-phase clock.

The processor employs a memory-to-memory form of architecture, whereby blocks of memory designated as workspace registers, replace the more common internal hardware registers. A total of 32 K words of memory can be addressed by the processors 15-bit address bus, which is separate from the 16-bit data bus, thus simplifying the system design.

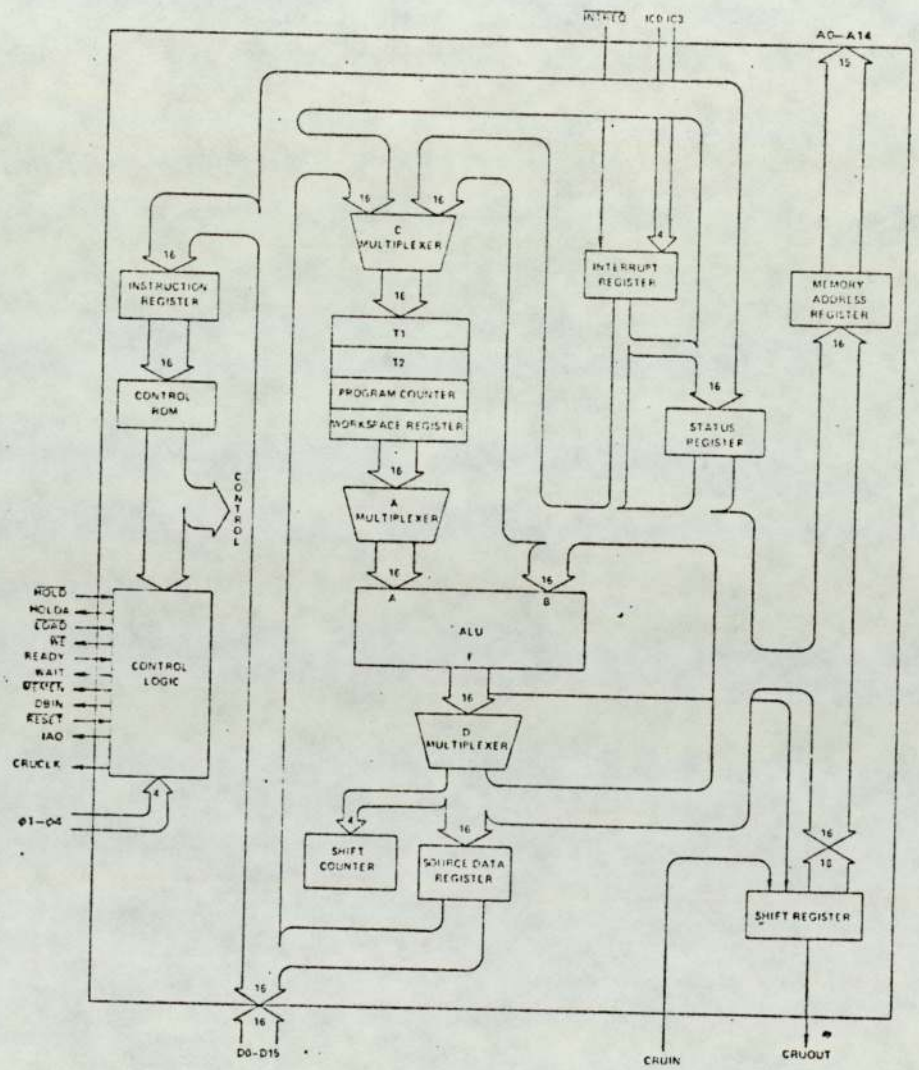
Within the processor there are three registers which are accessible by the user. These are the program counter (PC) which contains the address of the instruction following the current instruction being executed, the status register (ST) which contains the interrupt mask level and status information relating to the instruction operation. The third and final register is the workspace point (WP), which contains the address of the first word in the current active workspace area. A workspace area consists of 16 consecutive memory words in the general memory area.

Input and output data transfers to and from the processor are performed by a direct command-driven I/O

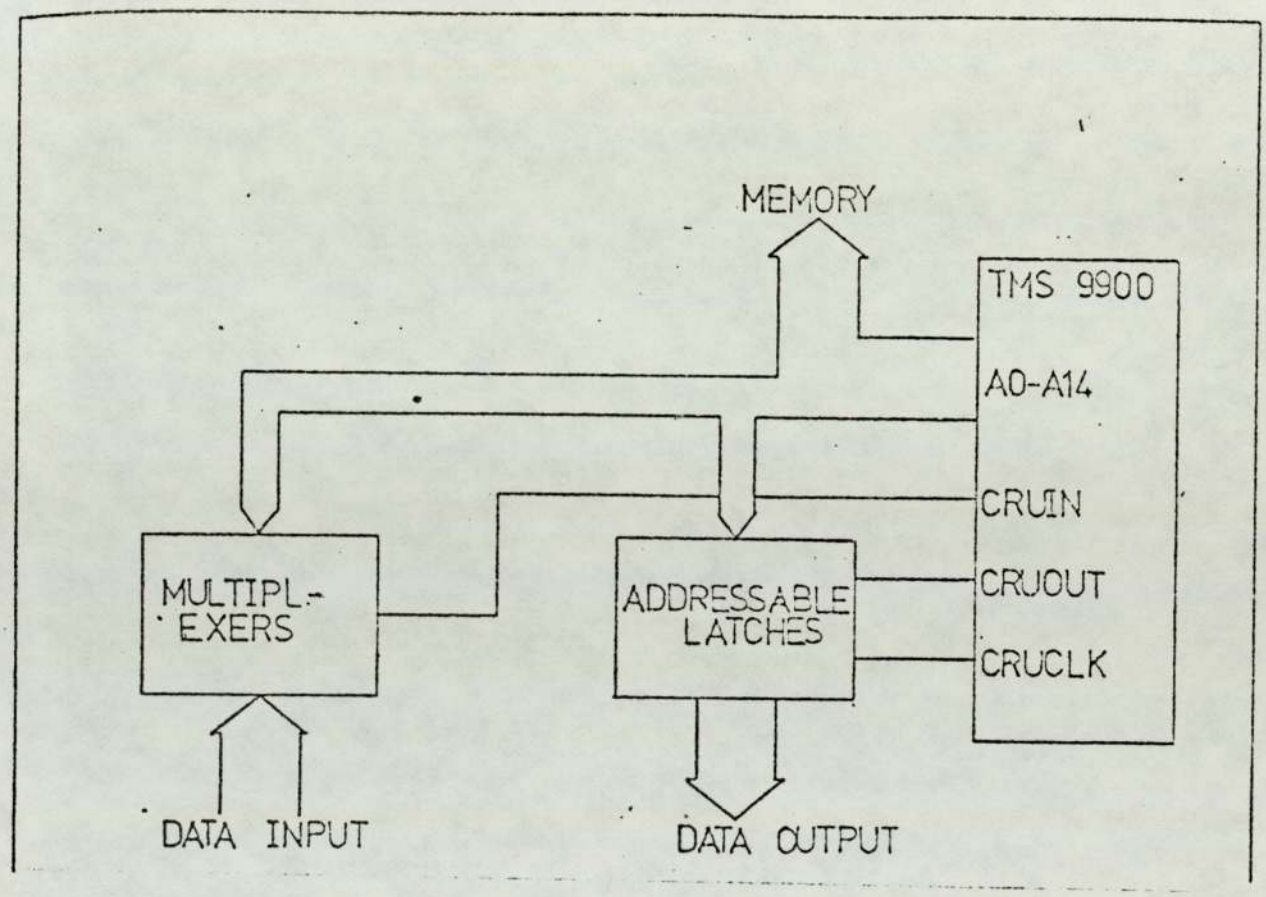
interface designated as the communications-register unit (CRU). The CRU provides up to 4096 directly addressable input and output bits. Both input and output bits can be addressed individually or in fields of 1 to 16 bits. The TMS 9900 uses four clock cycles (ϕ_1-ϕ_4) each of typical duration of 83 ns.

The TMS 9900 microprocessor instruction set provides the same capabilities as those offered by full minicomputers. The instruction set provides 69 different instructions, which includes unsigned multiply and divide instructions.

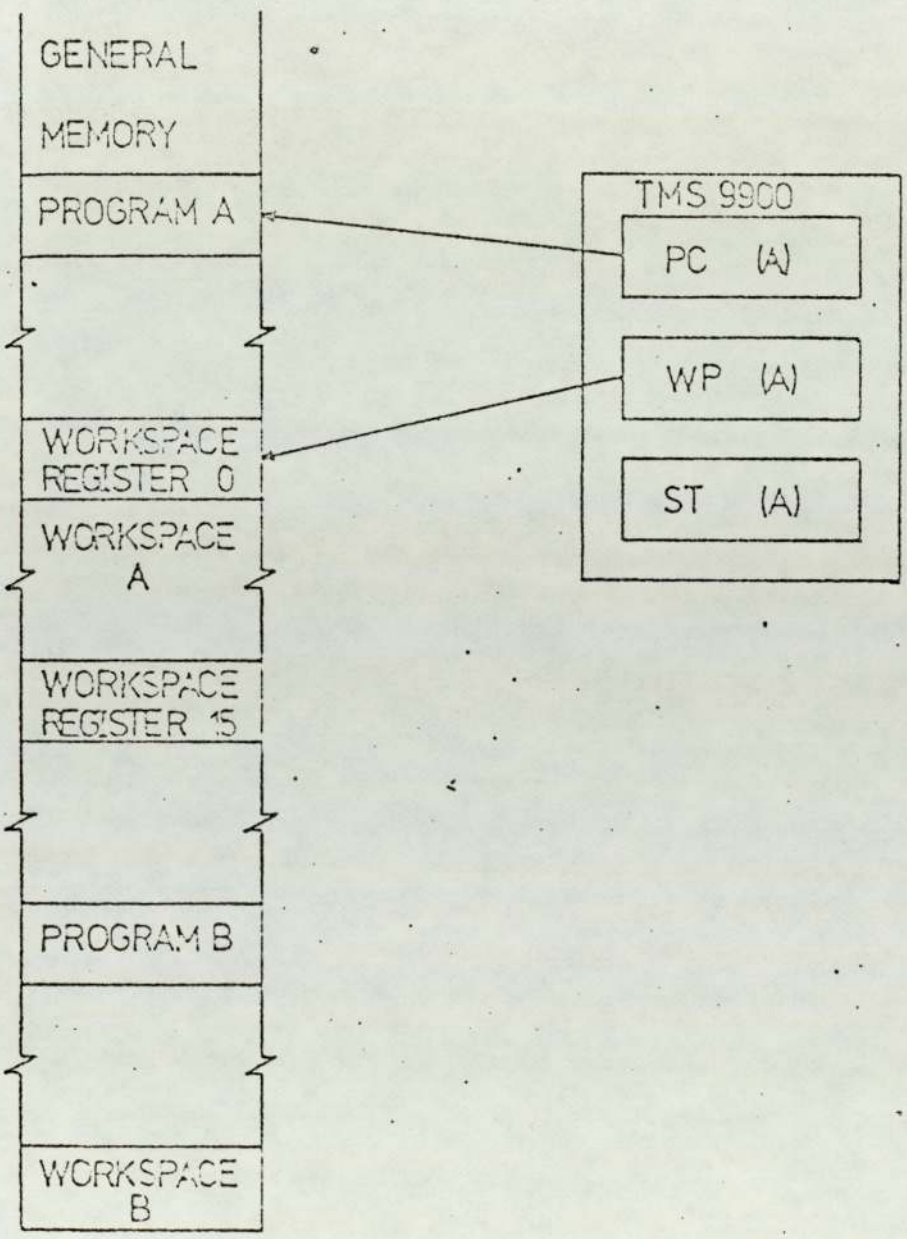
With a clock frequency of 3 MHz the average instruction execution time is approximately 10 μs.



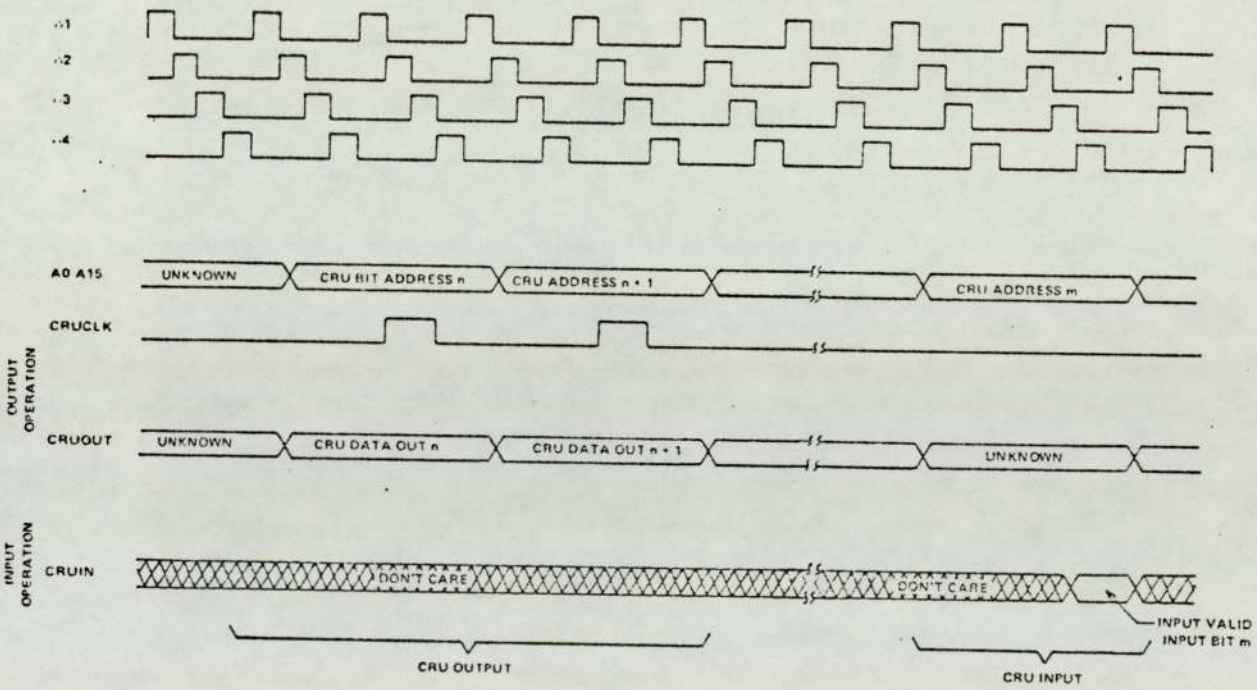
B.1 TMS 9900 Architecture



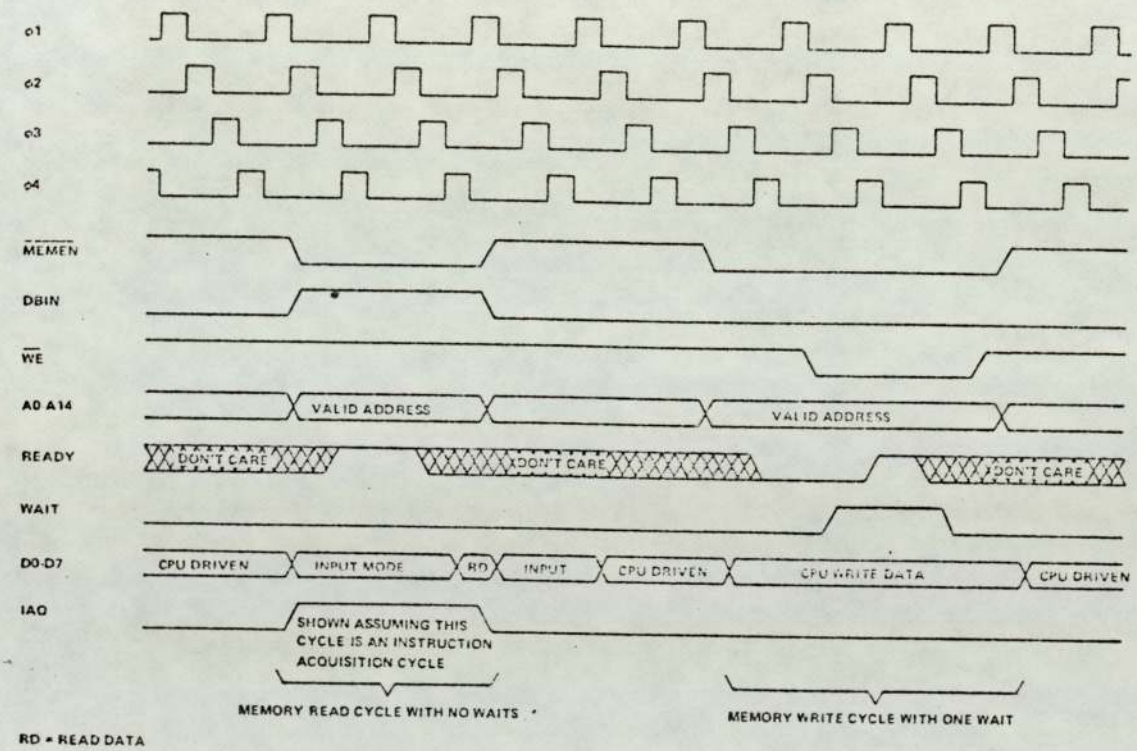
B.2 CRU Concept I/O Interface Logic



B.3 Relationship between Workspace Pointer and Corresponding Workspace



B.4 TMS 9900 CRU Interface Timing



B.5 TMS 9900 Memory Bus Timing

APPENDIX C

MULTI-PROCESSOR SHARED MEMORY TEST PROGRAMS

'SLOW COUNT' PROGRAM

MICROPROCESSOR 1

```
        XRA
        LBA
LOOP    INB
        LAB
        LHI     14B
        LLI     100B
        LMA
        LCI     377B
        LDI     100B
KEY     DCC
        JFZ     KEY
        DCD
        JFZ     KEY
        JMP     LOOP
```

'SLOW COUNT' PROGRAM

MICROPROCESSOR 2

```
XRA
LAA
LHI     14B
LLI     100B
LAM
OUT     10B
XRI     377B
OUT     11B
RST     0
```

MULTI-PROCESSOR SHARE MEMORY TEST PROGRAM

"FAST COUNT" PROGRAM

MICROPROCESSOR 1

```
        XRA
        LAA
        LHI     14B
        LLI     200B
        LCA
        LAA
TOP     LAM
        NDA
        JFZ     TOP
        INC
        LAC
        OUT     10B
        XRI     377B
        OUT     11B
        LMI     1
        RST     1
```

"FAST COUNT" PROGRAM

MICROPROCESSOR 2

```
        XRA
        LAA
        LHI     14B
        LLI     200B
        LMA
        LBA
TOP     LAM
        NDA
        JTZ     TOP
        LMB
        LAM
        OUT     10B
        XRI     377B
        OUT     11B
        RST     1
```

TEST PROGRAM FOR DECODING RAM AND PROM MEMORIES

```
TOP     XRA
        LAA
        LCA
        LDA
        LHA
        LLA
        LAA!LAA
LOCK    LCI         100B
KEY     DCC
        JFZ         KEY
        INL
        LAM
        OUT         10B
        LAL
        CPI         377B
        JFZ         LOCK
        INH
        LAM
        OUT         10B
        LAH
        CPI         20B
        JFZ         LOCK
        RST         1
```

TEST VDU PROGRAM

```
        XRA
        LCA
        LDA
        LBA
LOOP    LAB
        OUT     11B
        LAI     60B
        OUT     10B
        XRA
        OUT     10B
        INB
        LCI     300B
        LDI     100B
KEY     DCC
        JFZ     KEY
        DCD
        JFZ     KEY
        JMP     LOOP
```

TEST PROGRAM - I/O BUSES AND VDU

```
        XRA
        LAA
        LBA
        LCA
TOP     DCB
        LAB
        OUT     11B
        LAI     50B
        OUT     10B
LOOP    INP     3B
        NDI     100
        JTZ     LOOP
        LAI     60B
        OUT     10B
        XRA
        OUT     10B
        JMP     TOP
```

APPENDIX D

PROGRAM ASSEMBLY

ASSEMBLER - 8008 MICROPROCESSORS

MICROPROCESSOR 1

HEX Address	Program in HEX
00:00	A8
01	C8
02	26 - 01
04	D0
05	D8
06	06 - 32
08	2E - 08
0A	36 - 40
0C	F8
0D	06 - 64
0F	2E - 08
11	36 - 48
13	F8
14	06 - 96
16	2E - 08
18	36 - 50
1A	F8 - C8
1B	06 - C8
1D	2E - 08
1F	36 - 58
21	F8
22	06 - FA
24	2E - 08
26	36 - 60
28	F8
29	06 - 00
2B	47
2C	3C - 00
2E	68 - 2B - 00
31	2E - 08
33	36 - 40
35	CF
36	B9
37	60 - 3D - 00
3A	40 - 6D - 00
3D	2E - 08
3F	36 - 48
41	CF
42	B9
43	60 - 49 -00
46	40 - 6D -00
49	2E - 08
4B	36 - 50

```
00:4D                                   CF
   4E                                   B9
   4F                                   60 - 55 - 00
   52                                   40 - 6D - 00
   55                                   2E - 08
   57                                   36 - 58
   59                                   CF
   5A                                   B9
   5B                                   60 - 61 - 00
   5E                                   40 - 6D - 00
   61                                   2E - 08
   63                                   36 - 60
   65                                   CF
   66                                   B9
   67                                   40 - 6D - 00
   6A                                   60 - 6D - 00
   6D                                   E0
   6E                                   C1
   6F                                   55
   70                                   57
   71                                   C8
   72                                   C1
   73                                   C0
   74                                   C0
   75                                   06 - 00
   77                                   55
   78                                   57
   79                                   2E - 0C
   7B                                   36 - 40
   7D                                   FA
   7E                                   46 - A0 - 00
   81                                   2E - 0C
   83                                   36 - 80
   85                                   C7
   86                                   3C - 00
   88                                   6A - A0 - 00
   8B                                   3C - 00
   8D                                   68 - 81 - 00
   90                                   55
   91                                   57
   92                                   C0
   93                                   C0
   94                                   06 - 00
   96                                   55
   97                                   57
   98                                   2E - 0C
   9A                                   36 - 80
   9C                                   F8
   9D                                   44 - 2B - 00
   A0                                   1E - BF
   A2                                   19
   A3                                   48 - A0 - 00
   A6                                   07
```

MICROPROCESSOR 2

HEX Address	Program in HEX
00:00	A8
01	C8
02	D0
03	D8
04	26 - 01
06	06 - 46
08	2E - 08
0A	36 - 40
0C	F8
0D	06 - 46
0F	2E - 08
11	36 - 48
13	F8
14	06 - 28
16	2E - 08
18	36 - 50
1A	F8
1B	06 - 1E
1D	2E - 08
1F	36 - 58
21	F8
22	06 - 14
24	2E - 08
26	36 - 60
28	F8
29	06 - 00
2B	2E - 0C
2D	36 - 40
2F	C7
30	3C - 00
32	6A - 9D - 00
35	3C - 00
37	68 - 2B - 00
3A	E0
3B	C8
3C	06 - 00
3E	2E - 0C
40	36 - 40
42	F8
43	C1
44	C4
45	3C - 32
47	68 - 5E -00
4A	3C - 32
4C	68 - 69 - 00
4F	3C - 96
51	68 - 74 - 00
54	3C - C8
56	68 - 7F - 00

```
00:59                                   3C - FA
   5B                                   68 - 8A - 00
   5E                                   2E - 08
   60                                   36 - 40
   62                                   C7
   63                                   46 - 95 - 00
   66                                   44 - 2B - 00
   69                                   2E - 08
   6B                                   36 - 48
   6D                                   C7
   6E                                   46 - 95 - 00
   71                                   44 - 2B - 00
   74                                   2E - 08
   76                                   36 - 50
   78                                   C7
   79                                   46 - 95 - 00
   7C                                   44 - 2B - 00
   7F                                   2E - 08
   81                                   36 - 58
   83                                   C7
   84                                   46 - 95 - 00
   87                                   44 - 2B - 00
   8A                                   2E - 08
   8C                                   36 - 60
   8E                                   C7
   8F                                   46 - 95 - 00
   92                                   44 - 2B - 00
   95                                   2E - 0C
   97                                   36 - 80
   99                                   F8
   9A                                   C0
   9B                                   C0
   9C                                   07
   9D                                   1E - BF
   A0                                   19
   A1                                   48 - 9D - 00
   A4                                   07
```

```
                SDSMAC      3.1 *    14:15:43 THURSDAY, AUG 16, 1979.
ACCESS NAMES TABLE                                                    PAGE

SOURCE ACCESS NAME=         SCR2.ELIAS
OBJECT ACCESS NAME=         DUMY
LISTING ACCESS NAME=        SCR2.ELIASL
ERROR ACCESS NAME=          DUMY
OPTIONS=
MACRO LIBRARY PATHNAME=
```

```
TAD             SDSMAC      3.1 *    14:15:43 THURSDAY, AUG 16, 1979.
TASK ALLOCATION "DISTANCE"                                            PAGE
 0002
0003 0000 02E0  START  LWPI WS
     0002 013A'
0004 0004 06A0         BL   @INIVDU        INITIALISE UART
     0006 0070'
0005      0008' NEXTO  EQU  $
0006 0008 06A0         BL   @OUTXT
     000A 0086'
0007 000C 0D0A         DATA >0D0A
0008 000E   44         TEXT 'DISTANCE='
     000F   49
     0010   53
     0011   54
     0012   41
     0013   4E
     0014   43
     0015   45
     0016   3D
0009 0017   A0         BYTE ' '+>80
0010 0018 06A0         BL   @READNO        INPUT NUMBER
     001A 00BC'
0011 001C 020C         LI   R12,>1000      ; OUTPUT
     001E 1000
0012 0020 06C8         SWPB R8             ;  DISTANCE
0013 0022 3208         LDCR R8,8           ;  TO 8008
0014 0024 04C0         CLR  R0                 ;WAIT FOR READY
0015 0026 3600  TEST1  STCR R0,8               ; SIGNAL FROM
0016 0028 13FE         JEQ  TEST1              ;  8008
0017 002A 1000         NOP                 DELAY
0018 002C 1000         NOP
0019 002E 3602         STCR R2,8           RANGE FROM 8008
0020 0030 04C0         CLR  R0
0021 0032 3200         LDCR R0,8            O/P READY TO 8008
0022 0034 3600  TEST2  STCR R0,8           ;WAIT FOR
0023 0036 16FE         JNE  TEST2          ;ACKNOWLEGE
0024 0038 3600  TEST3  STCR R0,8               ;WAIT READY
0025 003A 13FE         JEQ  TEST3              ; SIGNAL
0026 003C 1000         NOP                     ;
0027 003E 1000         NOP                     ;
0028 0040 3603         STCR R3,8           ELEVATION FROM 8008
0029 0042 06A0         BL   @OUTXT
     0044 0086'
```

```
0031 0048   52          DATA >0D0A
     0049   41          TEXT 'RANGE='
     004A   4E
     004B   47
     004C   45
     004D   3D
0032 004E   A0          BYTE ' '+>80
0033 0050 C202          MOV  R2,R8
0034 0052 06A0          BL   @WRITNO        DISPLAY RANGE
     0054 0108'
0035 0056 06A0          BL   @OUTXT
     0058 0086'
0036 005A 0D0A          DATA >0D0A
0037 005C   45          TEXT 'ELEVATION='
     005D   4C
     005E   45
     005F   56
```



```
     0060   41
     0061   54
     0062   49
     0063   4F
     0064   4E
     0065   3D
0038 0066   A0          BYTE ' '+>80
0039 0068 C203          MOV  R3,R8
0040 006A 06A0          BL   @WRITNO        DISPLAY ELEVATION
     006C 0108'
0041 006E 10CC          JMP  NEXT0          ENDLESS LOOP
0042                    •
0043                    •••••••••••••••••••••••••
0044                    •
0045                    • ROUTINE TO INITIALISE UART
0046                    •
0047 0070 020C  INIVDU  LI   R12,>1B40      CRU BASE
     0072 1B40
0048 0074 1D1F          SBO  31             RESET
0049 0076 0200          LI   R0,>6200
     0078 6200
0050 007A 3200          LDCR R0,8           CTRL
0051 007C 1E0D          SBZ  13             NO TIMER
0052 007E 0200          LI   R0,>34         SPEED=9600
     0080 0034
0053 0082 3300          LDCR R0,12
0054 0084 045B          B    •R11
0055                    •
0056                    • ROUTINE TO OUTPUT TEXT STRING
0057                    •
0058 0086 C28B  OUTXT   MOV  R11,R10        SAVE RET.
0059 0088 D27A  OUTXT1  MOVB •R10+,R9       GET CHAR
0060 008A 06A0          BL   @OUTCHR        O/P
     008C 00AC'
0061 008E D249          MOVB R9,R9          LAST?
0062 0090 15FB          JGT  OUTXT1         NO,LOOP
0063 0092 050A          NEG  R10            ; YES,
0064 0094 024A          ANDI R10,>FFFE      ;  EVEN
     0096 FFFE
0065 0098 050A          NEG  R10            ;    ADDRESS
0066 009A 045A          B    •R10           ; & RETURN
```

```
0067                    *
0068                    * ROUTINE TO INPUT ONE CHAR
0069                    *
0070 009C 020C  INCHR   LI    R12,>1B40     CRU BASE
     009E 1B40
0071 00A0 04C9          CLR   R9
0072 00A2 1F15  INCHR1  TB    21            CHAR REC'D ?
0073 00A4 16FE          JNE   INCHR1        NO, LOOP
0074 00A6 35C9          STCR  R9,7          YES, PUT IN R9
0075 00A8 1E12          SBZ   18
0076 00AA 045B          B     *R11
0077                    *
0078                    * ROUTINE TO OUTPUT ONE CHAR
0079                    *
0080 00AC 020C  OUTCHR  LI    R12,>1B40     CRU BASE
     00AE 1B40
0081 00B0 1D10          SBO   16            RTSON
0082 00B2 1F16  OUTCH1  TB    22            TX BUFFER EMPTY?
0083 00B4 16FE          JNE   OUTCH1        NO, LOOP
```

```
TAD              SDSMAC      3.1 *    14:15:43 THURSDAY, AUG 16, 1979.
TASK ALLOCATION "DISTANCE"                                          PAGE
0084 00B6 3209          LDCR  R9,8          YES, LOAD
0085 00B8 1E10          SBZ   16
0086 00BA 045B          B     *R11
0087                    *
0088                    * ROUTINE TO INPUT NUMBER
0089                    *
0090 00BC C14B  READNO  MOV   R11,R5        SAVE RETURN
0091 00BE 0206  READ0   LI    R6,10         BASE
     00C0 000A
0092 00C2 04C7          CLR   R7
0093 00C4 06A0  READ1   BL    @INCHR        INPUT CHAR
     00C6 009C'
0094 00C8 0289          CI    R9,>0D00
     00CA 0D00
0095 00CC 130E          JEQ   READ0         END IF CR
0096 00CE 0989          SRL   R9,8          MOVE TO RHB
0097 00D0 0229          AI    R9,->30
     00D2 FFD0
0098 00D4 110C          JLT   ERROR         TOO SMALL
0099 00D6 8189          C     R9,R6
0100 00D8 140A          JHE   ERROR         TOO BIG
0101 00DA 04CA          CLR   R10
0102 00DC 39C6          MPY   R6,R7         SUM=SUM*10
0103 00DE C1C7          MOV   R7,R7
0104 00E0 1606          JNE   ERROR         OVERFLOW
0105 00E2 C1C8          MOV   R8,R7
0106 00E4 A1C9          A     R9,R7         SUM=SUM+DIGIT
0107 00E6 1803          JOC   ERROR         OVERFLOW
0108 00E8 10ED          JMP   READ1         GET NEXT DIGIT
0109 00EA C207  READ0   MOV   R7,R8         ANSWER IN R8
0110 00EC 0455          B     *R5           RETURN
```

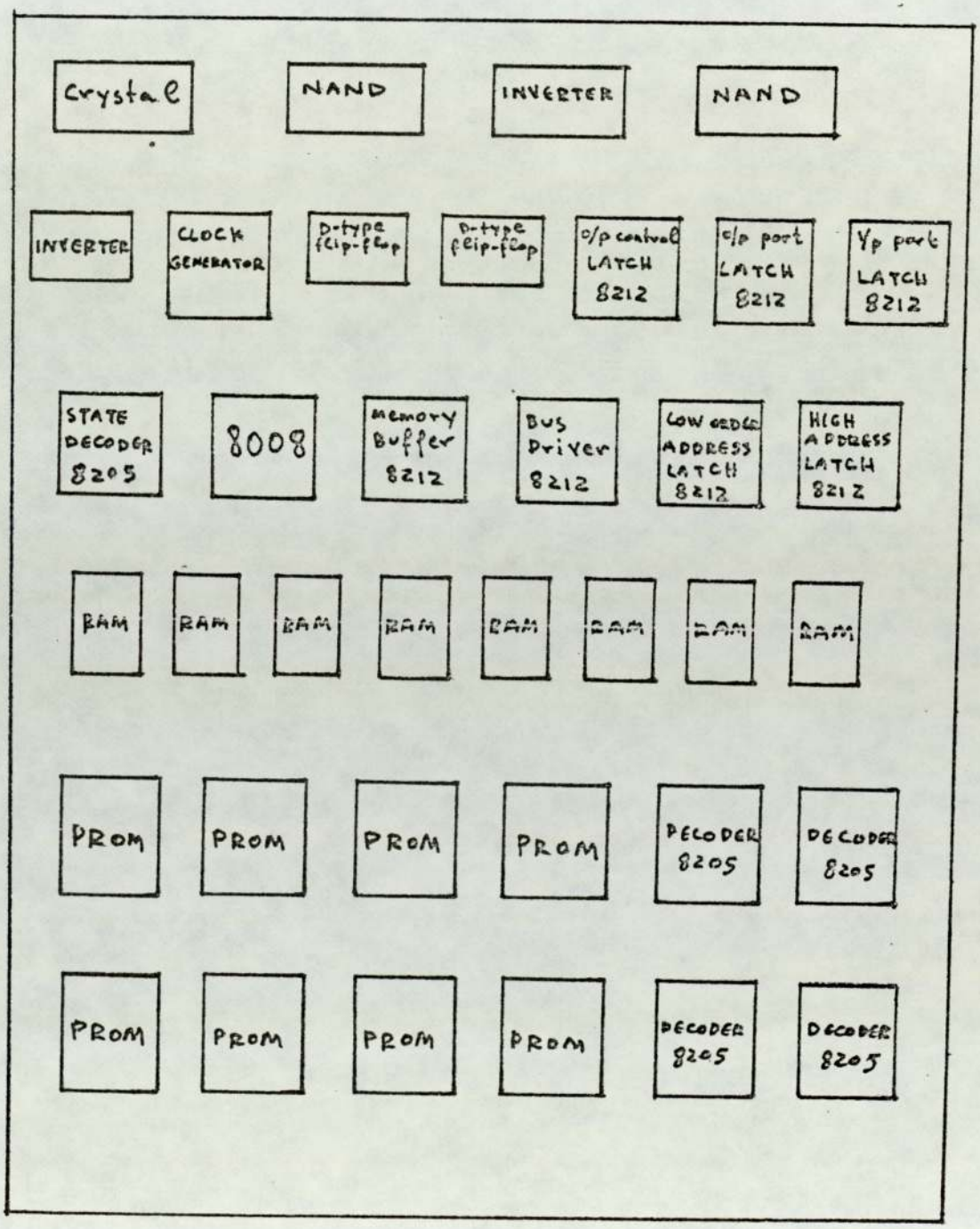
```
0111                    *
0112 00EE 06A0  ERROR   BL    @OUTXT  - 150 -    : ERROR--
     00F0 0086'
0113 00F2 0D0A          DATA >0D0A                : OUTPUT
0114 00F4   45          TEXT 'ERROR,TRY AGAIN' :  MESSAGE
     00F5   52
     00F6   52
     00F7   4F
     00F8   52
     00F9   2C
     00FA   54
     00FB   52
     00FC   59
     00FD   20
     00FE   41
     00FF   47
     0100   41
     0101   49
     0102   4E
0115 0103   0D          BYTE >0D,>0A+>80          ;
     0104   8A
0116 0106 10DB          JMP   READ0               ; THAN LOOP
0117                    *
0118                    * ROUTINE TO OUTPUT NUMBER
0119                    *
0120 0108 0205  WRITNO  LI    R5,3          MAX.3 DIGITS
     010A 0003
0121 010C 0206          LI    R6,10
     010E 000A
```

```
AD              SDSMAC      3.1 *    14:15:43 THURSDAY, AUG 16, 1979.
ASK ALLOCATION "DISTANCE"                                          PAGE 0
0122 0110 0605  WRITN1  DEC   R5
0123 0112 1109          JLT   ANSWER
0124 0114 04C7          CLR   R7
0125 0116 3DC6          DIV   R6,R7           DIV BY 10
0126 0118 0228          AI    R8,>30          ASCII
     011A 0030
0127 011C 06C8          SWPB  R8
0128 011E D948          MOVB  R8,@BUF(R5)     STORE IN BUFFER
     0120 015A'
0129 0122 C207          MOV   R7,R8           REMAINDER
0130 0124 10F5          JMP   WRITN1          LOOP
0131 0126 0205  ANSWER  LI    R5,3
     0128 0003
0132 012A 0206          LI    R6,BUF          POINT TO BUFFER
     012C 015A'
0133 012E D276  NEXTCH  MOVB  *R6+,R9         GET CHAR
0134 0130 06A0          BL    @OUTCHR         O/P
     0132 00AC'
0135 0134 0605          DEC   R5              LOOP 3 TIMES
0136 0136 16FB          JNE   NEXTCH
0137 0138 045B          B     *R11
0138                    *
0139                    * DATA AREA
0140                    *
0141 013A       WS      BSS   32              WORKSPACE
0142 015A       BUF     BSS   6               DECIMAL NO. BUFFER
0143                    *
0144      0000'         END   START
 ERRORS
```

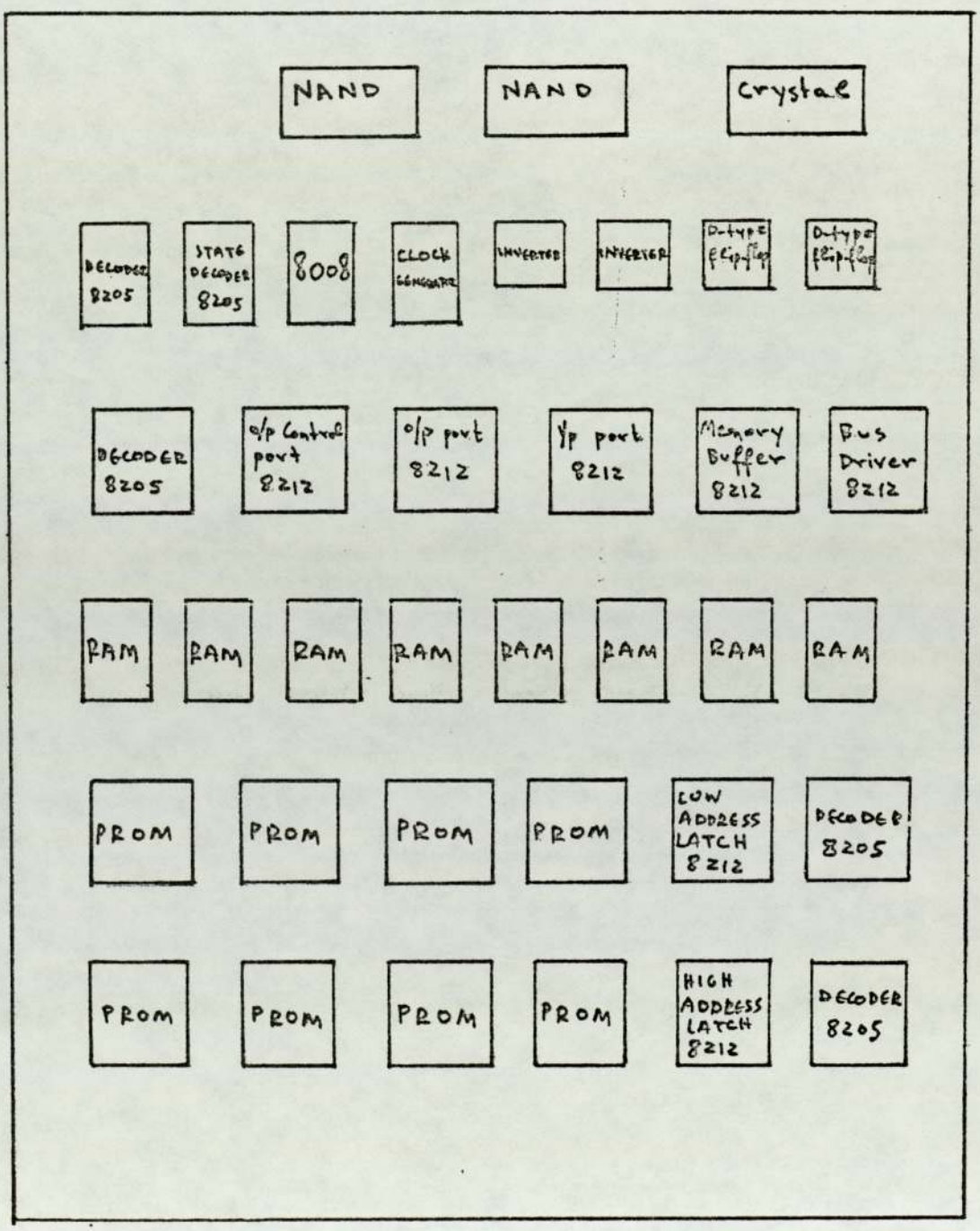
APPENDIX E

BOARD LAYOUTS

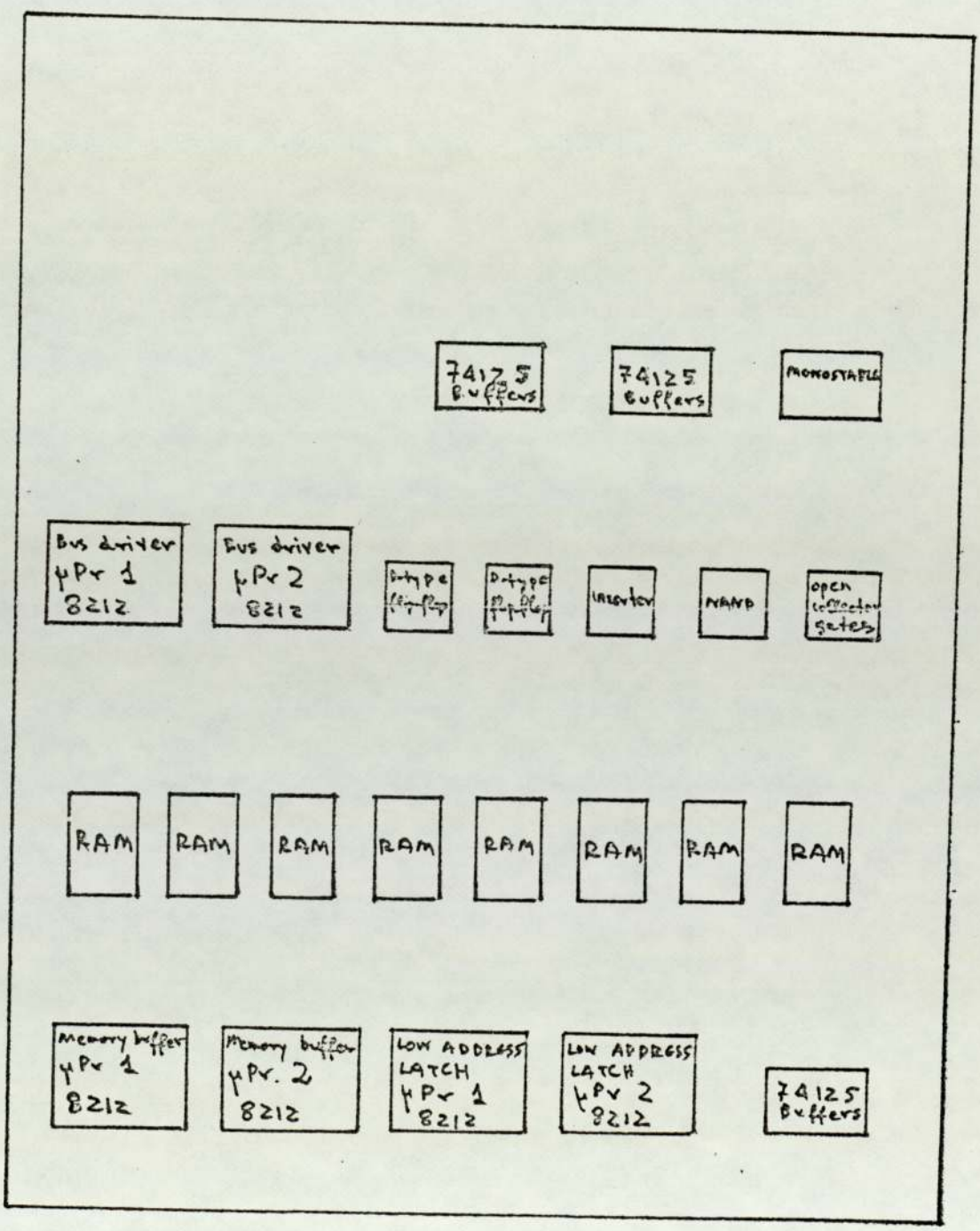
MICROPROCESSOR 1



MICROPROCESSOR 2



SHARED MEMORY BOARD LAYOUT



APPENDIX F

SHARED MEMORY BOARD CONNECTIONS WITH THE TWO PROCESSORS

MICROPROCESSOR 1

Connections

4 to 11	:	Memory buffer o/p's
12	:	WAIT 1
13	:	A_8 address line from the high address latch
14 to 21	:	Bus driver o/p's
23	:	P1O3 line
24	:	T3A line
25	:	$\overline{T1}$
44	:	A_9 address line from the high address latch
59	:	$\overline{PCI}.\overline{PCR}$ line
60	:	DBIN
61	:	$\overline{PCW}.\overline{T3}$ line
62 to 69	:	Low address latch o/p's

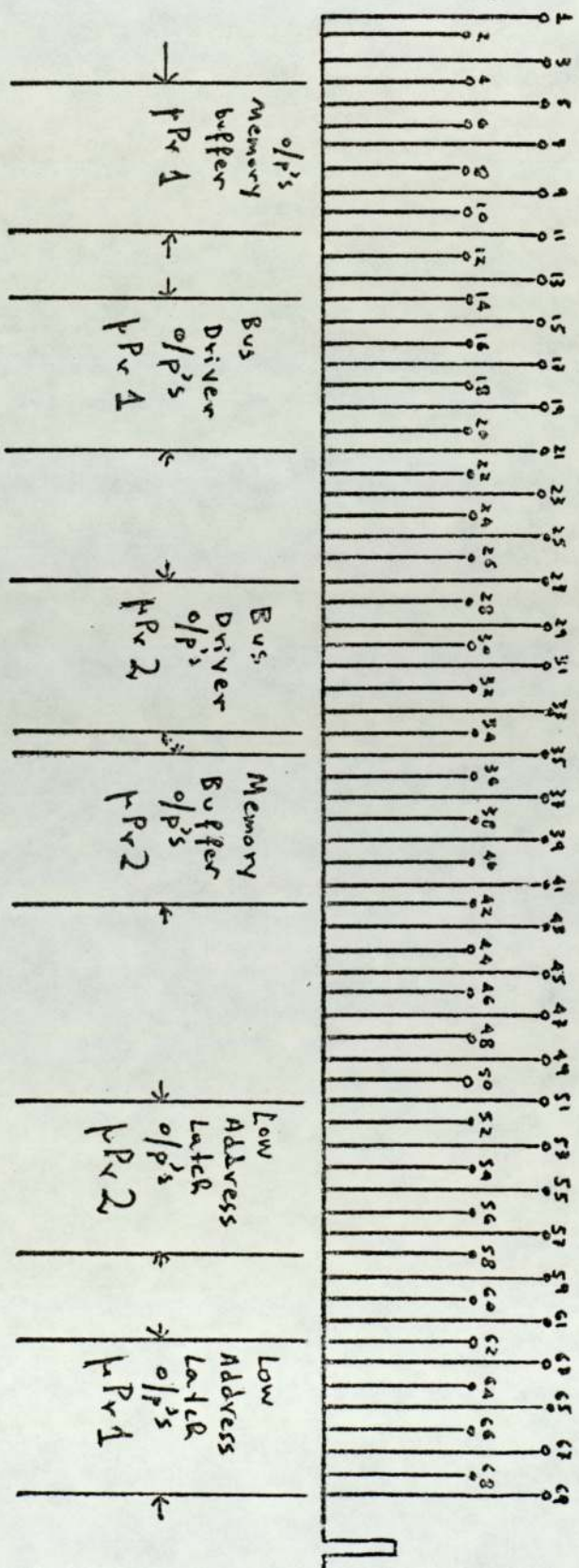
SHARED MEMORY BOARD CONNECTIONS WITH THE TWO PROCESSORS

MICROPROCESSOR 2

Connections	27 to 34	:	Bus driver o/p's
	43	:	A_8 address line
	26	:	A_9 addres line
	35 to 42	:	Memory buffer o/p's
	22	:	WAIT 2
	45	:	P203 line
	46	:	T3A
	47	:	$\overline{T1}$
	48	:	$\overline{PCW}.\overline{T3}$
	58 to 51	:	Low address latch o/p's
	49	:	DBIN
	50	:	$\overline{\overline{PCI}.\overline{PCR}}$ line

SHARED MEMORY BOARD PIN CONNECTIONS

WITH THE TWO PROCESSORS



APPENDIX G

INTERFACE CONNECTIONS BETWEEN THE 8008 AND THE 9900 PROCESSORS

9900 Connections

51 : Memory Address 0
50 : Memory Address 1
49 : Memory Address 2
48 : Memory Address 3
47 : Memory Address 4
46 : Memory Address 5
45 : Memory Address 6
44 : Memory Address 7
43 : Memory Address 8
42 : Memory Address 9
41 : Memory Address 10
40 : Memory Address 11
39 : Memory Address 12
38 : Memory Address 13
37 : Memory Address 14
58 : CRU CLK line
53 : RESET line
56 : CRU OUT line
57 : CRU IN line

8008 Connections

62 to 69 : Data o/p port 8.
14 to 21 : Data o/p port 9.
4 to 11 : Data 1/p port 3.

REFERENCES

1. F. Heart, S. Ornstein, W. Crawther and W. Barker. "A new minicomputer/multiprocessor for the ARPA network". Newnam Inc. Cambridge, Massachusetts, NCC Proceedings, June 1973.

2. K. Ohmori, N. Koike, K. Nezu and S. Susuki. "A multi-microprocessor system", COMPCON, September 1974.

3. L. Eaton and E. Page. "An interprocessor communication scheme for multiple-microprocessor systems", Clemson Univeristy, South Carolina.

4. J. R. Pierce. "How far can data loops go?", IEEE Transactions, Comm - 20, June 1972.

5. J. May and L. Krakawer. "The architecture of a multiple-microprocessor network processor", Codex Corporation, Newton, Massachusetts.

6. M. Moore, Wright and Patterson, "A distributed microprocessor system for avionics", AFB, Ohio.

7. G. Reyling. "Performance and control of multiple microprocessor systems", Computer Design, March 1974.

8. S.Fuller, J. Ousterhort, L. Raskin, P. Rubinfield and R. Swan.
"Multi-microprocessors: An overview and working example", IEEE Proceedings, Vol. 66, February 1978.

9. S. Crowther, S. Ornstein, M. Kraley, R. Bressler and F. Heart,
"Pluribus - A reliable multiprocessor", AFIDS Conference Proceedings, Vol. 4, 1975.

10. J. E. Wirshing.
"Computer of the 80's - Is it a network of microprocessors?", IEEE Proceedings, Compcon, 1975.

11. V. Ravindran and T. Thomas.
"Characterisation of multiple-microprocessor networks", Stanford University, California, IEEE Comp. Soc. Int'l Conference, 1973.

12. C. W. Wiatrowski and C. R. Teeple.
"Add flexibility to your control system with distributed data processing", Instruments and Control Systems, March 1976.

13. R. Nilsen.
"Distributed computer architectures", Hughes Aircraft Company, California.

14. T. Burton.
"Multi-microprocessor systems combine the efficiency", Electronic Design, August 1977.

15. A. Weissberger.
"Analysis of multiple-microprocessor system architecture", Computer Design, June, 1977.

16. A. J. Nichols.
"An overview of microprocessor applications", IEEE Proceedings, Vol. 64, June 1976.

17. V. May and G. Forney.
"Application of LSI microprocessors in data network hardware", Codex Corporation, Newton, Massachusetts.

18. M. Lewin.
"Integrated microprocessors", Transactions on Circuits and Systems, No. 7, July, 1975.

19. M. Johnson.
"Microprocessors in unconventional architectures", Honeywell Systems - Minneapolis, Minnesota.

20. P. Russo.
"Interprocessor communication for multi-microprocessor systems", Computer, April 1977.

21. A. Weissberger.
"Distributed function microprocessor architectures", Computer Design, November 1974.

22. B. Parasuraman.
"High performance microprocessor architectures", IEEE Proceedings, Vol. 64, No. 6, June 1976.

23. Intel Corporation, SBC 80/20 Hardware Reference Manual, 1976.

24. Widdows.
"The minerva multi-microprocessor", Stanford Digital Systems Lab., July 1975.

25. B. C. Searle and D. E. Freberg.
"Microprocessor application in multiple processor systems", Computer, October 1975.

26. D. McAuliffe and K. Hagstrom.
"Multi-processor application in communications switching", North Electric Company, Ohio.

27. Ford.
"Harware support for inter-process communication and processor sharing", COMPCON Proceedings, 1976.

28. H. Lorin.
"Parallelism in hardware and software; real and apparent concurrency", Prentice-Hall, 1972.

29. L. Anderson,
"The microcomputer as distributed intelligence", IEEE International Symposium on Circuits and Systems, April 1975.

30. D. Forney and J. V. May.
"8-bit microprocessors can control data networks", Electronics, Vol. 49, No. 13, June 1975.

31. A. Anden and A. Berenbourn.
"A multi-microprocessor computer system architecture", Operating systems Review, Vol. 9, No. 5..

32. J. Harrison.
"Micros-minis and multiprocessing", Instrumentation Technology, February 1978.

33. K. Hagstrom and B. Beizer.
"Communications processor system study", North Electric Company, Ohio.

34. P. Jessel.
"Localised Microcomputer-processor based networks", Massachusetts Institute of Technology, Cambridge, Massachusetts.

35. K. Kerorbian.
"Microprocessors and LSI in stored program controlled systems", Le Materiel Telephonique, France.

36. Baer.
"Multiprocessing systems", IEEE transactions on Computers, December 1976.

37. J. W. Bowra and H. C. Torng.
"The modelling and design of multiple function unit processors", IEEE Transactions on Computers, March 1976.

38. B. N. Jordan and M. Gonzalez.
"Operation and control of multiple microcomputer systems", Northmester University, Evanston, Illinois.

39. J. Bass.
"A peripheral-oriented microcomputer system", IEEE Proceedings, Vol. 64, June 1976.

40. C. Ogden.
"Fundamentals of microcomputer systems", Mini-Micro Systems, November 1977.

41. J. Nicoud.
"Peripheral interface standards for microprocessors", IEEE Proceedings , Vol. 64, No. 6, June 1976.

42. R. Pond.
"Let microprocessors communicate", Electronic Design, November 1977.

43. D. Larson.
"Microprocessor intertie and communication system", Signal, April 1977.

44. D. Waddington.
"Microprocessors", Wireless World, 1974.

45. M. Helsig, D. Schueffler and C. Rose.
"Microprocessor based communication and instrument control for distributed control systems", Systems Research Center - Case Western Reserve University, Cleveland.

46. B. Kirk.
'Interrupts -- the tender trap", New Electronics, 1976.

47. B. Cook.
"Give flexibility to memory systems", Electronic Design, September 1974.

48. H. Falk.
"Linking microprocessors to the real world", IEEE Spectrum, September 1974.

49. G. Fisher.
"Speed microprocessor responses", Electronic Design, November 1975.

50. M. Gerla and L. Kleinrock.
"On the topological design of distributed computer networks", IEEE Transactions on Communications, Vol. COM-25, No. 1, January 1977.

51. J. Wakevly.
"Microcomputer reliability improvement using triple-modular redundancy", IEEE Proceedings, Vol. 64, No. 6, June 1976.

52. P. Enslow.
"What is distributed data processing systems", Computer, Vol. 11, January 1978.

53. G. Reyling.
"Extend LSI-processor capabilities", Electronic Design, October 1974.

54. C. Torrero.
"Focus on microprocessors", Electronic Design, September 1974.

55. W. Farnback.
"Bring up your microprocessor bit-by-bit", Electronic Design, July 1976.

56. C. Bass and D. Brown.
"A perspective on microcomputer software", IEEE Proceedings, Vol. 64, No. 6, June 1976.

57. "Microcomputer may answer a need in your next design project", Article. Product Engineering, 1976.

58. "Designers need and are getting plenty of help - microprocessors", Article. Electronics, April 1976.

59. "Microprocessors - Designers gain new freedom on options multiple", Article. Electronics, April 1976.

60. Intel MC8-8 User's Manual.

61. Texas Instruments - Assembly Language Programmer's Guide.

62. A. Jones, R. Chansler, I. Duram and P. Feiler.
"Programming issues raised by a multiprocessor", IEEE Proceedings, February 1978.

63. D. Melvin.
"Microcomputer applications in Telephony", IEEE Proceedings, Vol. 66, No. 2, February 1978.

64. D. Stanzione.
"Microprocessors in Telecommunication Systems", IEEE Proceedings,Vol. 66, February 1978.

65. V. Klig.
"Biomedical applications of microprocessors", IEEE Proceedings, February 1978.