National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Ottawa, Canada
K1A 0N4

NL-339 (Rev. 8/80)

TASK-DRIVEN MULTI-MICROCOMPUTER SYSTEM

BY

E.T. FATHI, P. Eng.

A thesis submitted to the School of Graduate Studies and Research
in partial fulfillment for the degree of Master of Applied Science
Department of Electrical Engineering

Ottawa Canada

May 1981

© E.T. Fathi, Ottawa, Canada, 1981.

ABSTRACT

The concept of a Task-Driven multi-microcomputer system as an appropriate solution for a wide range of applications with diversified computational needs, is investigated. In general, the performance of a multiple processor system is governed by its architecture, the control scheme used to coordinate the activities of the various processors, and its interprocessor communication. In this thesis, each of these subjects is treated separately. After introducing multiple processor systems in general terms, the functional design of a multi-microcomputer system is presented. The main characteristics of a Task-Driven architecture are then highlighted. To provide for an efficient control scheme, the design procedures and implementation aspects of a modular two level executive, which is "tailor made" for a Task-Driven system, are presented. To provide interprocessor communication with a high degree of availability and reliability, the concept of segmented bus is introduced. Following that, the communication protocol and general implementation aspects associated with a centrally controlled segmented bus are described.

ACKNOWLEDGEMENTS

The author wishes to thank his thesis advisor, Dr. M. Krieger, whose guidance and support throughout the research period is greatly appreciated.

The author wishes to acknowledge the help of the Royal Canadian Mounted Police for granting him the opportunity to attend the graduate school at the University of Ottawa. Special thanks are due to Mr. G.P. Lutley, Officer in Charge of the Engineering Branch, whose constant support and encouragement made this possible.

Also acknowledged is the help of the following: Mr. D.D. Dawson for constructive technical review, Mr. S.H. Coughlin for drawing all the figures, Mrs. D.A. Kind and Mrs. D.R. Sharp for typing and editing.

Finally, the author wishes to thank his wife Arleen for supporting him during his studies.

TABLE OF CONTENTS

PAGE

CHAPTER 1 - INTRODUCTION

1.1 PROBLEM STATEMENT 1
1.2 THESIS OBJECTIVE. 4
1.3 THESIS STRUCTURE. 6

CHAPTER 2 - THE W3 OF MULTIPLE PROCESSOR SYSTEMS

2.1 OVERVIEW 9
2.2 INTRODUCTION 9
2.3 WHAT IS A MULTIPLE PROCESSOR SYSTEM? 12
2.3.1 TIGHTLY COUPLED MULTIPLE PROCESSOR SYSTEM CHARACTERISTICS. 13
2.3.2 LOOSELY COUPLED MULTIPLE PROCESSOR SYSTEM CHARACTERISTICS. 16
2.3.3 MULTI-MICROCOMPUTER SYSTEM CHARACTERISTICS 19
2.4 WHY A MULTIPLE PROCESSOR SYSTEM? 22
2.4.1 SYSTEM PROCESSING CAPABILITIES 22
2.4.2 SYSTEM RELIABILITY MEASURES. 26
2.4.3 SYSTEM DESIGN AND DEVELOPMENT MEASURES 32
2.5 WHEN TO USE/NOT USE A MULTI-MICROCOMPUTER SYSTEM?. 35
2.6 FUNCTIONAL DESIGN OF A MULTI-MICROCOMPUTER SYSTEM. 37
2.6.1 PHYSICAL STRUCTURE 38
2.6.2 LOGICAL STRUCTURE. 44
2.6.3 CONTROL AND COORDINATION ASPECTS OF A MULTI-MICROCOMPUTER SYSTEM. 48
2.7 A TASK-DRIVEN MULTI-MICROCOMPUTER SYSTEM 54
2.7.1 SALIENT CHARACTERISTICS OF A TASK-DRIVEN SYSTEM.... 54
2.8 CONCLUSIONS 57
2.9 REFERENCES 58

PAGE

CHAPTER 3 - EXECUTIVE FOR A TASK-DRIVEN MULTI-MICROCOMPUTER SYSTEM

3.1 OVERVIEW 62
3.2 INTRODUCTION 62
3.3 OUTLINE OF SYSTEM HARDWARE 64
3.4 ELEMENTS OF SYSTEM EXECUTIVE 66
3.4.1 DESIGN PHILOSOPHY. 68
3.4.2 EXECUTIVE HIERARCHICAL STRUCTURE 69
3.4.3 PROCESS LEVEL. 70
3.4.4 TASK LEVEL 71
3.4.5 EXECUTIVE COMMUNICATION PROTOCOL 73
3.5 TASK INTERRELATIONS. 76
3.5.1 TASK HIERARCHIES 76
3.5.2 TASK STATES AND STATE DIAGRAMS 77
3.5.3 THE INACTIVE STATE 79
3.5.4 THE READY STATE. 80
3.5.5 THE RUNNING STATE. 81
3.5.6 THE WAITING STATE. 82
3.5.7 THE SUSPENDED STATE. 83
3.5.8 SYSTEM STATE DIAGRAM 84
3.6 IMPLEMENTATION ASPECTS 86
3.6.1 TASK ALLOCATOR, GENERAL OUTLINE. 87
3.6.2 TASK ASSIGNMENT PROCESS, GENERALIZED FLOW DIAGRAM. 89
3.6.3 THE LISTS OF THE EXECUTIVE 91
3.6.4 STATIC LISTS 91
3.6.5 DYNAMIC LISTS. 92
3.6.6 TASK ASSIGNMENT PROCESS, DETAILED FLOW DIAGRAM . . 95
3.7 TASK PARTITIONING AND ALLOCATION 97
3.7.1 TASK PARTITIONING. 97
3.7.2 TASK ALLOCATION. 100
3.8 CONCLUSIONS. 101
3.9 REFERENCES 104

PAGE

CHAPTER 4 - SEGEMENTED BUS ARCHITECTURE FOR MULTI-MICROCOMPUTER SYSTEMS

4.1 OVERVIEW 107
4.2 INTRODUCTION 108
4.3 THE SEGMENTED BUS CONCEPT. 110
4.4 SEGMENTED BUS ARCHITECTURES. 114
4.4.1 DISTRIBUTED CONTROL SEGMENTED BUS ARCHITECTURE . . 114
4.4.2 CENTRALLY CONTROLLED SEGMENTED BUS ARCHITECTURE. . 119
4.5. CONCEPTUAL DESIGN OF THE BUS CONTROLLER 136
4.5.1 PARTITIONING OF THE BUS CONTROLLER MAIN PROCESSES INTO TASKS 136
4.5.2 TASK-PROCESSOR ALLOCATION,A ROUGH DRAFT. 151
4.6 CONCLUSIONS. 158
4.7 REFERENCES 161

CHAPTER 5 - SUMMARY AND RELATED RESEARCH TOPICS

5.1 THESIS SUMMARY. 162
5.2 RECOMMENDED RELATED RESEARCH TOPICS. 164

BIBLIOGRAPHY. 169

LIST OF FIGURES

PAGE

CHAPTER 2

FIGURE NUMBER

2.1 SIMPLIFIED BLOCK DIAGRAM OF A TIGHTLY COUPLED MULTIPLE PROCESSOR SYSTEM. 15
2.2 SIMPLIFIED BLOCK DIAGRAM OF A LOOSELY COUPLED MULTIPLE PROCESSOR SYSTEM. 18
2.3 SEPARATE APPLICATION AND COMMUNICATION PROCESSORS . 21
2.4 THE PERFORMANCE OF A SYSTEM AS A FUNCTION OF COST. 23
2.5 THE SATURATION EFFECT. 24
2.6 INTERPROCESSOR COMMUNICATION VIA A COMMON MEMORY STRUCTURE. 40
2.7 INTERPROCESSOR COMMUNICATION VIA SYSTEM BUS STRUCTURE. 41
2.8 INTERCONNECTION TOPOLOGIES 43
2.9 SINGLE MASTER-MULTIPLE SLAVE ORGANIZATION. 46
2.10 FULLY CONNECTED MASTER-MASTER ORGANIZATION 47
2.11 PICTORIAL REPRESENTATION OF THE ARBITRATION PROCEDURE. 49
2.12 SIMPLIFIED BLOCK DIAGRAM OF A TASK-DRIVEN MULTI-MICROCOMPUTER SYSTEM 55

PAGE

CHAPTER 3

FIGURE NUMBER

3.1 SIMPLIFIED BLOCK DIAGRAM OF TASK-DRIVEN ARCHITECTURE 64
3.2 TYPICAL PROCESSOR ORGANIZATION AND A PROFILE OF ITS PROGRAM MEMORY 65
3.3 GRAPHICAL REPRESENTATION OF A SYSTEM IN TERMS OF PROCESSES AND TASKS. 66
3.4 GRAPHICAL REPRESENTATION OF PROCESS/TASK SEQUENCING. 67
3.5 GRAPHICAL REPRESENTATION OF TWO LEVEL EXECUTIVE MODEL. 70
3.6 COMMUNICATION PROTOCOL USED DURING TASK CALL AND EXECUTION. 75
3.7 TASK CLASSIFICATION TREE 76
3.8 THE IDEAL EXECUTION CYCLE. 78
3.9 STATE TRANSITIONS ASSOCIATED WITH THE INACTIVE STATE. 79
3.10 STATE TRANSITIONS ASSOCIATED WITH THE READY STATE. 80
3.11 STATE TRANSITIONS ASSOCIATED WITH THE RUNNING STATE. 81
3.12 STATE TRANSITIONS ASSOCIATED WITH THE WAITING STATE. 82
3.13 STATE TRANSITIONS ASSOCIATED WITH THE SUSPENDED STATE. 83
3.14 COMPLETE SYSTEM STATE TRANSITION DIAGRAM 85
3.15 GENERAL FLOW DIAGRAM OF THE TASK ASSIGNMENT PROCEDURE. 90
3.16 DETAILED FLOW DIAGRAM OF THE TASK ASSIGNMENT PROCEDURE. 96

PAGE

CHAPTER 4

FIGURE NUMBER

4.1 OPEN ENDED SEGMENTED BUS 113
4.2 CLOSED LOOP SEGMENTED BUS. 113
4.3 CENTRALLY CONTROLLED SEGMENTED BUS 121
4.4 HARDWIRED PRIORITY ENCODER AND INTERFACE HANDSHAKE CONTROL. 130
4.5 CONTROL INTERFACE BLOCK DIAGRAM. 132
4.6 BUS CONTROLLER PROCESS SEQUENCING RELATIONS. . . . 137
4.7 CONTROL FLOW DIAGRAM OF THE COMMUNICATION PROCESS. 139
4.8 CONTROL FLOW DIAGRAM OF THE ARBITRATION PROCESS . 143
4.9 CONTROL FLOW DIAGRAM OF THE DIAGNOSTICS PROCESS IN TERMS OF TASKS 148
4.10 CONTROL FLOW DIAGRAM OF THE DIAGNOSTICS PROCESS IN TERMS OF TASKS 149
4.11 CONTROL FLOW FIAGRAM OF THE DIAGNOSTICS PROCESS IN TERMS OF TASKS 150
4.12 TASK-PROCESSOR ALLOCATION FOR THE BUS CONTROLLER. . 158

LIST OF TABLES

TABLE NUMBER		PAGE
4.1	BIDIRECTIONAL DATA SWITCH STATUS TRUTH TABLE . . .	126
4.2	MEMORY ACCESS SELECTION AND ELEMENT INTERRUPT TRUTH TABLE.	135
4.3	ESTIMATED RELATIVE TASK EXECUTION TIMES.	153
4.4	READ CONTROL INFORMATION TASK CODE	155

1.0 INTRODUCTION

This chapter introduces the main problems that led to the investigation presented in this thesis. Next, thesis objectives are briefly outlined and finally, a short summary of the major chapters of the thesis is given.

1.1 PROBLEM STATEMENT

The advances in microprocessor technology created the proper environment for introducing digital processing in many new and diversified areas. As the number of applications with more elaborate computational demands increases, one needs to provide more processing power. This can be achieved at two levels:

(a) At the processor level - To rely on technological improvements to push the microprocessor beyond its current maximum capabilities.

(b) At the system level - To extend the capabilities of a single microprocessor by exploiting the concept of concurrent execution.

Microprocessor technological improvements are made exclusively by the manufacturers, and even though some user feedback may influence future development, they are out of control of the average user. With the existing

technology, there are definite limits to the extent and type of improvements possible. Unless new technology is developed, all the technological improvements can be regarded as evolutionary rather than revolutionary, and as such, concentrate on the following areas:

(i) Longer word length.

(ii) A more extensive and powerful instruction set.

(iii) Addition of memory and various input/output (I/O) capabilities on the chip.

(iv) Higher speed of operation and reduced power consumption.

(v) Introduction of additional control lines to support multiple microprocessor architecture.

(vi) Addition of various software support on the chip.

The last two points indicate a definite trend by the industry to move towards multiple microprocessor systems by providing the hardware and software support to facilitate their design.

Improvements at the system level can be obtained by exploiting the concept of concurrent execution thus enhancing system performance.

Concurrency of execution can be obtained in two ways:

(i) Using a single microprocessor with a multitasking real-time executive.

(ii) Using a multiple microprocessor system controlled by a real-time executive.

The first approach is only useful in small special purpose systems as it tries to obtain maximum resource utilization from a single microprocessor. Due to the physical limitations of the microprocessor, this method is more appropriate for minicomputers or mainframe processors.

The second method, in general, does not attempt to maximize individual microprocessor utilization but rather complete system utilization. By adding more microprocessors, it can be regarded as being independent of the microprocessor's physical characteristics. Thus, theoretically, this method can be considered as having unlimited room for expansion and improvement. In practice, this method has its own limitations which are indicated in terms of the following three problem areas:

(i) How to define the most suitable hardware architecture for utilization in a multiple microprocessor system.

(ii) How to define an executive for controlling and coordinating the various microprocessors activities.

(iii) How to define a communication facility which connects all the microprocessors in the system such that the bottleneck condition associated with the transfer of data and control information is reduced.

1.2 THESIS OBJECTIVE

The objective of this thesis is to find solutions to the problems of hardware architecture, executive, and interprocessor communication as they arise in multiple microprocessor systems. It should be mentioned that even though this thesis considers these points from a general point of view, all are application dependent. To achieve application independence the suggested solutions are made flexible, so that later, they can be fine tuned to a particular application.

In terms of system hardware architecture, the most important aspect is to extract the best characteristics from the general multiple processor organizations and then to combine them to form a new architecture. This architecture must be well-suited for microprocessor implementation and as such must exploit the positive

feature of microprocessors being low cost sources of computing power. Also, it must counteract the negative aspects of microprocessors, namely the limited I/O flexibility and the memory-processor bottleneck.

In terms of system executive, a flexible control scheme for multiple microprocessor systems that does not have high overhead, must be developed. This can be done if system operation can be subdivided into independent tasks, and if maximum hardware utilization is not paramount. The design must provide the mechanisms to ease the system operation partitioning problem and the allocation of its tasks to the various processing elements. Also, a systematic way of coordination and synchronization of the various activities in the system must be developed. Finally, the system control must be developed in such a way that it offers system transparency to the user. Thus, when a new job has to be executed, the user need not concern himself with the actual hardware details of the system.

The problem of the communication facility is solved along three general guidelines. First, the interconnection scheme must be both flexible and efficient in order to support a wide spectrum of applications. The applications may range from a dedicated special purpose system for which the communication needs are well defined and fixed, to a general purpose computing system for which the communication needs are relatively unknown and vary with the jobs being executed. Secondly, it must be able to support a wide variety of hardware architecures. Third,

it must provide an efficient means of transferring both data and control information.

1.3 THESIS STRUCTURE

The material presented in this thesis is divided into five chapters. After the introduction, each of the above stated problems is treated separately in a dedicated chapter, each including introduction and conclusions. The final chapter highlights the most significant results obtained in the three chapters and suggests related research topics. The major ideas presented in each of the three main chapters are outlined below.

CHAPTER 2

"The W3 of Multiple Processor Systems" deals with the solution to the problem of defining the most suitable hardware architecture for multiple microprocessor systems. The presentation is done using the "onion peel" method. In the outer most layer, the discussion attempts to shed light on the concept of multiple processor systems and to clarify some of the terminology. The questions 'What','Why' and 'When' in regard to such systems are considered both in general, for all processors regardless of their size, and specifically for microprocessors. In the middle layer, a functional design of a multi-microcomputer system is presented in terms of both physical and logical structures. Also, the various aspects associated with the control and coordination of system resources are

discussed. In the innermost layer, a specific multi-microcomputer architecture, known as a Task-Driven system, is described.

CHAPTER 3

"Executive for a Task-Driven Multi-Microcomputer System", is concerned with the problem of defining the design procedures and implementation aspects of an executive for the Task-Driven architecture described in Chapter 2. The design philosophy of this executive follows the guidelines presented in the thesis objective section (i.e. designing for maximum system utilization and flexibility rather than maximizing individual resource utilization). The resulting executive consists of a modular two level hierarchical structure which is easy to implement. Also presented in this chapter are some of the relevant ideas concerning task partitioning and allocation in multiple processor systems.

CHAPTER 4

"Segmented Bus Architecture For Multi-Microcomputer Systems" deals with the problem of defining a general interconnection scheme for multi-microcomputer systems. The scheme is designed to facilitate interprocessor communication activities in terms of providing a high degree of availability and reliability. The proposed interconnection scheme is based on the segmented bus concept which can accommodate simultaneous mutually exclusive transfers on a data bus. The basic characteristics of the segmented bus, as well as the

architectures associated with its control are described. Here, distributed and centrally controlled segmented bus architectures are presented. This chapter concentrates mainly on the centrally controlled segmented bus and describes in some detail the various functions associated with the central control unit and its conceptual design.

2.0 THE W3 OF MULTIPLE PROCESSOR SYSTEMS

2.1. OVERVIEW

The questions WHAT, WHY and WHEN in regard to multiple processor systems are considered in detail, hence the term "The W3 of Multiple Processor Systems". The concept of Multiple Processor Systems is presented as an appropriate solution to increasing demands for additional computing power necessary to meet new requirements and/or more complex applications. The presentation is made both in general and in specific terms. Those aspects which are applicable to all processors regardless of their size are presented in general. However, as the main interest here lies with multi-microcomputer systems, the aspects which are dependent on processor power and I/O flexibility are considered specifically with respect to microprocessors.

2.2 INTRODUCTION

The need for increased computing power arises in many systems in order to meet new requirements and/or more complex applications. There are a number of ways in which one can increase the processing power of a computer system. The first, most obvious way is to use a more powerful Central Processing Unit (CPU), however, beyond a certain point, this approach has limitations in terms of cost, complexity and reliability. The second alternative is to include special purpose sub-systems such as an I/O driver or a fast arithmetic unit. This solution is most appropriate for special purpose systems,as the system enhancement is restricted to

specific areas. The third and most interesting option from the point of view of a general solution is to use multiple processor systems.

In a system with a single processor, execution of the various tasks is sequential, as the processor can execute only one instruction at any given time. However, by time sharing the CPU, one can obtain apparent concurrency. For true concurrency, that is the execution of a number of cooperating and/or independent tasks simultaneously, one requires a multiple processor system. Here, systems which contain more than one processing sub-system with some sort of on-line interconnection will be considered.

Although at times multiple processor systems are discussed in general, the main interest lies with their applicability to microprocessor based systems. The idea of using more than one processing element in a system in order to achieve better performance preceded the development of microprocessor technology. This technology "opened the door" for the utilization of computing power in a wide range of applications which were out of reach otherwise, due to the cost and physical size of computers. The use of multiple microprocessor systems extended the range and capabilities of a single microprocessor to more complex areas which were previously in the domain of large computers. As will be discussed later, they also led to other system enhancements like reliability and ease of design.

In order to shed light on the concept of multiple processor systems and to clarify the terminology associated with them, the presentation is both in general and specific terms. Aspects which are applicable to all processors regardless of their size are presented in general terms. However, as the main interest here lies with microprocessor based systems, aspects which are dependent on processor power and I/O flexibility are discussed specifically with respect to microprocessors.

This chapter consists of three major parts. In the first part, the questions WHAT, WHY and WHEN in regard to multiple processor systems are discussed in detail. The question "What are the prominent multiple processor architectures?", is considered in Section 2.3 by highlighting ~~the~~ salient features of each organization. The question Why, i.e. the motivations behind developing these systems, is discussed in Section 2.4. The discussion is mainly from a general point of view, as the aspects associated with this question are usually applicable to all the various multiple processor organizations. Finally, the question "When to use a multiple processor system?", is discussed in Section 2.5 specifically in relation to microprocessor based systems, due to the strong interrelation between these systems and processor capabilities.

The second part consisting of Section 2.6, describes the functional design of a multi-microcomputer system in terms of physical and logical structures, and the various aspects associated with the control and

coordination of system resources. In the last part, consisting of Section 2.7, the general characteristics of a specific multi-microcomputer architecture, known as a Task-Driven multi-microcomputer system are presented.

It should be noted that the terminology used throughout this chapter follows what appears consistently in the literature. However, when experiencing large variations, the term felt to be the most appropriate was chosen.

2.3 WHAT IS A MULTIPLE PROCESSOR SYSTEM?

In 1966 Flynn [FLYN 66] classified computer systems into four categories based on the instruction stream and data stream. The resulting four classifications are:

(a) SISD - Single Instruction Single Data

(b) SIMD - Single Instruction Multiple Data

(c) MISD - Multiple Instruction Single Data

(d) MIMD - Multiple Instruction Multiple Data

Although Flynn's classification has been accepted as the most basic one, it is much too restrictive, as it only considers execution at the instruction level. In order to include newer organizations created by the developing technology, a number of modifications and extensions

to Flynn's taxonomy are suggested in the literature [HIGB 73], [PREN 79], [BOWE 80]. In the most general form, a multiple processor system capable of executing concurrently a number of tasks each utilizing different sets of data, can be considered as being a MIMD system. There are a number of multiple processor organizations such as pipeline, array, and vector which are not necessarily MIMD systems [BAER 76]. These systems will not be considered here, since they act effectively as a single fast processor adapted to specific applications.

Multiple processor systems can be differentiated by the degree of coupling and the nature of interaction that exist between the various processors. The two extremes are the tightly and loosely coupled systems. In general, tightly coupled systems share common primary data memory which can be used as a communication channel between the processors. On the other hand, loosely coupled systems do not share a common memory and therefore must use an explicit communication interface between processors.

2.3.1 TIGHTLY COUPLED MULTIPLE PROCESSOR SYSTEM CHARACTERISTICS

Tightly coupled multiple processor system, also known as multiprocessor system [BOWE 80] has the following characteristics:

(a) COMMON MEMORY - A primary memory that can be accessed by all processors in the system. In addition each processor may have a separate data memory.

(b) COMMON OPERATING SYSTEM - A single common operating system controls and coordinates all the interaction between processors and processes.

(c) SHARED RESOURCES - I/O facilities and other system resources are generally shared among the processors. However, some resources may be dedicated to specific processors.

(d) EQUAL PROCESSING POWER - Generally, a symmetrical configuration with the general purpose processors exhibiting similar capabilities.

(e) DYNAMIC LOAD SHARING - Permits uniform load sharing across all processors by dynamically distributing the load of an overloaded processor.

(f) PROCESSOR AUTONOMY - Each of the cooperating processors can execute significant computations individually.

(g) SYNCHRONIZATION - There is a need for synchronization between cooperating processors.

A simplified Block Diagram of a tightly coupled multiple processor system is shown in Figure 2.1.

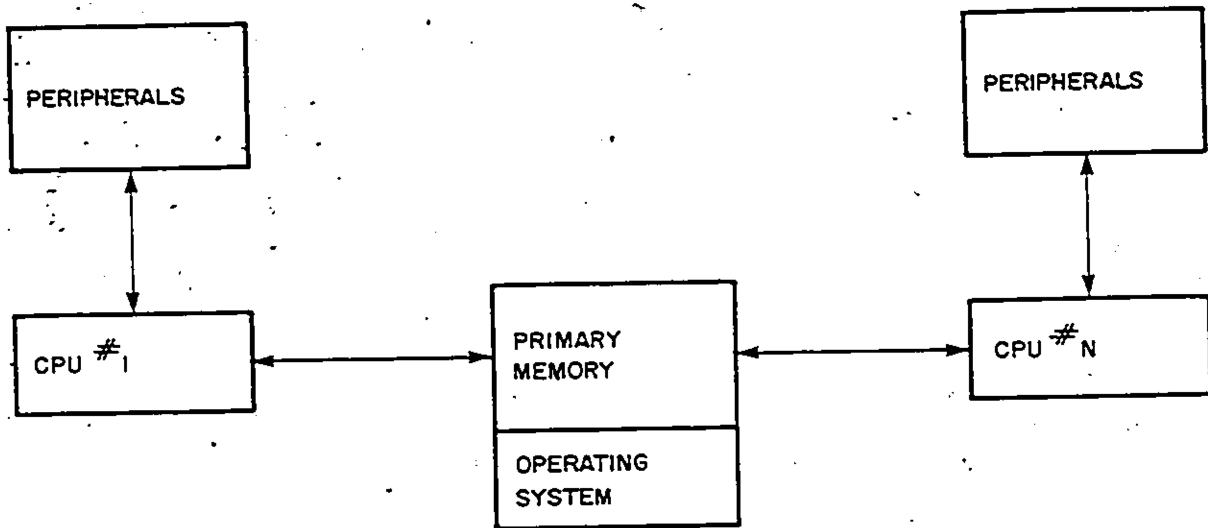


FIGURE 2.1 – SIMPLIFIED BLOCK DIAGRAM
OF
TIGHTLY COUPLED MULTIPLE PROCESSOR SYSTEM

The major limitation to the application of a tightly coupled multiple processor organization is the possibility of primary memory access conflicts. This restriction tends to put an upper bound on the number of processors which can effectively be supported by a single operating system. Most processor-memory switching structures attempt to reduce the amount of main memory access conflicts. The three most fundamental processor-memory organizations are: the common bus, the crossbar switch, and the multiport memory. Further details can be found in reference [ENSL 77].

Basically, multiprocessor organization can be considered as a collection of low speed processors performing the work of a single high speed processor. In general, the applicability of this multiple processor organization to microprocessor based systems is limited, due to the processor-memory bottleneck which characterizes microprocessors.

Multiprocessor organizations have been used in the implementation of various computer systems [SATY 80]. Example of a multiprocessor system which utilizes microprocessors in its implementation is the Stanford University Minerva System [WIDD 76].

2.3.2 LOOSELY COUPLED MULTIPLE PROCESSOR SYSTEM CHARACTERISTICS

Loosely coupled multiple processor organization also known as computer network [FULL 78] has the following characteristics:

(a) AUTONOMOUS COMPUTERS - The system contains a number of independent computer systems that can be located at geographically dispersed areas.

(b) COMMUNICATION INTERFACE - The various computers in the system are interconnected via a communication interface.

(c) COMMUNICATION PROTOCOL - Intercomputer communication must follow a rigid communication protocol.

(d) SERIAL COMMUNICATION - The intercomputer communication links are generally high speed serial lines.

(e) COMPUTER ACCESSIBILITY -A user at any site can use the computing facilities at all other sites.

(f) ON SITE COMPUTATION - In general, the network is only used for communication with actual computing being done at a single site.

A simplified Block Diagram of a loosely coupled multiple processor system is shown in Figure 2.2.

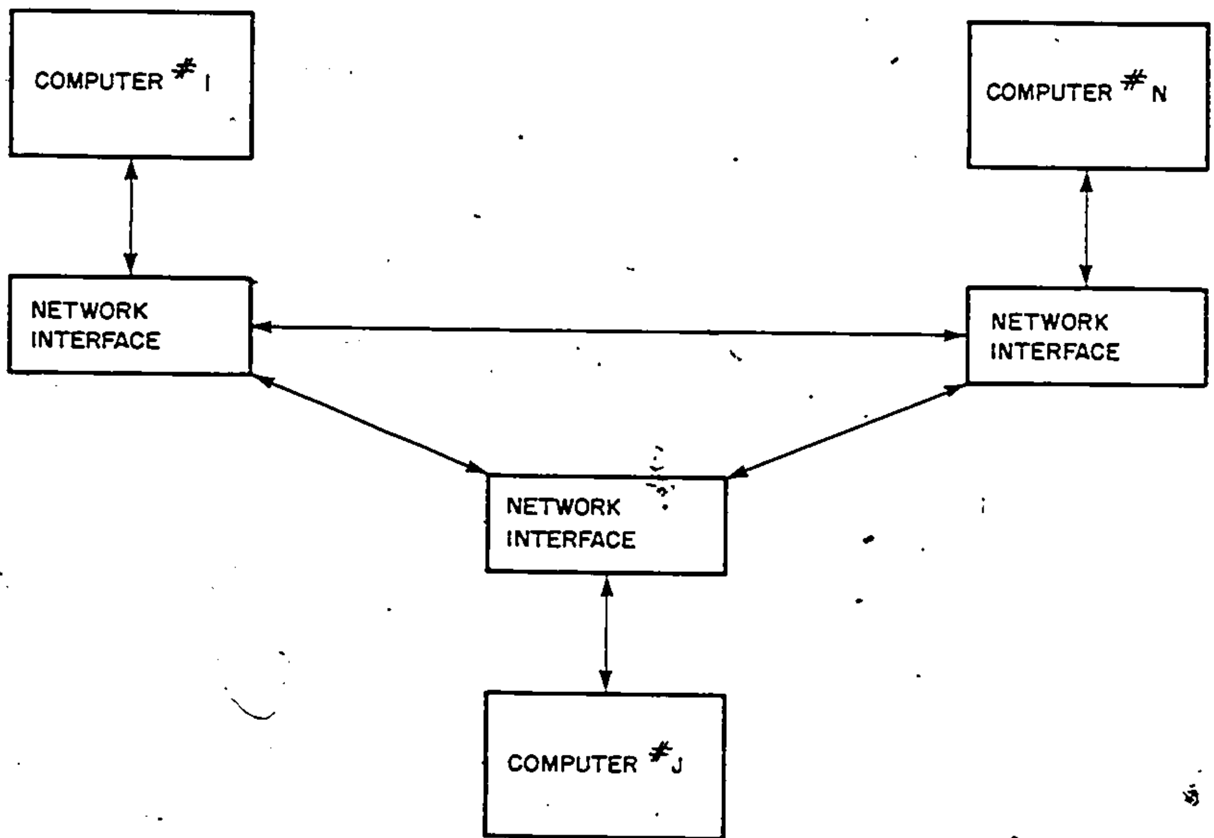


FIGURE 2.2 — SIMPLIFIED BLOCK DIAGRAM OF A LOOSELY COUPLED MULTIPLE PROCESSOR SYSTEM

The best known and largest computer network is the Advanced Research Projects Agency (ARPA) Network, which connects over 50 major computing facilities across the United States [SCHW 77].

Computer network organization, as it stands, with its rigid interprocessor communication requirements is not very applicable to microprocessor based systems. However, with the increasing number of applications which require distributed computing power, it is likely that some modified versions of computer network organization may be used. For example, the communication protocol and other control aspects associated with a local loop connecting the various computing facilities of a university may be implemented using microprocessor based systems.

2.3.3 MULTI-MICROCOMPUTER SYSTEM CHARACTERISTICS

The tightly coupled and loosely coupled structures are the two extremes of multiple processor organizations. There are various other organizations which do not qualify as either one and generally can be considered as moderately coupled multiple processor organizations. This organization is the most suitable for microprocessor based systems, as it combines the better qualities of the tightly coupled and loosely coupled systems. Microprocessor based systems designed to have the major characteristics listed below are known as multi-microcomputer systems [RUSS 77] and are often called distributed intelligence microcomputer systems (DIMS), [ANDE 75].

In a multi-microcomputer system one first has to partition the general workload into relatively independent tasks which can then be assigned to the various elements in the system. Such a system has the following main characteristics:

(a) AUTONOMOUS ELEMENTS -Each individual element generally consists of a CPU, local program and data memory, plus any additional peripherals it may use or control.

(b) PROCESSORS DEDICATED TO A TASK -Ideally, each element is dedicated to perform a specific task which determines its relative complexity.

(c) PROCESSORS WITH VARIED COMPLEXITIES - System configuration is not necessarily symmetrical due to the various complexities of the elements.

(d) SOFTWARE AND HARDWARE OPTIMIZATION - The hardware and software of each element is "tailor made" to fit the specific task that it has to perform.

(e) DATA LEVEL COMMUNICATION - The interprocessor communication is generally at the data level. However, in certain situations data may contain commands or include responses to specific requests.

(f) SEPARATE APPLICATION AND COMMUNICATION PROCESSORS - In general, each element handles both I/O control and system communication. In the case of heavy communication activity, one of these functions may be delegated to another processor [SPET 77], as shown in Figure 2.3. For example, the various I/O

functions such as device interface and software driver could be carried out by a special dedicated I/O processor.

(g) STATIC LOAD SHARING - Because of the dedicated processors, a minimal system cannot support dynamic load sharing thus proper load balance must be done during the design phase. However, to introduce some load sharing one could include additional units.

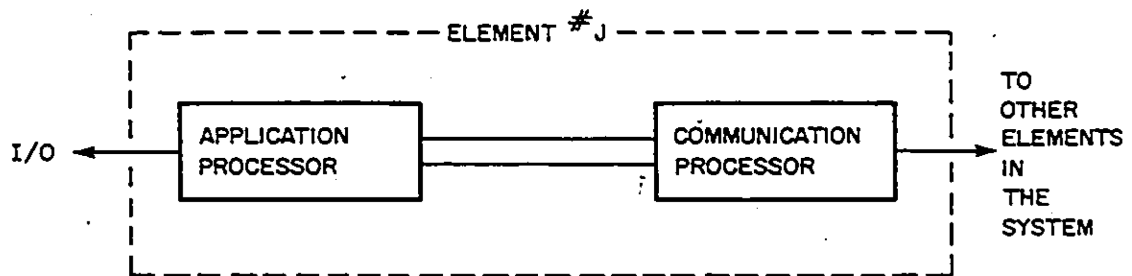


FIGURE 2.3 - SEPARATE APPLICATION AND COMMUNICATION PROCESSORS

A more detailed description of the physical and logical structures as well as the control aspects associated with multi-microcomputer systems will be presented in Section 2.6.

2.4 WHY A MULTIPLE PROCESSOR SYSTEM?

In order to properly evaluate multiple processor systems, one must postulate a number of system performance measures related to: Processing Capabilities, Reliability Issues, and Design and Development Aspects. It should be noted that certain properties of these measures are strongly interrelated and although they are included in one specific category they may influence all others.

2.4.1 SYSTEM PROCESSING CAPABILITIES

Under measures related to the processing capabilities of a system, cost-performance relationship, throughput, and resource sharing will be considered.

(a) COST-PERFORMANCE RELATIONSHIP

In general, the performance of a processor on a system level increases with cost. The question is How? In the case of a uniprocessor system, the performance of the processor increases more rapidly than cost. Grosch's Law [BAUM 75] suggested that the performance of a processor on a system level is proportional to the square of its cost, as shown in Figure 2.4. Thus the use of more than one processor strictly for the purpose of obtaining more raw processing power, was not very economical. However, with the rapid advancement in microprocessor technology, we are approaching the era of "No Cost" processing power relative to the cost of the other system parts. In line with this,

a cost/performance curve can be developed for which the incremental cost/performance ratio is shown to be more linear, as shown in Figure 2.4.

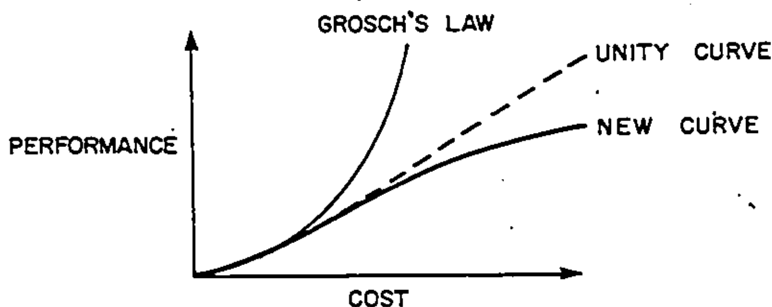


FIGURE 2.4 – THE PERFORMANCE OF A SYSTEM AS A FUNCTION OF COST

Using the curve in Figure 2.4, one can see that it becomes economically attractive to consider using additional processors in order to increase the system performance. Theoretically, optimum performance of a multiple processor system is at best equal to the sum of the optimum performance of each processor. In practice, it will be less.

In order to complete the discussion, one should consider the other costs associated with multiple processor systems. Ignoring software costs, it should be noted that many hardware elements like boards and connectors do not decrease in price as rapidily as microprocessors and memories. Thus, the reduction in the incremental cost/performance ratio achieved due to the decreasing cost of microprocessors and memories may be offset by the other system costs.

(b) THROUGHPUT

An appropriate indicator of system performance is system throughput. It is defined as the reciprocal of the time required to execute a given set of algorithms and is measured in terms of number of operations per unit time.

In the ideal situation, for a multiple processor system, the system throughput should increase at a rate proportional to the number of new processors added to the system. In practice, due to the saturation effect, the relationship between throughput and the number of processors in the system resembles the curve shown in Figure 2.5

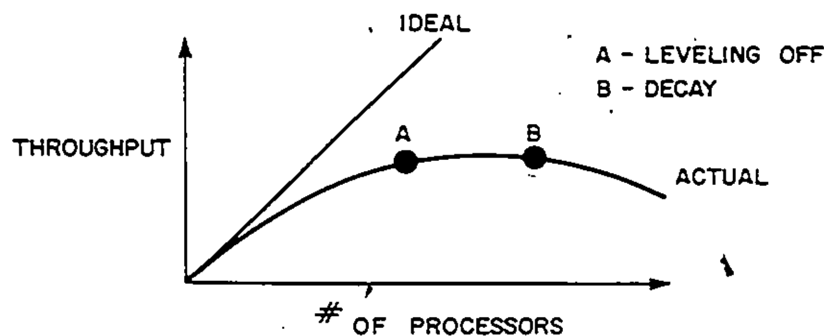


FIGURE 2.5 – THE SATURATION EFFECT

The saturation effect [JENN 77] is defined as the degradation in throughput for incremental increase in the number of computing elements. As can be seen from Figure 2.5, the system throughput follows the ideal linear curve when the number of processors in the system is small. As more processors are added to the system, the system throughput levels off and finally starts to decrease. This effect can be attributed to the

fact that as the number of processors increases so does the amount of contention on the shared resources. There is also an increase in the amount of overhead information which is required for proper interprocessor communication.

The points of levelling off and of decay may be different for various multiple processor organizations. Ideally, one would like to operate on the linear portion of the curve, which can be extended if the amount of contention and interprocessor communication activity are reduced. One way of achieving this is by properly partitioning the main job into smaller independent tasks which have relatively little intercommunication needs.

(c) RESOURCE SHARING

Resource sharing is a general characteristic of most multiple processor systems and is mainly influenced by financial considerations. In certain situations, it may be expensive to provide a dedicated resource to individual processing elements, if needed for only limited time periods. Thus, when the utilization factor of the resource is low, it makes sense to time share the resource among the processing elements in the system. However, time sharing of resources must be done in an orderly manner so as not to overload the resources and maintain conflict free systems.

The types of resources that may be shared in a multiple processor system range from a dumb printer or memory module to a sophisticated intelligent high speed math processor.

2.4.2 SYSTEM RELIABILITY MEASURES

Prior to discussing the different reliability related measures, the various aspects of reliability will be considered in general. The classical definition of reliability is the conditional probability that a system has survived the interval (0,dt), given that it was operational at time t=0. A measure of reliability is mean time before failure (MTBF).

In the case of multiple processor systems, the execution of a given job may require the cooperation of several processors, some of which may operate under different external conditions. For this case, the classical definition of reliability is found to be too general. A more appropriate definition of reliability for multiple processor systems is: the probability of executing a given task under a given condition for a specified time [DAVI 77].

The need for reliable systems arises when repair cannot take place due to one or more of the following constraints:

(a) Physical - e.g. a satellite system.

(b) Time - e.g. a life support system which cannot be switched off.

(c) Financial - e.g. when repair costs are equal to or exceed system costs.

For analysis purposes, most systems can be classified under one of two general categories: Non-redundant systems and Redundant systems [SIEW 78]. In non-redundant systems, each component must function properly in order for the system to work. In redundant systems, one can duplicate parts or all of the system to ensure at least limited operation in the event of a failure. To obtain the reliability model of redundant systems, one must first determine the reliability of each system module. Based on that, one can define a fault-tolerant system which describes the system reliability.

Due to the inherent redundancy in most multiple processor organizations, one can classify them as redundant systems. In addition, higher reliability can be achieved not only via duplication of physical elements, but also by duplicating the various tasks in more than one processor. The following reliability measures will be described in detail: Fault Tolerance, Flexibility, Serviceability, and Availability.

(a) FAULT TOLERANCE

A fault tolerant system is capable of overcoming hardware malfunction and/or software errors without human intervention, thus extending overall system MTBF beyond that of individual elements. To achieve this the system may employ massive or selective redundancy [WEIT 80].

Systems with massive redundancy utilize identical units operating simultaneously to protect against failures and/or errors. The effects of the faulty unit are logically masked out by the remaining properly operating units.

Systems with selective redundancy employ real-time recovery procedures in order to automatically switch a standby unit to replace a faulty one. In order to successfully initiate the recovery procedures, the system must first execute the following: Fault Detection, Fault Containment, and Fault Diagnosis [RENN 80].

Hardware faults and software errors must be detected as soon as possible after their occurrence. During the detection latency period (the time between fault occurrence and fault detection) the fault containment unit must prevent fault damaged data from propagating through the system and contaminating it. After detection, diagnostic programs can be used to determine the extent of the failure and to localize the faulty part. Subsequently, it is logically isolated from the system. Following the isolation phase, the recovery procedures are initiated so that the overall system operation can be restored while maintaining data integrity.

In order to overcome the high cost associated with massive redundancy, one can exploit the inherent redundancy of multiple processor systems to obtain a fault tolerant system. By trading off some of the computing power available for continuous

operation, a fail soft or graceful degradation operation can be achieved. Fault tolerance is obtained by transferring the job responsibilities of the faulty element to other elements in the system. If the system is designed to include a limited number of standby elements, then the functions of the faulty element will be assigned to one of them. Otherwise, the remaining, properly functioning elements will be asked to accept additional assignments, if possible.

If the remaining operational parts of the system are capable of handling the additional load, yet maintain system operation at full capacity, the system can be considered to be fail safe. If on the other hand, only a limited number or none of the faulty element responsibilities can be carried out by other system elements, yet the system can maintain operation at reduced capacity, it is considered to be in fail soft mode. It should be noted that if only limited numbers of assignments can be passed on to other elements, a preference should be given to those assignments which are most vital to the system.In order to provide an uninterrupted operation at full or reduced capacity the system must have the flexibility to bypass the faulty unit and pass its workload to other elements in real-time and under program control.

(b) FLEXIBILITY

Flexibility is a measure of how easy it is to reconfigure the system topology and reallocate the job responsibilities of an element among the other

system elements. This characteristic is necessary to facilitate system recovery procedures in the case of element failure.

In order to give the system the fail safe or fail soft capability, a real-time flexibility is needed (that is, system reconfiguration must be done in real-time and under program control). In multiple processor systems, the flexibility is mainly achieved in terms of software, although the hardware must be able to support system flexibility. However, the introduction of flexibility into the design, generally implies the use of more complex hardware and software which in turn, could reduce the individual element reliability, if not done properly.

Finally, it should be mentioned that the flexibility characteristic of a system may facilitate any future system expansion. However, it is closely related to the degree of system modularity. A highly modular system is generally capable of modular expansion in terms of hardware and it is the software flexibility that permits effective system expansion.

(c) SERVICEABILITY

Serviceability contains both maintainability and repairability aspects. The former is concerned with preventive maintenance by continuously running diagnostics routines and by periodic maintenance checks. The latter can be used to describe the ease of detecting and locating hardware failures and/or software errors once the

system is down. A measure of maintainability is mean-time to repair (MTTR) which is defined as the sum of the expected mean-time for periodic maintenance and the expected time for repair after failure [POPO 79].

It follows that since serviceability is concerned with repairability it complements the fault tolerance property. Both fault tolerance and serviceability attempt to guard against total system failure. The former, only while the system is up and the latter while the system is either up or the system is partially or completely down. Fault tolerance is introduced into the system during the design phase and its function is of a logical nature as to mask out the effects of a faulty unit. On the other hand, the serviceability function is mainly of a physical nature as it is concerned with the physical repair of a faulty part and attempts to prevent the occurrence of any future failures.

Multiple processor systems generally facilitate serviceabilty, as the total system complexity is broken down into simpler sub-systems. The reduction in sub-system's complexity implies that testing and repair are much easier and occasionally may even be partially carried out while the system is operating.In some multiple processor systems, the various processing elements hardware may be very similar or even identical, thus eliminating the need for large spare inventories. In the case of a failure, the faulty unit can be replaced by a spare, thus maintaining a low MTTR figure.

(d) AVAILABILITY

Availability can be used as a figure of merit to describe system availability to users, that is, the probability that the system is operational at time t. Availability can be expressed as the percentage of time the system is up (available) and is given by:

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

In multiple processor systems, availability is improved due to higher MTBF figures achieved by higher reliability, and due to lower MTTR figures obtained by improved serviceability.

2.4.3 SYSTEM DESIGN AND DEVELOPMENT MEASURES

In complex applications, it is desirable to partition the main job into simpler units (tasks). The primary goal of the partitioning process is to minimize the amount of inter-dependency between tasks. Ideally one would like to assign each task to a dedicated processor and in this way inter processor communication is limited to the data level. A more elaborate discussion on task partitioning will be presented in Chapter 3. A properly partitioned system, when implemented using multiple processor system, will lead to performance improvements in terms of: system deployment, modularity, prolonged life cycle as well as facilitating the human engineering aspect.

(a) SYSTEM DEPLOYMENT

One basic difference between a multiple processor system and a single processor system is the former's potential of being useful, even though it is only partially implemented. Furthermore, the time required for a system to become operational is faster in a multiple processor system. This is due to the fact that the development, implementation, and installation phases can overlap. Once a functionally independent sub-system is developed it can be implemented, installed, and used. As more sub-systems are implemented they may be added to the existing section of the system until the complete system is implemented and operating at full capacity.

(b) MODULARITY

System modularity can be defined in terms of the compactness and isolation of all the different elements in the system. Modular system design will generally shorten development and debug time as well as facilitate serviceability due to the fact that independent hardware and software modules with reduced complexity are designed and implemented.

The various units in the system are also more responsive to their dedicated tasks. The software and hardware of each sub-system can be optimized to the given task for which it is responsible.

This allows for a faster and more efficient response time. Consider a sub-system which may be required to control a process with rapid responses to interrupts. If the response time of one processor is not sufficient, more processors may be added in order to optimize the response time.

System modularity may improve system versatility. The association of software functions with specific hardware modules, if done properly, may result in the ability to control the software configuration within the different end-user systems, thus making the system more versatile.

(c) PROLONGED LIFE CYCLE

In terms of future development, system modularity also facilitates system enhancement. Traditionally, systems designed around a single processor have to be replaced once optimal performance limits are reached. In the case of the average user, this implies a major financial investment which he may not always be able to afford. However, using the modularity property, a multiple processor system can be upgraded to meet new requirement at minimal costs thus prolonging system life.

In general, system enhancement may be desired to eliminate a bottleneck discovered in the existing system, to add more features to the existing system and/or to improve the existing system performance in terms of speed and power. Also, a modular

system, implemented with multiple processor architecture, will allow fine tuning of the system's operation. Only one portion of the system needs be modified without affecting the rest of the system.

(d) HUMAN ENGINEERING ASPECT

Finally, one should mention the man/machine interface. There are increasingly large numbers of applications where computer controlled systems are being developed for use by the average non-professional users. Such systems are generally highly interactive and must accommodate users who may have relatively little or no experience in computer technology. The human engineering concept must be used so as to provide a simple man/machine interface which can be easily understood by the average users. This requirement can be most economically achieved by using a local dedicated processor which will provide the proper interface.

2.5 WHEN TO USE/NOT USE A MULTI-MICROCOMPUTER SYSTEM?

As mentioned in the introduction, the question "When to use a multiple processor system?" is dealt with specifically in terms of microprocessors. The reason being that the application of such a system, is closely related to the capabilities of the various processors used in the system. In general, a microprocessor does not have the computational power of a larger computer (due to technological constraints), nor the communication flexibility (due to the limited number of pins). However, the microprocessor can provide a

versatile and low cost source of computing power which can be used in applications such as data acquisition, measurement, and/or supervisory and control.

Microprocessor technology has made it practical and financially attractive to apply computer control to large numbers of new applications. For some of the less elaborate applications, the use of a single microprocessor is sufficient to handle the computational requirements. However, multi-microcomputer systems should be considered for applications which have some or all of the following characteristics:

(a) Elaborate process control applications having diversified computational demands coupled with real-time constraint. For these applications, the heavy processing requirements generally far exceed the capabilities of a single microprocessor based system.

(b) Applications with extensive amounts of I/O processing requirements. The need to interface with a large variety of I/O processes generally imposes unacceptable control overhead on a single microprocessor and causes a severe degradation to the system's responsiveness.

(c) Applications which demand high reliability but, due to financial and/or space constraints cannot afford the use of massive redundancy.

In certain complex applications it is good practice to attempt to decompose the operation of the overall system

into a number of independent tasks. These tasks can be executed on individual or cooperating microprocessor based systems. In line with this, one should generally avoid using multi-microcomputer systems if one or both of the following conditions exist:

(i) If all that is needed is more raw processing power. In this case, it is advantageous to get a more powerful CPU, or add one or more dedicated specialized processors.

(ii) If the global task to be executed cannot be partitioned into relative independent tasks with minimal intertask communication needs.

2.6 FUNCTIONAL DESIGN OF A MULTI-MICROCOMPUTER SYSTEM

Up to this point, whenever possible, multiple processor systems were considered in general, without limiting the discussion to a particular processor size. In the following, the presentation is strictly about multi-microcomputer systems. The functional design of a multi-microcomputer system can be defined in terms of the following aspects: physical and logical structures, task arbitration, task allocation, and tasks interaction and coordination.

The physical structure of multi-microcomputer systems is considered in terms of the various interprocessor communication arrangements and the physical interconnection topologies. In line with this, the possible logical designation assigned to various elements in these systems is discussed. Note that both

structures are strongly interrelated and must be well defined prior to obtaining the actual interprocessor communication protocol.

By using the logical abstraction of physical entities, the various aspects associated with the control and coordination of system resources are considered. Therefore, the analysis becomes hardware independent and is uniform across different system architectures.

2.6.1 PHYSICAL STRUCTURE

The physcial structue of a multi-microcomputer system is a function of the interprocessor communication arrangement and the interconnection topology.

(a) INTERPROCESSOR COMMUNICATION ARRANGEMENTS

The data transfers between processors can be carried out either via a common memory structure or via a direct bus structure often referred to as centralized and distributed structures respectively [RAMA 77]. The common memory structure shown in figure 2.6 requires a direct connection from each element to a common memory. Memory access is controlled via a centralized multiple access memory interface unit. All data transfers are done via the common memory and elements have no direct access to each other. This arrangement doubles the common memory access time associated with each transfer as data must be written by one processor and then read by another one. In systems with frequent data transfers and/or large amounts of data, this structure is not efficient,

due to increased memory contention which causes system performance to degrade.

In the system bus structure there is a direct connection between elements, thus the time the resource is being utilized (i.e. local memory and bus) is half of that of the common memory structure. In the most general case all data transfers are initiated and performed in a distributed fashion. However, this implies that each processor performs the various control aspects associated with interprocessor communication. In order to relieve the processors from performing this duty, a modified structure shown in figure 2.7 is suggested. With this structure, all requests for data transfers are routed via a centralized bus controller which decides which processors will be granted the use of the bus. However, as with the common memory structure, since the bus is a shared resource, the number of data transfers it can support is limited by the contention associated with it. Later, in Chapter 4 a scheme called a Segmented Bus Architecture will be described which enhances the capabilities of system bus structures to carry more data.

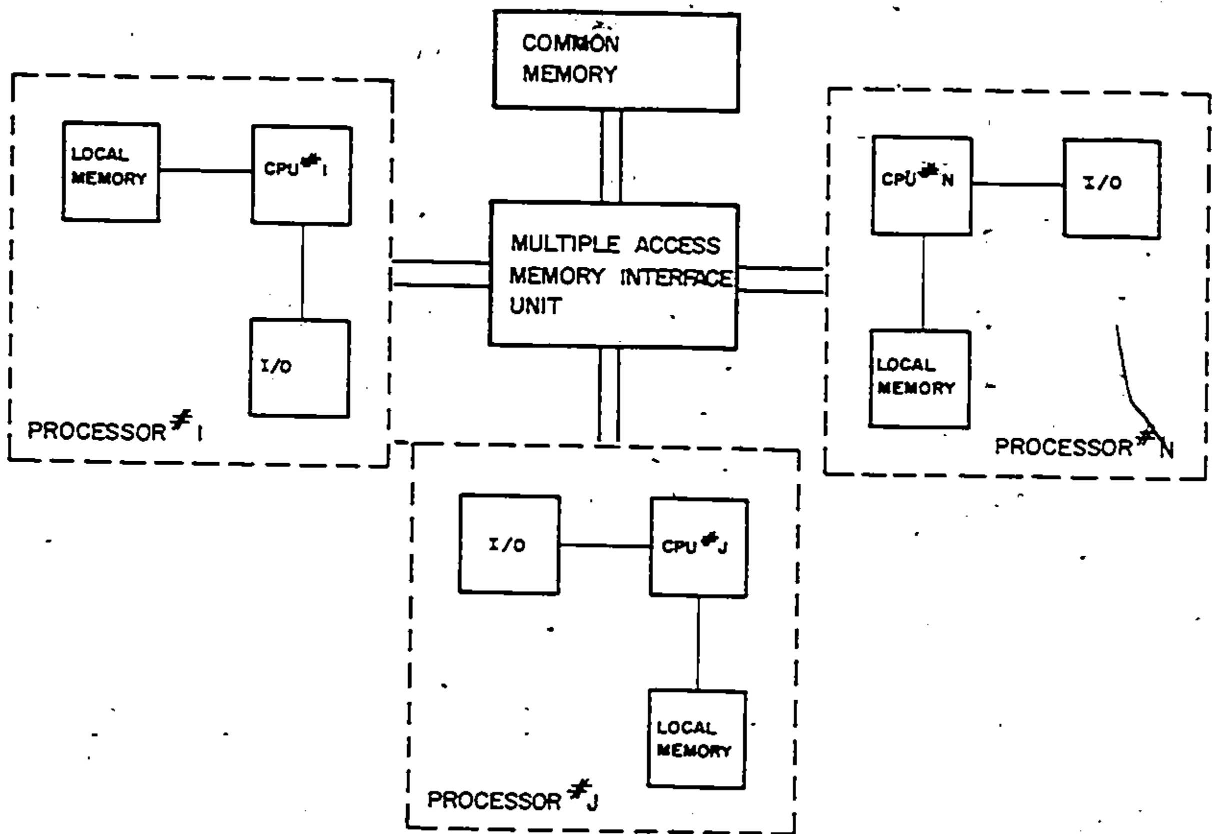


FIGURE 2.6 – INTERPROCESSOR COMMUNICATION
VIA
COMMON MEMORY STRUCTURE

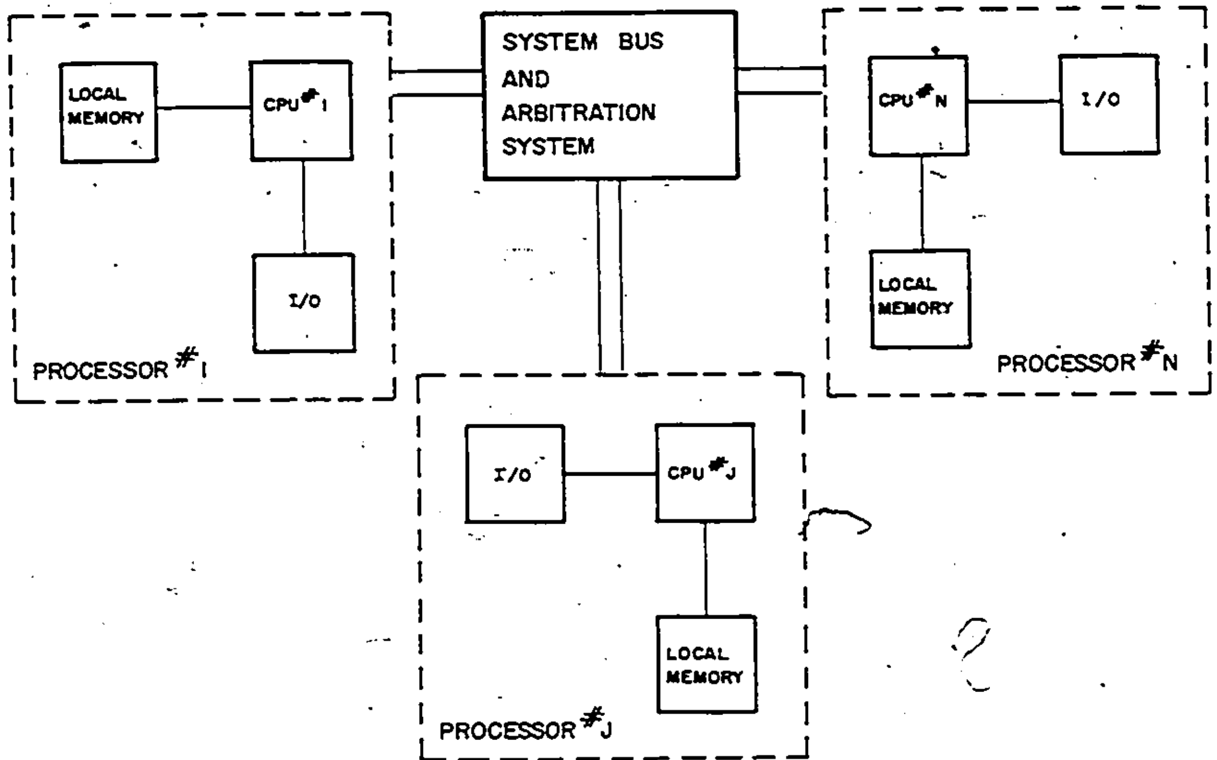
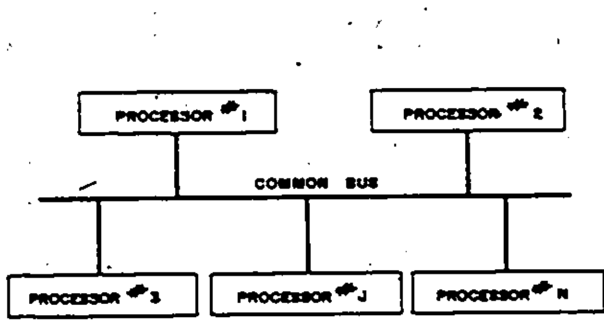


FIGURE 2.7 – INTERPROCESSOR COMMUNICATION VIA SYSTEM BUS STRUCTURE

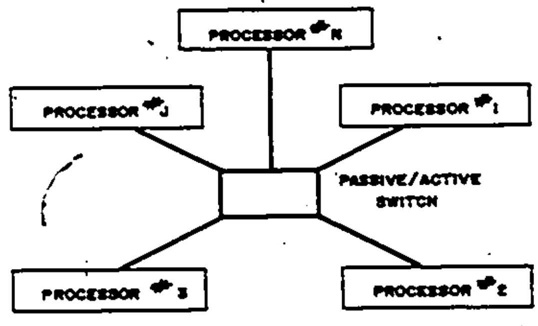
(b) INTERCONNECTION TOPOLOGIES

The way in which the elements are connected to the bus and to each other defines the interconnection topology of the system. Physically, there are many ways of interconnecting N elements in a system. Two important factors that should be considered when establishing the interconnection scheme are reliability and expandability. In terms of reliability the interconnection scheme should provide an alternate path in the event of a link failure. The expandability of the interconnection scheme is necessary in order to facilitate the addition of more elements to the system without affecting the existing structure.

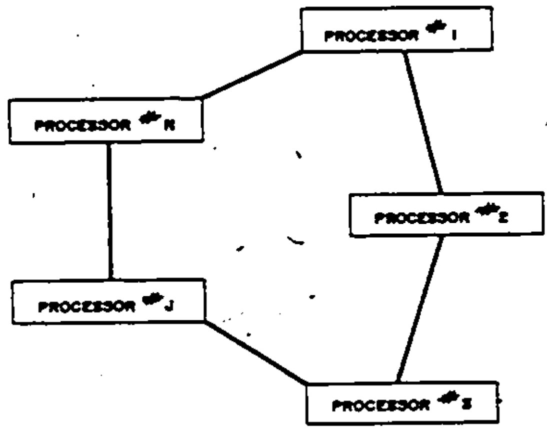
Anderson [ANDE 75] defined four design decision levels regarding the interconnection strategy of a computer system. The first decision level is concerned with the transfer strategy, namely should messages be passed from source to destination directly or indirectly? The second decision level defines the transfer control method, when using indirect connection, as either centralized or distributed routing. The third and fourth decision levels are concerned with the actual implementation based on the decisions made under the first and second levels. One must decide whether to use a dedicated or shared message transfer path and once this is determined, define the specific system architecture. Here, the four most basic interconnection schemes: Common Bus, Star, Ring, and Fully Connected, as shown in Figure 2.8, are considered.



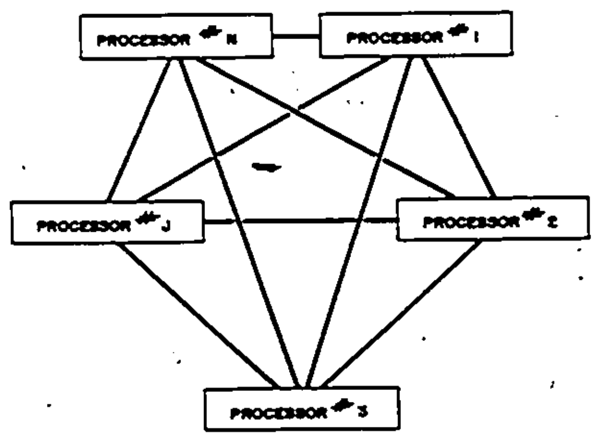
A - COMMON BUS



B - STAR



C - RING



D - FULLY CONNECTED

FIGURE 2.8 - INTERCONNECTION TOPOLOGIES

In the Star topology, message routing can be accomplished by a passive device like a read/write memory (by allocating special mail slots to each element) or by an active element like a processor with an elaborate communication protocol. The Fully Connected topology is rarely used in a multi-microcomputer system due to the large number of connections and the complexity of the required control scheme. The Common Bus is widely used, however, its main drawback is its low reliability and its restriction to the amount of data transfers it can support. The Ring topology is very useful for local loop networks and for data gathering systems.

Other interconnection topologies are basically combinations or variations of these four schemes and will not be considered here.

2.6.2 LOGICAL STRUCTURE

Logical structure refers to the various logical relations that can exist between the elements in a multi-microcomputer system. A system can be either vertical or horizontal. The elements in a vertical system are hierarchically structured implying a master-slave relation. On the other hand, in horizontal systems, the elements are logically equal, implying a master-master relation.

(a) VERTICAL ORGANIZATION

The vertical organization in its simplest form has a single master with multiple slaves. The main characteristics of such a system are:

(i) All elements are not logically equal.

(ii) All interprocessor communications must go through the master or be initiated by the master.

(iii) The slaves hardware may be identical to each other with customizing to a special task done via the software.

(iv) The control allocation mechanism is as follows:

(a) Either only one element is capable of assuming the master role, or

(b) Several elements are designated as masters, but only one element is active at any given instant while the others are on active standby.

A single master-multiple slave organization is shown in Figure 2.9. In general, in such systems "Number Crunching" is done by the master processor (which is generally the most powerful) whereas the "I/O

Processing" is carried out by various I/O slave processors,thus achieving a very high throughput[RUSS 77]. Systems may also contain more than one level of master-slave arrangement, thus forming a pyramid configuration.

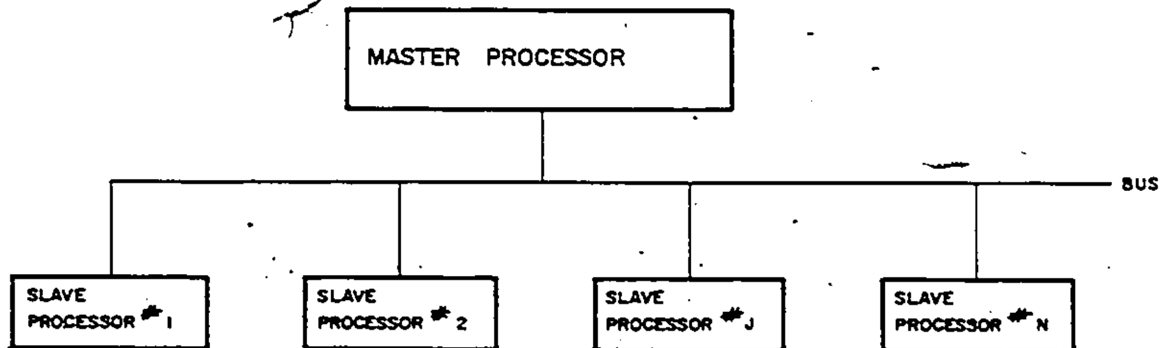


FIGURE 2.9 – SINGLE MASTER – MULTIPLE SLAVE ORGANIZATION

(b) HORIZONTAL ORGANIZATION

The main characteristics of the horizontal organization are:

(i) All elements are logically equal.

(ii) Any element may communicate with any other element in the system.

(iii) All elements must be able to support a compatible interprocessor interface.

The control allocation mechanism may be as follows:

(a) Either no designated master; All elements carry out their dedicated task without using a centralized controller, or,

(b) Floating master; each element can execute the master functions according to a master schedule list.

In general, a horizontal system is more flexible and capable of dynamic load sharing. However, it is not efficient when there are large numbers of vastly different tasks. A Block Diagram of a fully connected master-master organization is shown in Figure 2.10.

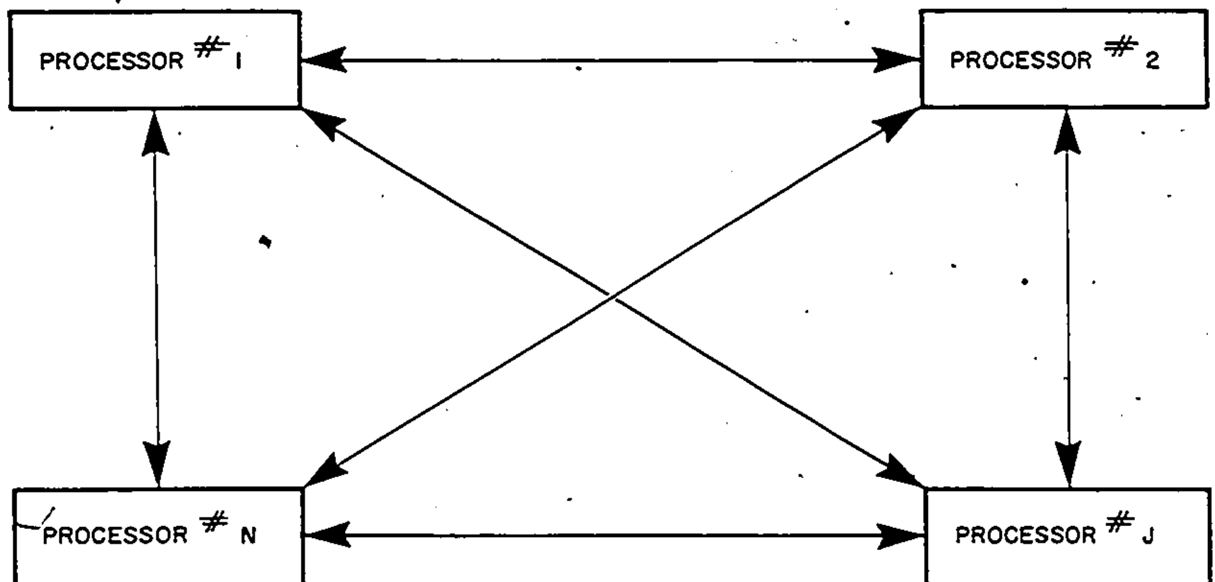


FIGURE 2.10 – FULLY CONNECTED MASTER – MASTER ORGANIZATION

2.6.3 CONTROL AND COORDINATION ASPECTS OF A MULTI-MICROCOMPUTER SYSTEM

One of the most important features of multi-microcomputer systems is the concurrency concept, namely the ability to execute more than one activity at the same time. In order to sustain this ability, the system must be able to perform the following control aspects: arbitration, allocation, and coordination. In general, in multi-microcomputer systems the various tasks are initiated asynchronously by external stimuli and/or internally within the system. This may introduce execution difficulties in terms of concurrency and conflicts [CHIE 79]. However, by using the concepts of process/task, events and process/task communication as logical equivalents of the actual physical structures, one can deal with the problem of concurrency and conflicts independently of the system structure [BRIN 79]. These concepts will be described in detail in Chapter 3 when a design for an executive which controls the various aspects of task scheduling and execution in a multi-microcomputer system, is presented.

(a) ARBITRATION

In order to execute an efficient task/resource allocation, the system must be able to resolve all potential contention by identifying the various events/requests and verifying the status of the task/resource whose services are being requested. The real-time demands for task/resource are physical entities in the form of asynchronous requests/interrupts. Apart from the resolution of contention and arbitration, the question of

scheduling or synchronization of the task/resource with these asynchronous activities is necessary. These demands can be represented by events which are their logical equivalent. Thus, the system software can treat all demands the same way regardless of their physical characteristics. In multi-microcomputer systems, contention resolution can be achieved by using either arbitration or test and set technique, each of which is explained in subsequent paragraphs.

THE ARBITRATION TECHNIQUE

The global arbitration procedures can be represented pictorially as a "black box" whose inputs are the requests for task/resource and whose outputs are the request grants. Such a representation is shown in Figure 2.11.

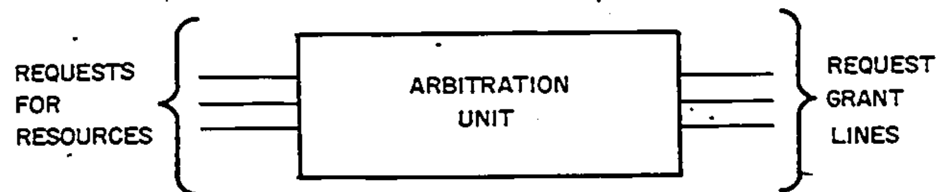


FIGURE 2.11 - PICTORIAL REPRESENTATION OF THE ARBITRATION PROCEDURE

Arbitration procedures can be carried out centrally by a single self-contained unit or in a decentralized fashion by small dedicated units, distributed among the various elements in the system [THUR 72]. The most commonly used arbitration control schemes are: daisy chain, polling, and asynchronous requests/interrupts.

The arbiter speed should match the device characteristics in order to reduce the time "wasted" during the arbitration process. This is especially important in centrally controlled arbitration procedures as the total arbitration speed is inversely proportional to the number of control lines.

THE TEST AND SET TECHNIQUE

The resolution of contention and proper resource/task allocation can be carried out by using the test and set technique. This technique is based upon the use of semaphore, a software-settable flag or switch which specifies the availability status of a resource. The semaphore is read by either the task allocation processor or by the elements competing for the services of the particular resource in order to determine the resource availability . However, in order to avoid erroneous readings, some locking mechanism must be used to prevent a reading by one element while the semaphore is being updated by another. Such a mechanism was implemented on some second generation microprocessors such as the 6809E and some 16 bit microprocessors. For example, the 8086 16 bit microprocessor has a software lock instruction that affects an output signal which may be used to coordinate access to shared resources, thus avoiding erroneous readings. In systems which require faster operations, the resource availability flags can also be set by hardware.

(b) TASK/RESOURCE ALLOCATION

One important factor which characterizes multi-microcomputer systems not found in single procesor based systems is the resolution of contention on shared resources and their proper and orderly allocation.

One can regard as a shared resource any part of the system, hardware or software, which potentially may be used in more than one process, where a process is a logical entity related to the execution of any well defined procedure. In multi-microcomputer systems one can consider a process as the logical grouping of one or more tasks with their associated data, where each task represents the smallest independent entity of procedure. As outlined previously, a multi-microcomputer system consists of a number of cooperating task execution units. In other words, a process execution may involve the assignment of a number of units. Thus in these systems, resource allocation and task allocation are a synonomous concept. The allocation method may be either static or dynamic.

STATIC ALLOCATION

In systems with static allocation, tasks/resources are assigned to specific processes and are unalterable under real-time conditions. These systems are simpler to design and implement because they do not require a special task allocation process. They may require more system resources as their physical and logical structures are very

rigid and they cannot accommodate dynamic load balancing. Future expansion and/or improvements must be obtained by adding new elements, or changing the existing system structure. Finally, a failure of a single task/resource implies the failure of the processes which use it.

DYNAMIC ALLOCATION

In systems with dynamic allocation, the various tasks/resources are assigned to any process upon request. This makes the system more flexible as the allocation is done in real-time, but requires more complex design. Dynamic allocation may be controlled by a floating executive where all elements in the system are capable of assuming this role, or by a special dedicated task allocation unit. The floating executive is more reliable due to duplication, but requires re-entrant code and lock out capabilities for some critical regions of the code [SEAR 75], where critical regions are segments of code which are non-sharable. A special dedicated task allocation unit, even though not as reliable, greatly reduces the software complexity of the system.

(c) TASK/PROCESS INTERACTION AND COORDINATION

Another characteristic of multi-microcomputer systems, not found in uniprocessor systems, is the need for interaction and coordination among the system's elements. As mentioned previously, in multi-microcomputer systems process execution corresponds to the execution of a number of

cooperating tasks. In general, some sort of interaction and coordination among several concurrent tasks is necessary. Thus, one can talk about intertask/process communication procedures, instead of communication between physical entities. This provides a uniform description of all system elements, and one need not bother with actual hardware characteristics. Also, by providing the proper task/process communication procedures at a logical level, (before system implementation), one can define more efficiently the actual physical commuication procedures.

Bowen [BOWE 80] defined four coordination problems associated with concurrent processes: determinancy, synchronization, deadlock, and mutual exclusion.

The determinancy problem deals with the issue of keeping common data at a known state at all times. In order to ensure that data is always determinate any data alterations must be well coordinated.

The synchronization problem deals with the fact that cooperating processes may require some degree of synchronization in order to properly coordinate the use of a shared resource.

The deadlock problem defines the condition that occurs when one process is waiting for an event to be initiated by another process which never takes place. Further details about the deadlock problem can be found in [ISLO 80].

Finally, the mutual exclusion problem is concerned with the fact that once a resource has been allocated to a process, it must remain under its control until the process finishes using it.

2.7 A TASK DRIVEN - MULTI-MICROCOMPUTER SYSTEM

Up to now, multiple processor systems have been considered in general, followed by detailed discussion about the functional design of a multi-microcomputer system. Here, a specific architecture known as a Task-Driven Multi-Microcomputer System [KRIE 79] is presented. This architecture was developed to exploit the advantages associated with concurrent processing yet to maintain a simple and reliable system. The salient characteristics of this architecture will be highlighted as some of them will be used later as the basis for further development.

2.7.1 SALIENT CHARACTERISTICS OF A TASK-DRIVEN SYSTEM

A Task-Driven multi-microcomputer system consists of a number of microprocessors, each having its own local program memory, data memory and some I/O capability. In addition, the system utilizes a global common memory for passing messages and for storing common variables.

The resources available to the system, including the various microprocessors are controlled by a dedicated task allocation and arbitration unit via a control bus and dedicated control lines. The task allocation and arbitration unit accepts requests from internal and external sources and initiates the proper control actions. The required task coordination is done via

handshake signals connecting the unit with the various elements in the system. The complexity of this unit may range from a hardwired logic circuit for simple appliations, to an intelligent controller in more elaborate systems. An executive architecture designed specifically to match this architecure will be described in Chapter 3: . A simplified block diagram of a Task-Driven multi-microcomputer system is shown in Figure 2.12.

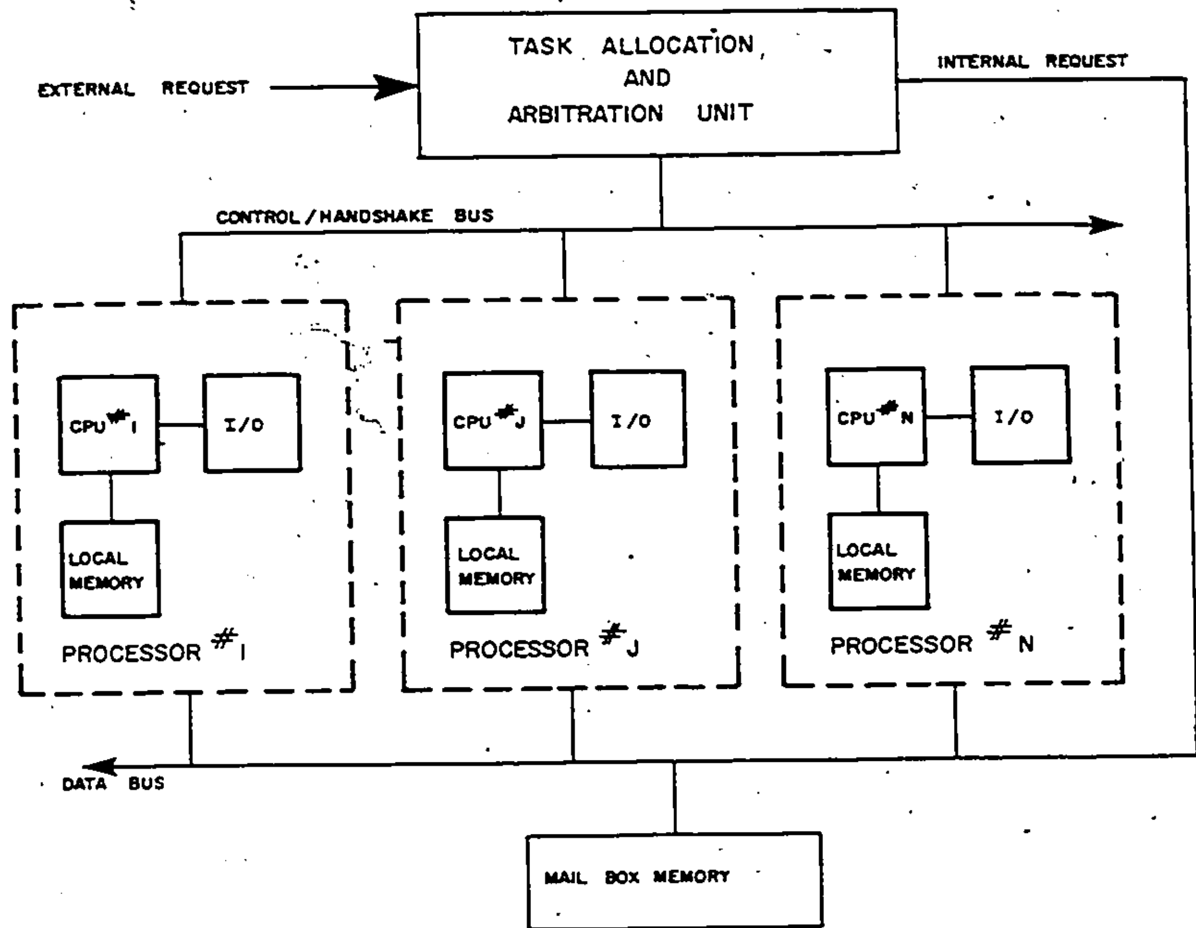


FIGURE 2.12 - SIMPLIFIED BLOCK DIAGRAM OF A TASK - DRIVEN MULTI - MICROCOMPUTER SYSTEM

To reduce conflicts, thus enhancing system responsiveness, and for reliability purposes, the code required for execution of a specific task is duplicated in the local memory of more than one processor. The controller verifies the availability status of all processors capable of executing the specific task, and determines which one will be assigned to execute the task. The controller utilizes time-out procedure which together with the handshake signals can be used to verify the status of a processor and in the event of a failure, to restart the task on another processor. The time-out procedure can be also used to indicate the real -time task execution progress, namely the remaining time for execution. This information is checked when deciding whether or not to suspend the execution of a task, thus reducing the system coordination overhead which in turn, improves system performance.

All new requests for task execution are put momentarily on a queue and are subsequently assigned to the various processors. If a processor capable of executing a specific task is not found, and it is not time critical, it remains on the queue until a processor becomes available. If on the other hand, a task must be executed in real-time and there are no available appropriate processors, the controller suspends a currently running task. Subsequently, the vacated processor is assigned to the time critical task. Finally, note that the use of a dedicated task allocation and arbitration unit "frees" the processors in the system from performing anything else but their assigned tasks. The individual processor's complexity is

reduced, which in turn results in modular, more reliable units. By designing the controller to be a reliable unit, the complete system reliability is enhanced.

2.8 CONCLUSIONS

The material presented in this chapter is meant to convey the following major points: First, to clarify some of the terminology associated with multiple processor systems and in doing so to give an insight into the topic of multiple processor systems. Secondly, to show the applicability of microprocessors in relation to this concept and the various aspects involved in the design of such systems.

The presentation followed the "onion peel" approach. The outer layer covered general aspects of multiple processor systems such as the various organizations and the motivations behind using these systems. Also included in this layer was a brief discussion about a multi-microcomputer organization and when to use a multi-microcomputer system. In the middle layer, one finds a detailed description of the functional design of a multi-microcomputer system. It includes the physical and logical structures, and some control and coordination requirements for this system. The innermost level, describes a Task-Driven Multi-Microcomputer Architecture, which will be used as the basic architecutre for the subsequent chapters.

2.9 REFERENCES

ANDE 75 L.H. Anderson, "The Micrcomputer as Distributed Intelligence", Proc. of the International Symposium on Circuits and Systems, Boston, Mass., April 1975, PP 337-340.

ANDE 75 G.A. Anderson and E.D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics and Examples", ACM Computing Surveys, Vol. 7, No. 4, Dec. 1975, PP 197-214.

BAER 76 J.L. Baer, "Multiprocessing Systems", IEEE Trans. on Computer, Vol. C.25, No. 12, Dec. 1976, PP 1271-1277.

BAUM 75 A. Baum and D. Senzig, "Hardware Considerations in a Microcomputer Multiprocessing System", Computer Technology to Reach the People - Digest of Papers-COMPCON Spring 75, San Francisco, Calf., Feb. 1975, PP 27-30.

BOWE 80 B.A. Bowen and R.J.A. Buhr, "The Logical Design of Multiple Microprocessor Systems", Prentice Hall Inc., Englewood Cliffs, N.J., U.S.A., 1980.

BRIN 79 P. Brinch-Hansen, "A Keynote Address on Concurrent Programming", IEEE Computer, May 1979, Vol. 12, No. 5, PP 50-56.

CHIE 79 Y.P. Chien, "Multitasking Executive Simplifies Realtime Microprocessor System Design", Computer Design, Jan. 1979, PP 109-117.

DAVI 77 C.G. Davis and C.R. Vick, "The Software Development System", IEEE SE-3, No. 1, Jan. 1977, PP 69-84.

ENSL 77 P.H. Enslow Jr., "Multiprocessor Organization - A Survey", ACM Computing Surveys, Vol. 9, No.1, Mar. 1977, PP 103-129.

FLYN 66 M.J. Flynn, "Very High Speed Computing Systems", Proc. IEEE, Vol. 54, No. 12, Dec. 1966.

FULL 78 S.H. Fuller et al, "Multi-Microprocessors: An Overview and working example", Proc, IEEE Vol. 6, No. 2, Feb. 1978, PP 216 - 218

HIGB 73 L.C. Higbie, "SuperComputer Architecutre", IEEE Computer, Dec. 1973, PP 48-58.

ISLO 80 S.S. Isloor and T.A. Marsland, "The Deadlock Problem: An Overview", IEEE Computer, Sep. 1980, PP 58-78.

JENN 77 C.J. Jenny, "Process Partitioning in Distributed Systems", Digest of Papers-NTC'77', 1977, PP 31:1 1-1-10.

KRIE 79 M. Krieger, "Task Driven Multi-Microprocessor System", Procedures of the First Canadian Workshop on the Design and Development of Computer Systems, May 1979, PP 81-88.

POPO 79 D. Popovic and D. Danziger, "Total Life Calculations With and Without Maintenance", Microprocessors and Microsystems, Vol. 3, No. 6, July/Aug. 1979, PP 257-261.

PREN 79 D. Prener, "Large Multimicroprocessor Systems", Microprocessors and Microsystems, Vol. 3, No. 6, July/Aug. 1979, PP 271-276.

RAMA 77 Ramamoorthy et al, "Hardware Software Issues in Multimicroprocessor Computer Architectures", 1st Annual Rocky Mountain Symposium on Microcomputers: Systems, Software,Architecture, Aug./Sept. 1977.

RENN 80 D.A. Rennels et al, "Distributed Fault-Tolerant Computer Systems", IEEE Computer, March 1980, PP 55-65.

RUSS 77 P.M. Russo, "Interprocessor Communication For Multi-Microcomputer Systems", IEEE Computer, Vol. 10, No. 4, April 1977.

SATY 80 M. Satyanarayanan, "Commercial Multiprocessing Systems", IEEE Computer, May 1980, PP 75-96.

SCHW 77 M. Schwartz, "Computer Communication Network Design and Analysis", Prentice-Hall Inc. Englewood Cliffs, N.J., U.S.A., 1977.

SEAR 75 B.C. Searle and D.E. Freberg,"Microprocessors", IEEE Computer, Oct. 1975, PP 75-83.

SIEW 78 D.P. Siewiorek, "Multiprocessors: Reliability, Modelling and Graceful Degradation", In System Reliability and Integrity, State of the Art Report, Infotech Ltd., Maidenhead, England.

SPET 77 W.L. Spetz, "Microprocessor Networks", IEEE Computer, Vol. 10, No. 7, July 1977.

THUR 72 K.J. Thurber et al, "A Systematic Approach to the Design of Digital Bussing Structures", Proc. Fall Joint Computer Conf., Dec. 1972, PP 719-740.

WEIT 80 C. Weitzman, "Distributed Micro/Minicomputer Systems, Structure, Implementation and Applications", Prentice Hall Inc., Englewood Cliffs, N.J., U.S.A., 1980.

WIDD 76 L.C. Widdoes Jr., "The Minerva Multi-Microprocessor", Conf. Proc. 3rd Ann. Symp. Computer Architecture, Clearwater, Fla., Jan. 1976, PP 34-39.

3.0 EXECUTIVE FOR A TASK-DRIVEN MULTI-MICROCOMPUTER SYSTEM

3.1 OVERVIEW

The design procedures and implementation aspects of a modular executive which is "tailor made" for a Task-Driven multi-microcomputer system are presented. The design philosophy of this executive differs from the general approach in that it emphasizes flexibility and simplicity of design rather than achieving maximum resource utilization. The executive has two level hierarchical structure with inter-level communications being done via asynchronous handshake signals. This results in a simple to implement, modular executive which can be easily made to fit other system architectures. Aside from simplicity of design, the main strength of this executive is that it leads to simple system verification procedures. The complete system operation can be simulated as chains of calls, thus system operation can be verified prior to actual implementation.

3.2 INTRODUCTION

This chapter considers the design of an executive which supervises the execution of various tasks in a multi-microcomputer system. In a system with a uniprocessor, only one task may be executed at any one time. However, by time sharing the processor, an apparent concurrency can be achieved. A uniprocessor system, in order to be able to support multitasking capabilities, has to use a real-time executive which schedules, controls and synchronizes the various tasks to achieve the desired apparent concurrency.

In these systems, task interrelations are well defined in terms of the particular executive used, however, this may not be sufficient to describe task interrelations in a multi-microcomputer system.

In the literature, there are a large number of papers related to real-time executive for a single microprocessor based system [TAVO 80], [LIND 80], [CHIE 79], [KAHN 79], [TOWN 77], however, much less is reported about multi-microcomputer executives [BOWE 80]. BRINCH-HANSEN [BRIN 78] proposed a new language concept for concurrent programming called "distributed processes" which is suitable for real-time applications controlled by microcomputer networks with distributed storage.

Here, an executive which is "tailor made" to fit a Task Driven system [KRIE 79] is presented. The basic idea behind the executive is to use hierarchical control for task scheduling and to provide inter-level communication via asynchronous handshake signals. This results in a highly modular, simple to implement executive which can easily be made to accommodate other system architectures.

In the following, the general system architecture is outlined first, in Section 3.3 followed by a detailed presentation of the elements of the executive in Section 3.4. Next, the various task interrelations are discussed in Section 3.5 and the main implementation aspects of the executive are presented in Section 3.6. Finally, some of the most important concepts regarding task partitioning and allocation are described in Section 3.7.

3.3 OUTLINE OF SYSTEM HARDWARE

The main feature of the proposed Task-Driven system is the hierarchical controller which supervises a number of heterogeneous processors, each having private program memory, read/write data memory, and some I/O capability. The system may also have global resources that include a global data memory which is used as a message centre and for storage of common variables. From the point of view of the executive, the Task-Driven system defined in Chapter 2, can be represented as shown in Figure 3.1.

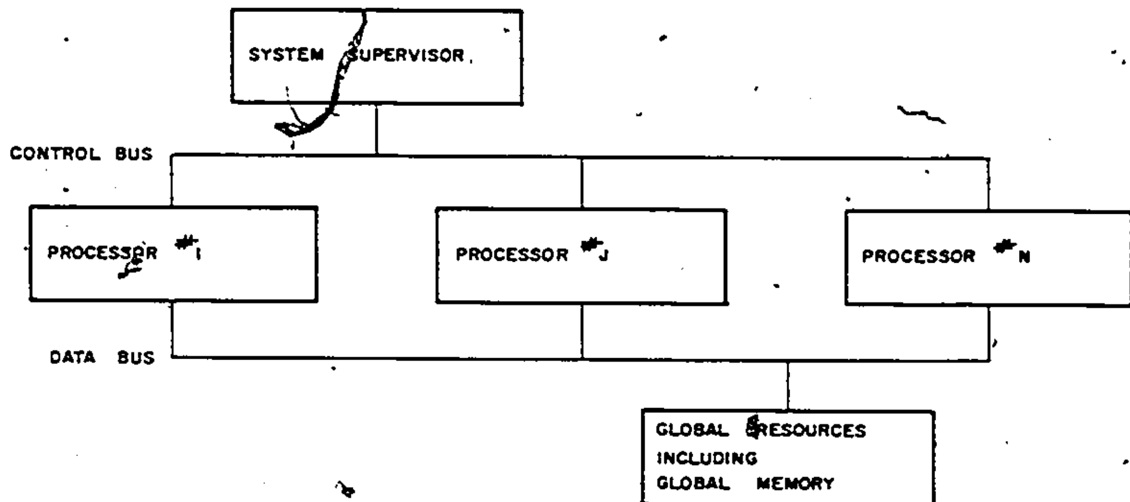


FIGURE 3.1 - SIMPLIFIED BLOCK DIAGRAM OF TASK-DRIVEN ARCHITECTURE

The various tasks available to the system are permanently stored in the local program memory of the individual processors. When a task is to be executed, it need only be awakened by specifying its starting address and some additional parameters, rather than down loaded from a central memory. In order to avoid excessive hardware duplication and to restrict the size

of the local program memory, not all tasks are duplicated in the program memories of all processors. For reliability reasons, and in order to maintain the system's performance at a specified level, each task is stored in two or more local program memories.

The duplication of tasks in various program memories necessitates the establishment of a fairly long list associating the various tasks and processors. To simplify the list, each task is assigned the same apparent address in each processor which is capable of executing it. Thus if a task appears in the local program memory of three different processors which may be heterogeneous, its apparent starting address in all three processors is the same. In order to allow as much flexibility as possible in the organization of the local program memories, the apparent addresses point to a jump instruction which will direct the processor to the actual starting address of the specified task. Thus, each processor must have an area reserved for apparent addresses. A typical processor organization and a profile of its program memory is shown in Figure 3.2.

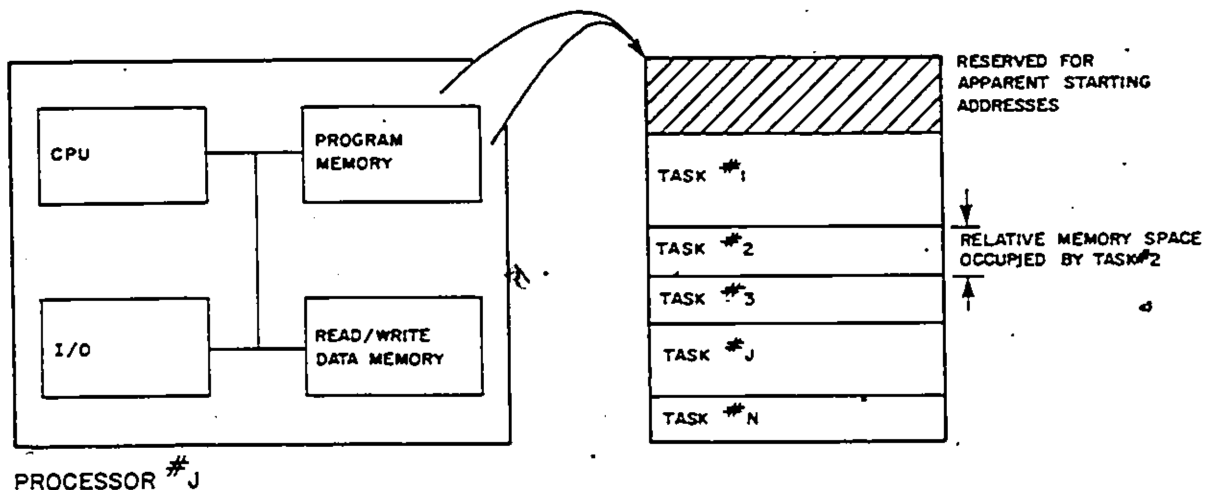


FIGURE 3.2 - TYPICAL PROCESSOR ORGANIZATION AND A PROFILE OF ITS PROGRAM MEMORY.

3.4 ELEMENTS OF SYSTEM EXECUTIVE

In general, a system can be partitioned into a number of units which can be represented by their logical abstractions called processes. Each process consists of one or more tasks, where the task is considered to be the smallest logical entity in the system. This can be represented graphically as shown in Figure 3.3.

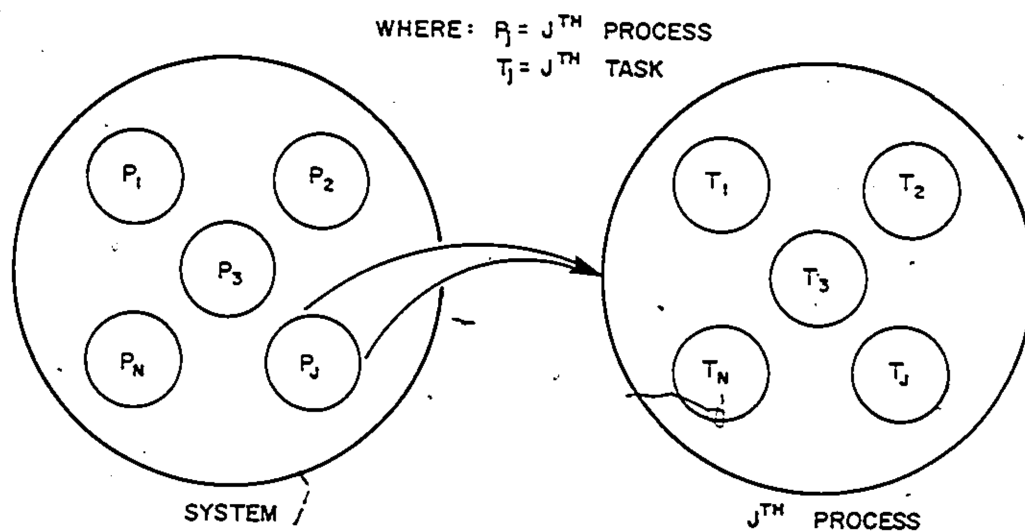


FIGURE 3.3 – GRAPHICAL REPRESENTATION OF A SYSTEM IN TERMS OF PROCESSES AND TASKS.

Whenever one considers the execution of a number of processes/tasks, because of possible data interdependency, one has to define a sequencing relation. Some processes/tasks can (a) be started at

the same time, (b) have to wait until another process/task is partially executed, (c) have to wait until another process/task is completely executed, as shown in Figure 3.4. Note, that the distinction between sequencing relation (b) and (c) is only important in the case of processes and/or large tasks. This simple notation can be used to represent graphically the complete sequencing relation of any number of processes/tasks.

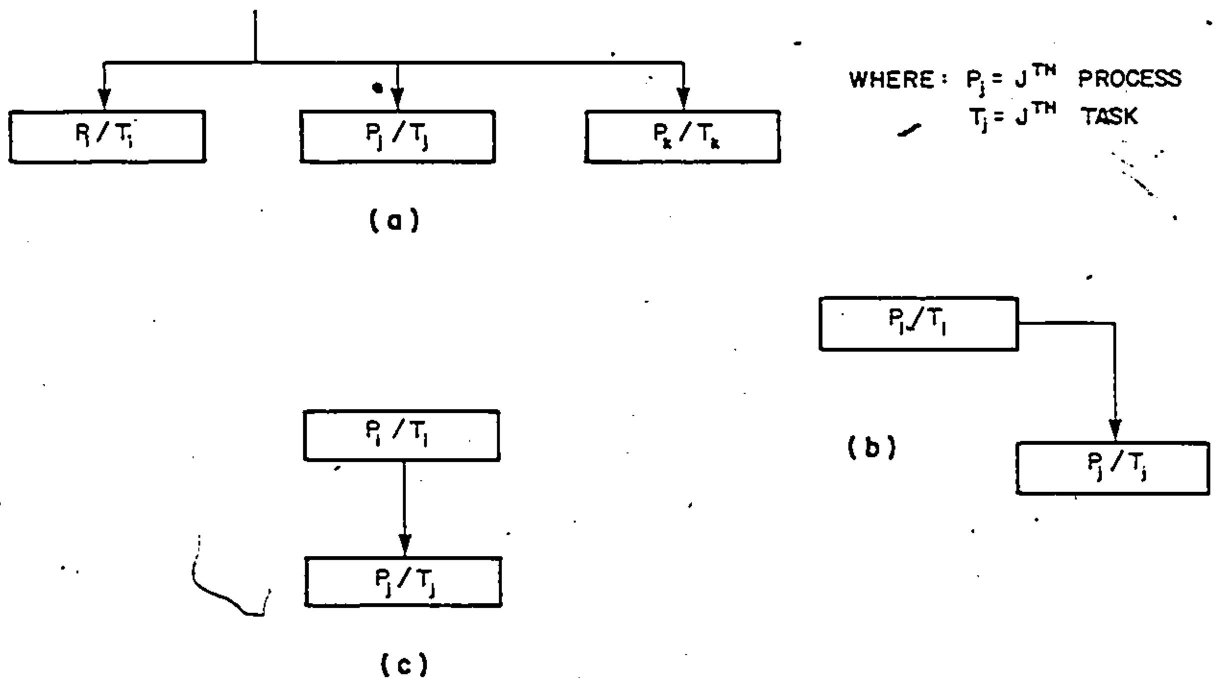


FIGURE 3.4 – GRAPHICAL REPRESENTATION OF PROCESS/TASK SEQUENCING.

3.4.1 DESIGN PHILOSPHY

In a multi-microcomputer system, the executive has to maintain an orderly execution of tasks in the most concurrent fashion possible. The design philosophy behind the proposed executive is different than the general design approach, in that the emphasis is placed on simplicity and flexibility rather than on achieving maximum utilization of resources. This philosophy is based on the fact that hardware is cheaper than software [GALL 81], and one can afford to under-utilize hardware resources in order to obtain flexibility and simplicity. In particular,to reduce complexity, the following restrictions are introduced:

(a) Only a few types of tasks are allowed, each with specific interaction capabilities.

(b) All communications are done via well defined logical interfaces.

(c) Task initialization is only done by a centralized entity.

To provide maximum flexibility the executive is designed using the following principles:

(a) The executive has multi-level hierarchical structure.

(b) Communication between levels and within each level is done via asynchronous handshake signals.

(c) The executive is defined and designed in a modular fashion to allow modifications and expansion.

3.4.2 EXECUTIVE HIEARCHICAL STRUCTURE

The executive model consists of two levels: the PROCESS LEVEL which contains all the relevant information about process relations and system parameters, and the TASK LEVEL which maintains information about task relations within each process.

At the process level, the executive utilizes a PROCESS MANAGER which controlls the process sequencing. It also performs task-processor assignments based upon requests for service received from the various TASK MANAGERS operating at the task-level. Aside from the above, the process manager controls all the communication activities with the external world.

At the task level, the executive has a number of active TASK MANAGERS, one for each currently running process. Each task manager controls the actual task sequencing for a given process by accepting information from the tasks it controls. Based on this, it initiates new requests to the process manager.

The communication activities between levels and within each level is handled via asynchronous handshake signals. The two level executive model can be represented graphically as shown in Figure 3.5.

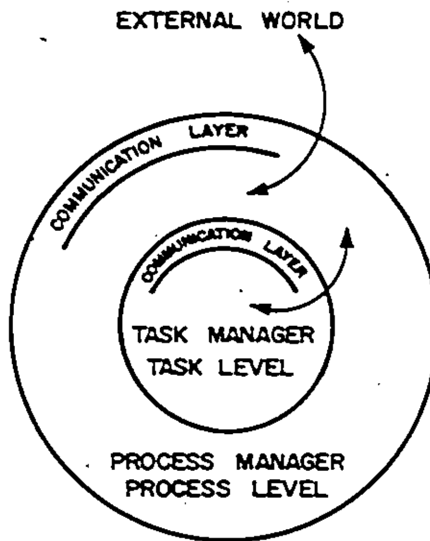


FIGURE 3.5 – GRAPHICAL REPRESENTATION OF TWO LEVEL EXECUTIVE MODEL.

3.4.3 PROCESS LEVEL

The process manager consists of two level hierarchical structure. The outer layer of the process manager is the PROCESS SEQUENCER which communicates with the external world. It controls, at the process level, all the various activities not directly affecting the system hardware. Also, it transfers to the next level of the process manager, the proper information regarding the next process to be executed. The inner level of the process manager is the TASK ALLOCATOR which communicates with the system hardware and is responsible for the task assignment process. It awakens the various task managers, transmits the proper parameters unique to each

process, assigns mail slots in the central memory, and controls all subsequent task-processor assignments. More details about the task allocator are given in Section 3.6.

The actual problems involved in process and task sequencing are exactly the same, therefore a detailed description of only one of them is considered. The task sequencing problem will be presented here, since tasks are the logical entities which have to be initiated on the system hardware. This will provide an insight into the system implementation aspects.

3.4.4 TASK LEVEL

At the task level there are two distinct types of tasks: MASTER tasks (TASK MANAGERS) and SERVICE tasks. The task manager has the responsibility to coordinate the task sequencing within a given process and the communication activities associated with it. During system operation, the number of active task managers varies according to the number of currently running processes. In this system, the distinguishing factors between various processes are the sequencing and specifications of the tasks associated with each process. Therefore the master task can be a general purpose program, which receives the parameter specifications unique to each process from the task allocator. Aside from the task sequencing relations, the task allocator also associates with each task manager, one or more mail slots in the central memory for passing data. Thus, in order to transfer data between tasks, one needs only to pass pointers rather than move actual data. In order to enhance system

responsiveness, this program is included in each processor's local program memory, and each individual processor can act as a task manager.

Due to the overhead associated with process suspension, a master task, once initiated, cannot be suspended. Thus, the processor assigned to it becomes dedicated to its execution. However, the master task execution may be terminated voluntarily based upon information received from one of its tasks, or be aborted by the process manager as a result of external information.

The master task controls several service tasks which communicate with it via asynchronous handshake signals. Hence, it must be capable of requesting service tasks at any time during the course of its execution.

In contrast to the master tasks, service tasks are special purpose programs. They can be of two types: The majority of them are REGULAR service tasks which cannot call any other tasks during their execution. A regular service task can be suspended only once by the process manager when the processor on which it is running is required by a higher priority task. The other service task type is the PRIVILEGED service task of which there are only a limited number. These tasks can call only one regular service task at a time. To reduce overhead, neither the privileged tasks nor the service tasks it calls can be suspended. By restricting task calls and suspensions in this fashion, one avoids nested loops of tasks, and thus the processing overhead associated with general multi-level suspensions. Also, one need not change task priority after suspension. However, a tag must be associated with (a) privileged

tasks, (b) regular service tasks called by privileged tasks, and (c) suspended regular service tasks, indicating that they cannot be suspended.

It should be noted that in the event that a currently running task is displaced, after the execution of the higher priority task, control need not be returned to the displaced task. The next task to be executed is decided using a global priority list which varies depending on the system's dynamic conditions. This list is maintained by the process manager which assigns the appropriate task to the available processor.

3.4.5 EXECUTIVE COMMUNICATION PROTOCOL

The handshake procedure used for the request, assignment, and execution of a given task is presented here in terms of a communication protocol. For illustration purposes, the control signals associated with a single task are described, although a task manager may request the concurrent execution of a number of tasks. It is assumed that the kth task manager requests the execution of only task i and the process manager assigns processor j to execute it. The communication protocol steps are as follows:

(a) At time t_1, task manager k initiates a request to the process manager, identifying itself and the requested task.

(b) The process manager, upon receiving the request, after a short processing time delay (dt) acknowledges receiving the request. This is done in order to indicate to the task manager that its

request is being looked after and to have some sort of check on the process manager. If the task manager does not receive an acknowledgement within a defined time period, it repeats its request and if it again does not receive an acknowledgement, it generates an alarm condition.

(c) After a search time t_s, the process manager assigns task i to processor j by identifying the apparent starting address and the task manager which requested it.

(d) The process manager notifies task manager k that it assigned task i to processor j. This is necessary so that the task manager can identify in advance the processor with which it has to communicate.

(e) Processor j sends a message to the task manager indicating that it is ready to start execution of task i. If this signal is not received, the task manager signals the processor manager to check the processor status.

(f) Task manager k specifies to processor j the mail slot pointers for data transactions.

(g) Task manager k signals to the process manager that execution of task i has been started. The process manager may use this signal to start a down counter which indicates the remaining time for execution. This can be used as additional data when deciding which task to suspend, and in the event of processor failure to reinitiate task execution.

(h) Upon completion of the execution of task i, processor j notifies the task manager k that it has completed execution.

(i) Task manager k signals the process manager that the execution of task i is completed.

This communication protocol along with the signals used is shown graphically in Figure 3.6.

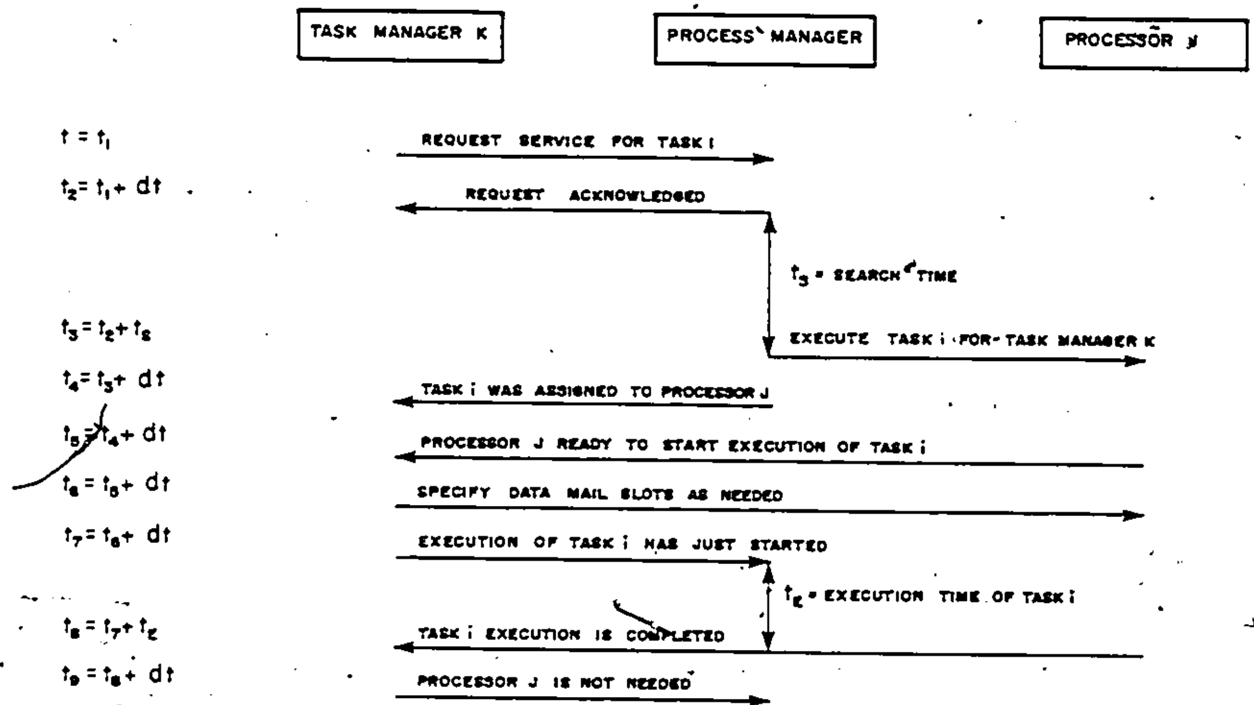


FIGURE 3.6 - COMMUNICATION PROTOCOL USED DURING TASK CALL AND EXECUTION.

3.5 TASK INTERRELATIONS

After a brief discussion of task hierarchies, task interrelations are presented in terms of a state digaram.

3.5.1 TASK HIERARCHIES

Tasks may be classified into two categories based on their interdependency and time relation. The task interdependency defines whether tasks are INDEPENDENT or RELATED. Independent tasks can be considered separate entities, as each is capable of operating alone and does not require the cooperation of other tasks in order to complete its execution. Related tasks on the other hand, interact with each other through cooperation or competition. It should be noted that not all independent tasks can be executed in parallel as their simultaneous execution may involve some conflicts. Also, not all related tasks need be executed in completely sequencial fashion.In line with this, one can have the classification tree shown in Figure 3.7.

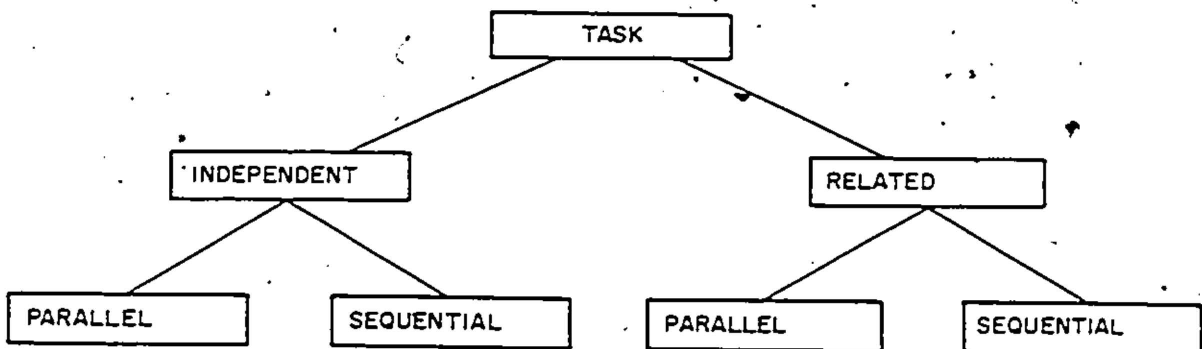


FIGURE 3.7 – TASK CLASSIFICATION TREE

Ideally, in order to achieve maximum freedom, one would like to partition the process into a number of independent and parallel tasks, which can be executed simultaneously. In practice, such partitioning is seldom feasible, rather, a collection of tasks which permit some form of concurrent execution can be obtained.

Next, task hierarchical structure based on the urgency of execution and their relative importance is considered. Using real-time constraint as a measure of urgency, one can classify the various tasks into URGENT (time critical) and NON-URGENT. Urgent tasks always take precedence and must be given the immediate attention of a processor, whereas non-urgent tasks need not be executed immediately. Furthermore, within each of these groups, tasks are asigned various levels of priority, based on their relative importance, and are executed accordingly. This will permit some tasks to displace a non-urgent and/or lower priority currently running task, if necessary.

3.5.2 TASK STATES AND STATE DIAGRAMS

To provide a systematic presentation of task interrelations, the concept of state diagrams can be used. In the state diagram, the states represent task status, and state transitions are executed under the influence of proper control primitives. Note that all the concepts presented in this section apply equally to process interrelations and can be used to define the actual process control.

At any time a task may exist in one of the following states: INACTIVE, READY, RUNNING, WAITING, or SUSPENDED. By defining the TASK EXECUTION CYCLE as the path taken by the task from the time of activation to the time of completion, the shortest path would involve going through only the READY and RUNNING states as shown in Figure 3.8. Note that the task execution cycle shown in Figure 3.8 corresponds to the ideal situation, whereas in practice, it may involve detours through other states (WAITING or SUSPENDED), or may even be aborted prior to being completed. In the following, each state is considered in detail separately, first by defining its properties, and then by describing its relations with other states. The interrelations between a given state and other relevant states are shown in terms of a state transition diagram. The labels associated with the state transitions correspond to the control primitives that govern the actual transitions.

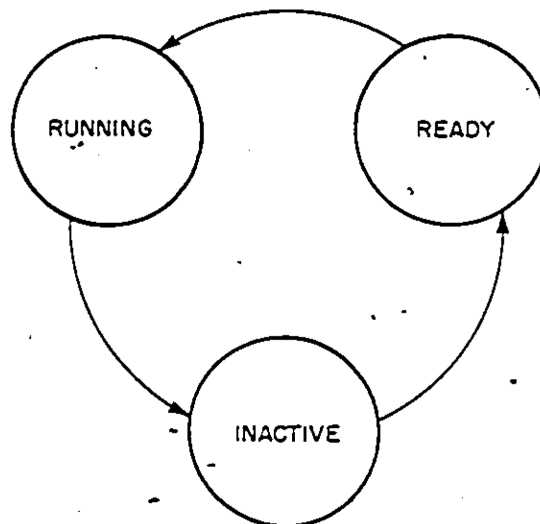


FIGURE 3.8 – THE IDEAL EXECUTION CYCLE

3.5.3 THE INACTIVE STATE

The INACTIVE state is an initial state in the sense that all tasks must start and finish in it. The inactive state acts as a source for tasks which have not yet been scheduled for execution and as a sink for completed or aborted tasks. The state transitions associated with the inactive state are shown in Figure 3.9.

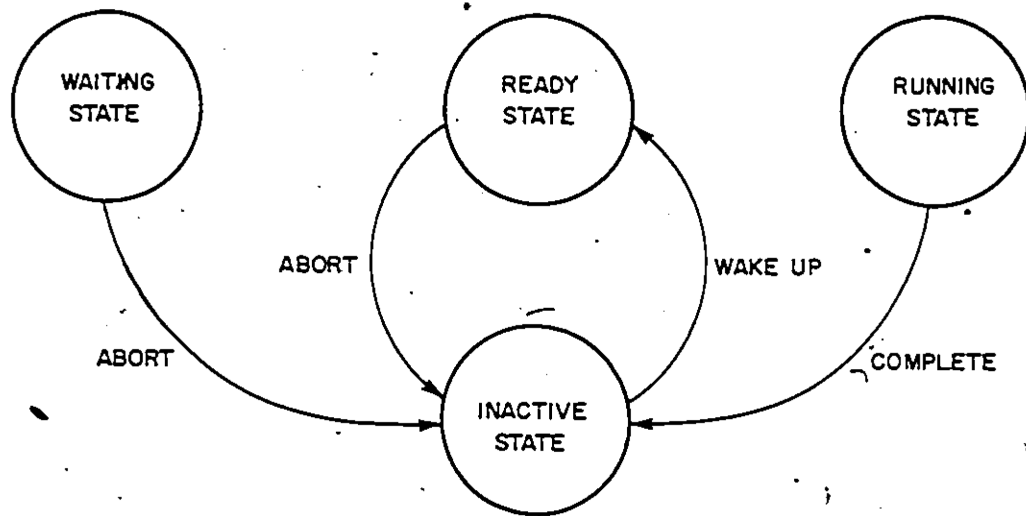


FIGURE 3.9 – STATE TRANSITIONS ASSOCIATED WITH THE INACTIVE STATE.

A task is transferred out of the inactive state to the ready state via the WAKE UP primitive. Upon completion of execution, a task is transferred from the running state into the inactive state via the COMPLETE primitive. Tasks are also transferred from the ready and waiting states into the inactive state via the ABORT primitive. Tasks in the ready state are aborted if the system has to respond to an emergency condition which will make subsequent execution of the task invalid or obsolete. Tasks in the waiting state can also be aborted as a result of information obtained from other

tasks or conditions set externally to the system.

3.5.4 THE READY STATE

A task is READY if it could be running, but a higher priority task is currently running and there are no other free processors which are capable of executing its code. As soon as the appropriate processor becomes available and there are no other higher priority tasks waiting to be executed, the task is assigned to the processor for its subsequent execution. The state transitions associated with the ready state are shown in Figure 3.10.

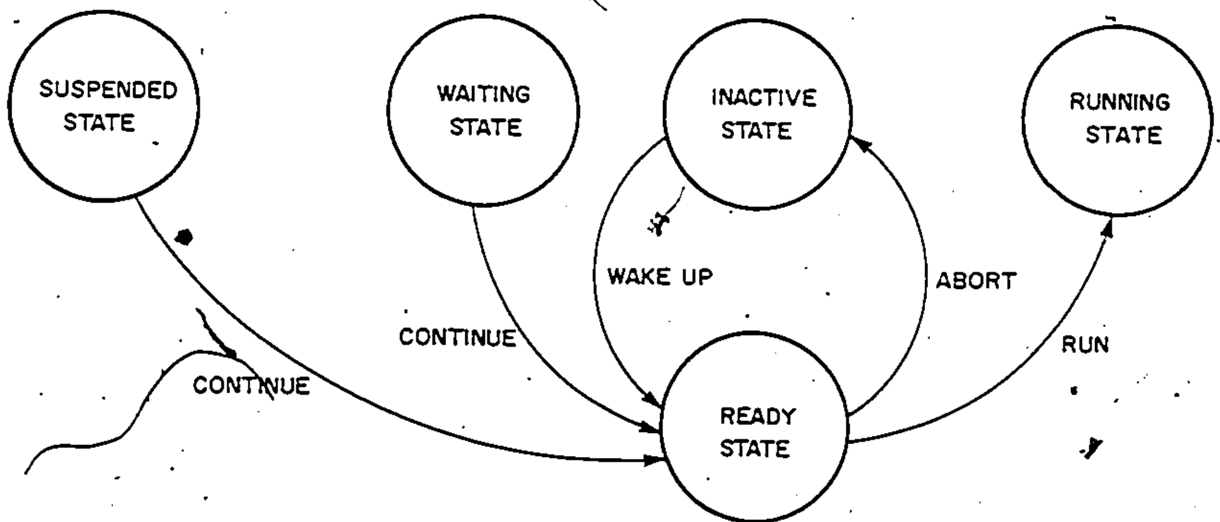


FIGURE 3.10 - STATE TRANSITIONS ASSOCIATED WITH THE READY STATE.

The transitions between the ready and inactive state are as discussed before. Tasks are transferred from the ready state to the running state via the RUN primitive whenever the appropriate processor becomes available. Using the control primitive CONTINUE, task can be transferred to the ready state from the waiting and

suspended states whenever the WAIT and SUSPEND conditions are removed. These conditions are described later when discussing these states.

3.5.5 THE RUNNING STATE

In general, the maximum number of running tasks in the system is equal to the number of active processors which are currently executing tasks. A task is in the RUNNING state if it has a processor assigned to it that is executing its code. In this sense, it is the only true active state in the task execution cycle. The state transitions associated with the running state are shown in Figure 3.11.

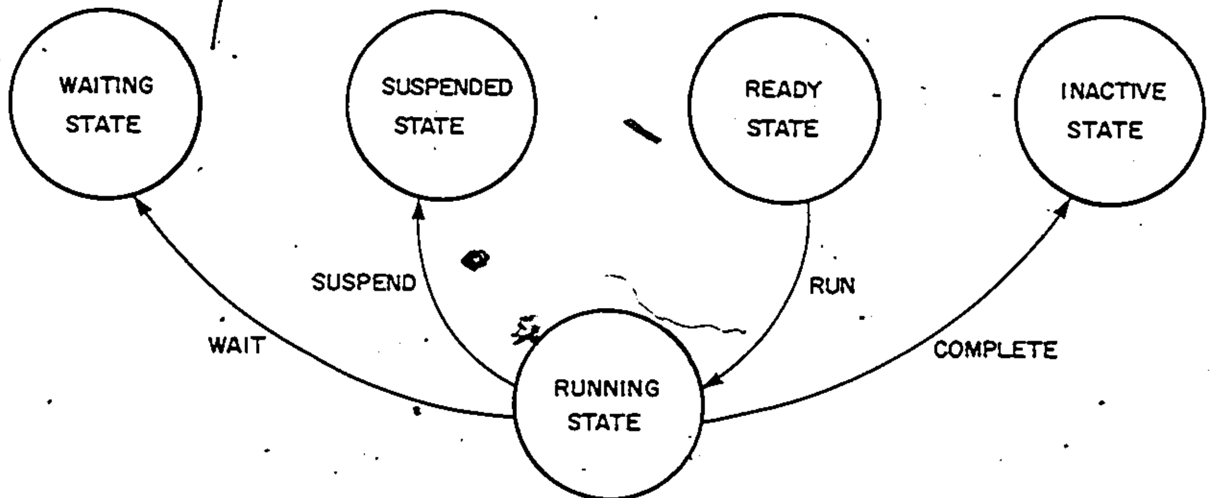


FIGURE 3.11 - STATE TRANSITIONS ASSOCIATED WITH THE RUNNING STATE.

The transitions between the running state and the inactive and ready state are as described before. A task is transferred from the running state into the waiting state via the WAIT primitive if it must wait for an external event (a message from another task or time

delay) before its execution can be resumed. Task execution can be suspended if a higher priority task must have the same processor. In this case the task is transferred to the suspended state via the SUSPEND primitive. Note that whenever a task has to be put in the waiting or the suspended state, one has to store the proper information before releasing the processor, so that it can resume its execution from the same point.

3.5.6 THE WAITING STATE

A task is in the WAITING state during the time that it has to await the occurrence of a directly related and predictable event. The state transitions associated with the waiting state are shown in Figure 3.12 and are as described previously.

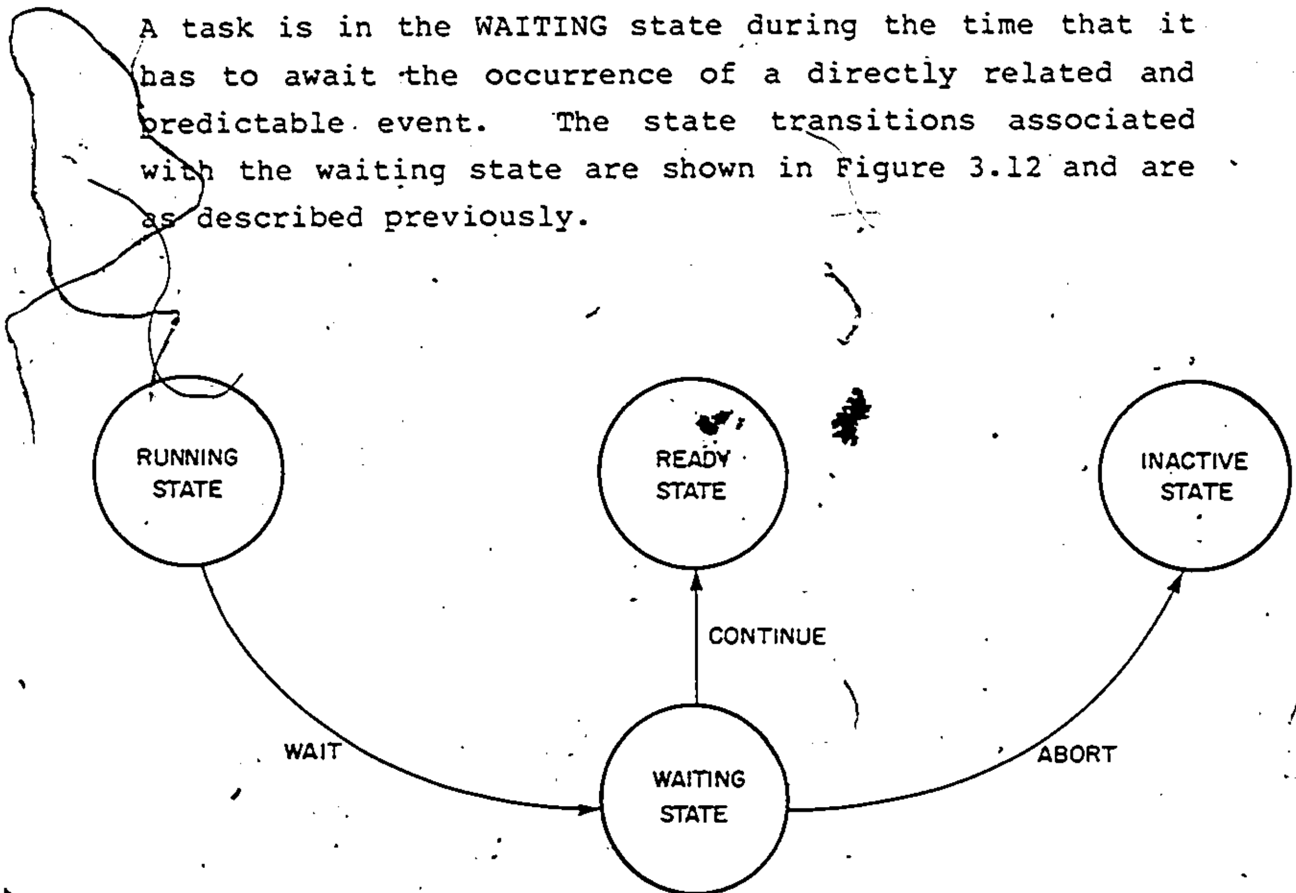


FIGURE 3.12 - STATE TRANSITIONS ASSOCIATED WITH THE WAITING STATE.

3.5.7 THE SUSPENDED STATE

A task is in the SUSPENDED state if its execution had to be discontinued as a result of system conditions unknown to it and not under its control. The state transitions associated with the suspended state are shown in Figure 3.13 and are as described previously.

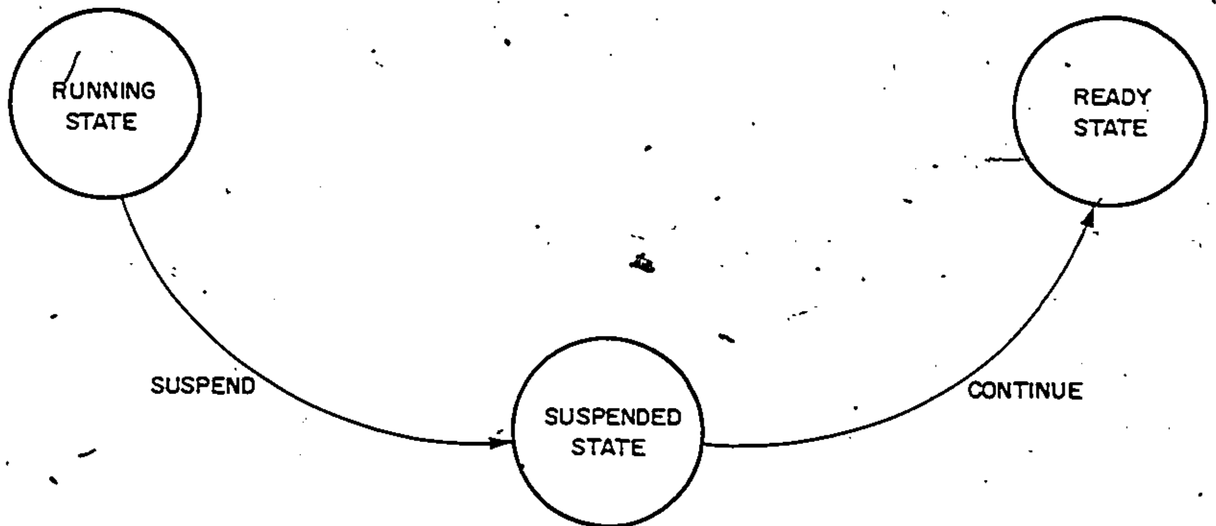


FIGURE 3.13 – STATE TRANSITIONS ASSOCIATED WITH THE SUSPENDED STATE.

Both of the above states correspond to a temporary inactive state, as no code of the tasks in these states is being executed. However, some activity is done on behalf of a waiting task, but no activity is directly associated with a suspended task. As stated earlier, before a task can be transferred to either of these two states, relevant information concerning its status must be stored. This information can be stored in local memory or in global memory. Storing the information in local memory implies that the task can be restarted only on this specific processor. This increases the

contention for the specific processor, and thus may cause unnecessary delay in the restart of waiting and/or suspended tasks. Storing the information in global memory eliminates this problem, since the task can be restarted on any appropriate processor, but it may increase the contention associated with global memory. Note that when using global memory the suspended state becomes only a transition state as tasks are automatically transferred to the ready state.

3.5.8 SYSTEM STATE DIAGRAM

A complete state transition diagram with all the control primitives associated with the system, is shown in Figure 3.14.

The actions associated with the control primitives of the above state transition diagram are listed below:

WAKE UP - remove task from inactive state
RUN - start immediate execution of the task.
COMPLETE - task execution is completed
ABORT - return to inactive state
WAIT - relinquish control of processor and store relevant information
SUSPEND - relinquish control of processor and store relevant information
CONTINUE - return to ready state.

The difference between wait and suspend primitives is in the type and amount of relevant information that has to be stored.

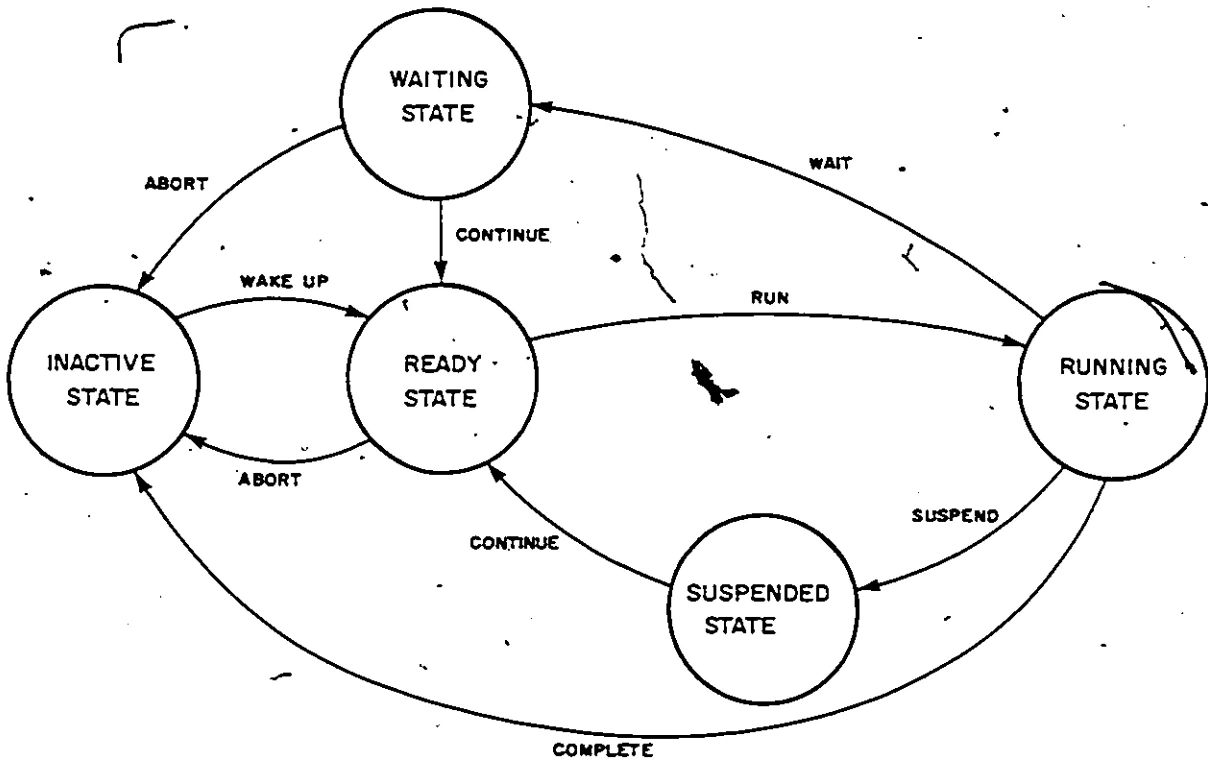


FIGURE 3.14 - COMPLETE SYSTEM STATE TRANSITION DIAGRAM.

3.6 IMPLEMENTATION ASPECTS

The main role of the executive is to respond in an efficient manner to various asynchronous service requests by assigning the appropriate processor to execute a requested task. In the suggested design, the executive utilizes a number of look-up tables to control task assignments. The actual decisions are made in hierarchical fashion based upon the contents of the various tables. These tables include the relevant system information and are updated whenever a task changes its status. In this sense, this structure can be considered as a set of asynchronous Task-Driven lists. This design philosophy matches well the Task-Driven multi-microcomputer architecture decribed in Section 3.3. These tables, beside facilitating control, can be also used for monitoring purposes to provide statistics about system utilization.

Aside from the simplicity of design, the main strength of this executive is the ease of obtaining system verification. Since all task control is based upon interrelations among the various tables, the complete system operation can be easily simulated as chains of calls and verified prior to actual implementation. The simulation can be a generalized procedure and for each system, one needs only to provide its unique hardware and operational parameters. Examples of the parameters are the number of processors along with the associated tasks each one is capable of executing, the estimation of the time required to execute each task, and the proper sequencing relations at each level of the executive. The simulation procedure can be used to detect any bottleneck or sequencing problems. Based

upon these results, one can adjust and optimize system performance either by further partitioning critical tasks, rearranging task allocation, or by using processors with different characteristics. More details about the partitioning and allocation of tasks are described in Section 3.7. The design and implementation of a general purpose simulator and optimizer is a specialized research topic on its own.

As previously defined, the executive is made up of the process manager and task managers. Each task manager is a simple general purpose program whose functions were defined previously. The process manager in turn, includes two parts, the process sequencer and the task allocator. Here, the exact definition of the task allocator will be described in detail, since it is the most complex and critical element of the executive. Furthermore, in most of its aspects, the process sequencer can be regarded as a simplified version of the task allocator.

First, a general outline of the task allocator will be described as well as a generalized flow diagram of the task assignment procedure. Next, the various lists used by the executive will be discussed followed by a detailed flow diagram of the task assignment procedure.

3.6.1 TASK ALLOCATOR,GENERAL OUTLINE

The task allocator sub-system is responsible for the various activiities associated with task assignment. Considering the required activities of the task allocator, one can identify the following five modules:

TASK ORDERING MODULE

In a system, each task may assume various priority levels based on its relations to other tasks within the same process, and its process relation to other processes. Therefore, the task allocator must be able to determine the proper ordering of all tasks which have to be executed. The task ordering module utilizes both static information about process relations and task priorities, as well as currently available dynamic data about the system, in order to generate the proper ordering of the tasks to be executed. The output of this module is a list, called the READY list, of all tasks to be executed starting from the highest priority task. In addition, as described later, the ready list also contains various parameters associated with these tasks which are used for control and decision purposes.

TASK INITIALIZATION-COMMUNICATION MODULE

This module handles the various communication activities associated with the initialization of a task on a particular processor. It accepts hardware signals and software messages from the various task managers, as well as from the processor which is assigned to execute the particular task, and responds with the proper control signals.

TASK SUSPENSION MODULE

In this system, in order to optimize performance, the decision to suspend a task is based on dynamic information. Whenever a time critical task has to be executed, the task suspension module forms a list of

tasks that are candidates for suspension, based upon the processors that can execute this task. Using their individual priorities and the remaining time for execution for each of the candidate tasks, the module determines the proper task to be suspended. The task initialization-communication module uses this information to initiate the task suspension procedure, and the assignment of the urgent task to the vacated processor.

LIST UPDATING MODULE

This module is used to maintain all lists used by the task allocator. Whenever there is a change in the system which affects any of the lists, the appropriate entries in the relevant tables are updated.

GLOBAL MEMORY MANAGEMENT MODULE

The task allocator stores relevant information associated with various tasks on a global memory, whose content is continously changing during system operation. In order to keep track of this activity, a MEMORY MANAGEMENT module is used to maintain a running file of all the currently unused memory space. The available memory is assigned upon request through the task initialization-communication module, which also signals back to it whenever memory locations become available.

3.6.2 TASK ASSIGNMENT PROCESS, GENERALIZED FLOW DIAGRAM

The task allocation process involves two decision levels:

(a) verification of availability of the processor.

(b) verification of the task execution urgency.

Based upon these decisions, one can define the generalized flow diagram shown in Figure 3.15, which is used during the task assignment process.

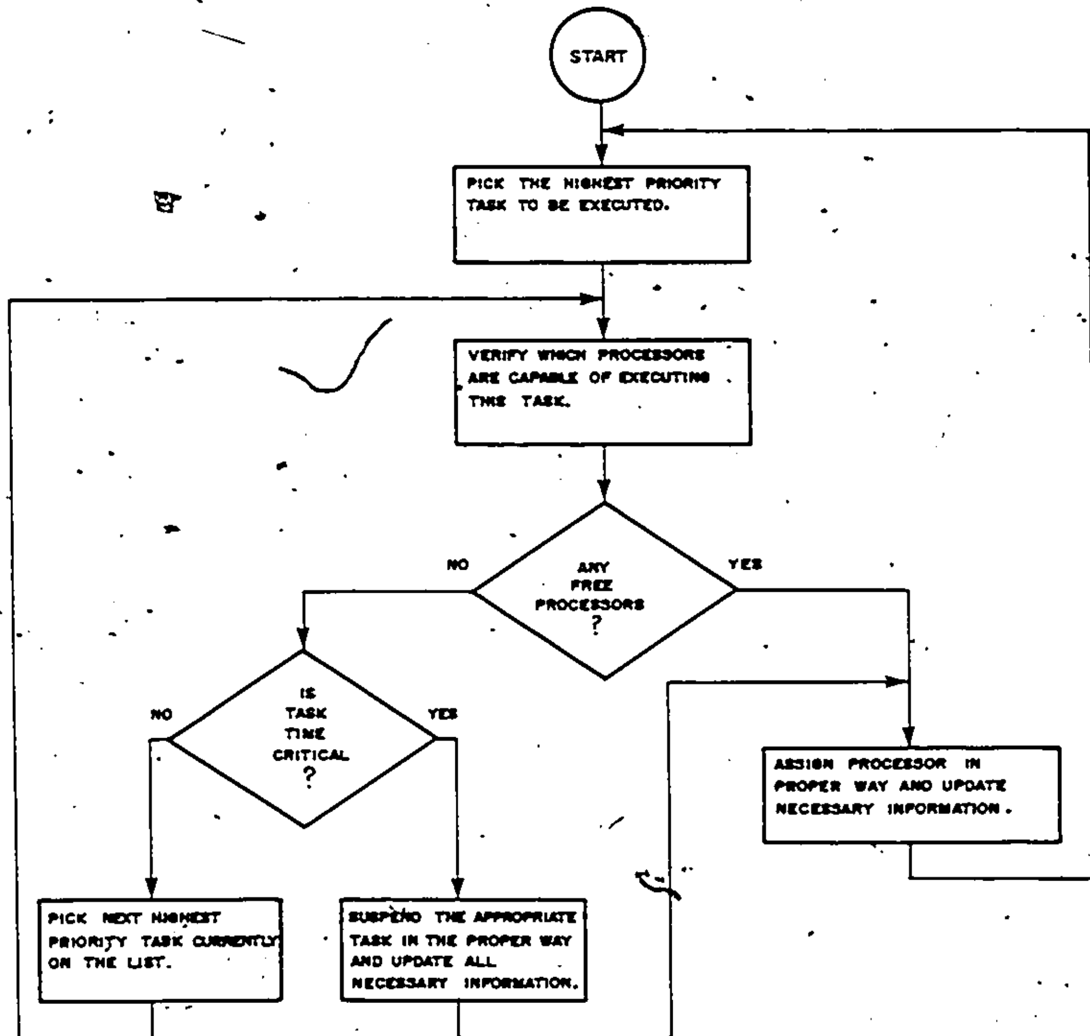


FIGURE 3.15 – GENERAL FLOW DIAGRAM OF THE TASK ASSIGNMENT PROCEDURE.

3.6.3 THE LISTS OF THE EXECUTIVE

The executive utilizes two types of lists: STATIC, which contains basic information which is fixed for a given system, and DYNAMIC, which maintains running statistics about the system's operation. The dynamic lists can be classified into two categories. The first category contains all lists which are needed for control purposes. These are the majority of lists, some of which may also be used for monitoring. The second category includes all the lists which are used strictly for monitoring purposes in order to get statistical information about the system. They are not used for any control activities and may be discarded without affecting system operation.

Whenever one associates priorities with dynamic lists, the method of inserting new elements must be defined either according to relative priorities, or in the sequential order of arrival. In the first ordering scheme, the selection of the next item for execution becomes trivial, but insertion in the proper position requires more processing. In the second ordering technique, insertion is simple, whereas removal requires more processing. The various lists are now described in more detail.

3.6.4 STATIC LISTS

PROCESS MASTER LIST

This list contains all the process sequencing relations including their relative priorities. It is used by the

process manager to generate the proper ordering of all processes waiting to be executed.

TASK SEQUENCING MASTER LIST

The proper task sequencing relations within each process are listed in this table. Whenever a task manager is awakened, the task sequencing relations of its process, as listed in this table, are transferred to it.

TASK-PROCESSOR MASTER LIST

For each task in the system, this list includes all the processors capable of executing it. The apparent starting address, which is the same in each processor, is also recorded in this list. After finding the next task to be executed, the task allocator uses this list to find which processors are capable of executing this task, and its apparent starting address.

3.6.5 DYNAMIC LISTS

PROCESS QUEUE LIST

All new requests for process execution are placed momentarily on a queue and subsequently transferred to the various task managers for execution.

ACTIVE PROCESS LIST

This list contains data about the currently active processes in the system. It is used strictly for monitoring purposes, as it provides information about how frequently each process is being called.

TASK QUEUE LIST

The process manager maintains a queue associated with the various task execution requests arriving from different task managers. The requested tasks are listed in the order of their arrival along with their relative priority within the process. Also, the requesting task manager identity is listed as it is needed for interprocessor communication. The task ordering module uses this information to insert the requested tasks in the proper priority position on the ready list.

THE READY LIST

The ready list lists in order of decreasing priorities all the tasks waiting to be executed. Tasks of equal priority are listed in the order of their arrival. For each task, one lists its task manager and the urgency level of the task. If the task is not time critical, one has to list also whether it is a new task or an old one after suspension or waiting, so it can be restored properly. Note that for an old task, if its status information was stored in a local memory, one also has to list the respective processor, since it is the only processor capable of completing its execution.

FREE PROCESSOR LIST

This table lists in a sequential order all currently free processors. It is updated every time a processor is assigned to execute a task, or if a currently running task is either completed or transferred to the waiting state. The length of this list can never exceed the

maximum number of processors in the system. This list is used by the task allocator to find the availability of a processor to execute a specific task. The list may also be used to provide statistical information regarding a processor under utilization.

ACTIVE PROCESSOR LIST

This table lists all currently active processors, together with the specific task each one is executing. In addition, it contains the information as to whether the task can be suspended, and the remaining time for execution if it can be suspended. This list is used by the task suspension module to decide on the processor that has to be suspended. It can also be used for monitoring purposes, to provide data about the actual loading on each processor. This table is updated whenever there is a change in processor status, and also periodically, to indicate the change in remaining time for execution.

THE WAIT LISTS

For simplicity, the process manager maintains two waiting lists, one based on time delay, and the other based on event occurrence. For each task, one lists its task manager and its processor if its status information is stored in local memory. The tasks in the TIME DELAY list are listed in ascending order according to the wait time. With each task, one lists only the differential delay to the previous task, the first being the absolute delay time. Using this technique, only one time measurement counter is needed. This is reinitialized every time the task on top of the list is transferred

into the ready list. Note that since this list is an ordered list, new tasks must be inserted into the proper position. Also, the time differential of the next task that follows the new inserted task on the list must be readjusted. The tasks on the EVENT list are listed in the order of their arrival, together with the code of the event for which they are waiting. One can check for event occurrence either periodically or on interrupt basis, and transfer the appropriate task onto the ready list.

THE SUSPEND LIST

This is a temporary list which contains in order of their arrival, all the suspended tasks not yet processed by the task ordering module. With each task, one lists the task manager associated with it and its processor if the status information is stored in a local memory.

With respect to both the waiting and the suspended tasks, in order to simplify control, each processor has its own SAVE and RESTORE routines. Based upon the type and amount of data, different routines may need to be associated with suspended and waiting tasks. These routines have the same apparent starting address in all processors and are treated as regular tasks. Each task manager holds the pointers either to local or global memory and transfers them upon request to the appropriate save or restore routines.

3.6.6 TASK ASSIGNMENT PROCESS, DETAILED FLOW DIAGRAM

The general flow diagram of the task assignment procedure of figure 3.15 can now be redrawn in more detail using the above defined lists, and is shown in Figure 3.16.

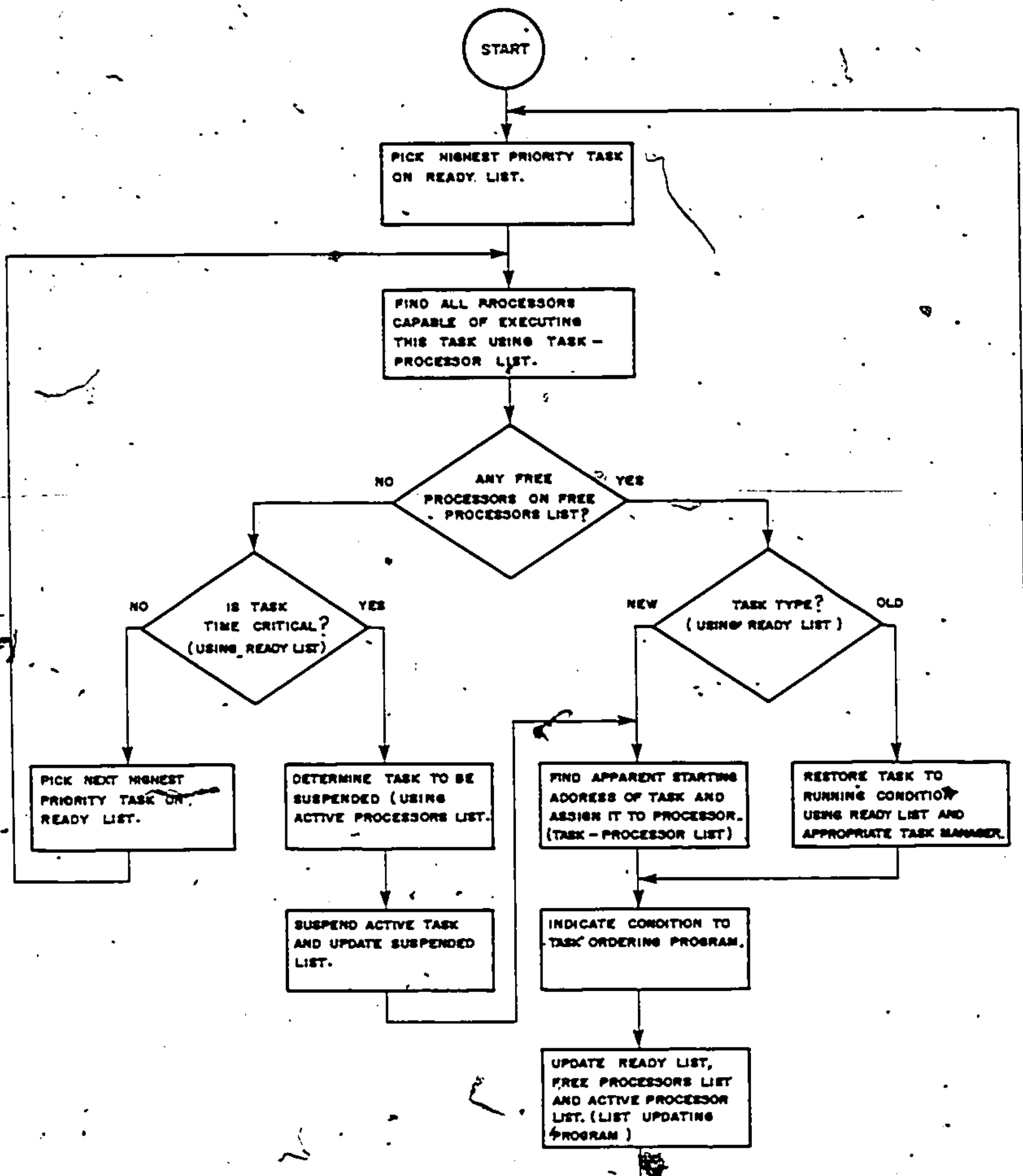


FIGURE 3.16 - DETAILED FLOW DIAGRAM OF THE TASK ASSIGNMENT PROCEDURE.

3.7 TASK PARTITIONING AND ALLOCATION

The effectiveness of a Task-Driven system and the proposed executive is highly dependent on the way the system is partitioned. As indicated earlier the partitioning process corresponds to the logical division of a job into processes, each of which is subdivided into a number of tasks, the tasks being the actual entities which are executed by the various processors in the system. The way in which these tasks are assigned to individual processors is called task allocation, which in turn greatly influences system performance. In line with this, prior to system implementation, one must execute the partitioning and allocation steps, as they directly define the system hardware.

The general problem of task partitioning and allocation, as related to distributed data processing systems was extensively studied and reported in on in [BUCK 79], [MARI 79a], [LAWS 78], [JENS 77], [GYLY 76]. Here, due to the scope and complexity of the problem, only the most important aspects of task partitioning and allocation, as related to the proposed system, will be highlighted. Furthermore, no specific optimization procedure will be described.

3.7.1 TASK PARTITIONING

The task partitioning problem is the most critical step leading to successful system development, and if executed properly, produces the following desirable effects:

(a) Simplified hardware and software levels since the system is subdivided into a number of smaller and simpler units.

(b) Modular design with improved software and hardware efficiency, as each module can be individually optimized both in software and hardware.

(c) Reduced intermodule communication overhead.

(d) Better reliability and serviceability since each module is simpler and if properly partitioned can be tested separately.

(e) Reduced system costs, since individual module design can be carried out by less experienced personnel.Aside from the financial gains, this may also be attractive in a situation where it is difficult to find highly qualified and experienced designers.

It should be noted that improper partitioning may lead to high interdependency between modules. This, besides increasing intermodule communication, also may eliminate most of the above stated benefits. In general, increased intermodule communication reduces total system efficiency, as additional processing time must be allocated for handling the intermodule communication needs.

The actual system partitioning can be done either from the bottom-up or from the top-down. Although generally the first approach is used [MARI 79b], here the top-down approach is suggested. This approach consists of two

levels; At the first level, the job is partitioned into a number of processes which are as logically independent as possible. The first partitioning level, being highly problem oriented, should be done by the system designers who are most familiar with the general job requirements. At the second level, the various processes are partitioned into tasks. This partitioning step is to be done by the computer designer in order to match the logical entities with the physical elements of the system.

An important decision in the partitioning process relates to the optimal number of modules that one associates with each partition. For example, at the task level, one extreme would be to partition the process into a large number of small tasks. This may allow for maximum concurrency of execution, but generally implies heavy intertask communication activity. The other extreme is to partition the process into a small number of large tasks. This in turn, reduces the ability for concurrency of execution. Also, since each large task requires longer execution time, there is a higher probability of suspension, which increases suspension overhead.

For best results, one must find the proper balance between maximum concurrency and minimum intertask communication needs. This is nothing else but stating in a complex way that task partitioning should always be done along natural boundaries of data and control flow [JENN 77].

3.7.2 TASK ALLOCATION

The task allocation is the next design step following the task partitioning step and corresponds to the assignments of various tasks to individual processors in the system. To maximize throughput, the task allocation procedure must provide proper load balance. In the suggested system, this is achieved by assigning a task to a number of processors, proportional to its activity. Here, activity is a measure of the frequency of the task being called and its length of execution. Furthermore, two tasks with a high level of activity will not be assigned to the same processor. These are in line with the previous statement of designing for system performance and reliability, rather than for maximum resource utilization.

Another aspect related to task allocation is interprocessor communication overhead, which is associated with the data movement required for the intertask communication activity. There are various task allocation schemes which minimize the interprocessor communication overhead [CHU 80], [JENN 77]. However, in the proposed system such minimization is not essential due to the following reasons. First, all programs are resident in local memories, thus easing the load on the communication facilities, as there is no need to down load the program from a central location. Second, most of the system control is done via a central controller using separate control bus, thus relieving individual processors from the majority of the control communication activities. The corresponding reduction in communication requirements allow the excution of all data transfers to be done via global memory. This is

done even if two consecutive tasks are executed by the same processor. Also, the fact that all data transfers are done via one central entity eases the memory management problem.

To conclude, it should be stated that due to their complexities, one can spend a lot of time and effort trying to optimize the partitioning and allocation steps. However, this may not always be economical today, because of the generally low cost of microprocessors and hardware. Furthermore, by using a general simulation procedure, as indicated before, this design step can be further simplified. Using the general simulation procedure iteratively, one needs only to start with a rough partitioning and allocation and refine them until the desired system performance level is reached.

3.8 CONCLUSIONS

In this chapter, a systematic way describing the necessary design and implemention steps of a modular executive, well suited for a Task-Driven multi-microcomputer organization, was presented. Although designed specifically for a Task-Driven architecture, this executive design can be easily adapted to other system architectures.

The executive consists of a two level hierarchical structure: a) The process level - made up of the process sequencer and the task allocator sections. The process sequencer is responsible for activities only at the process level, like deciding on the next process to

be executed, as well as the communication with the external world. The task allocator is a general purpose program, consisting of five major modules, which controls the task assignment process, and as such, bridges between the process level and the task level.

b) The task level containing a number of task managers running concurrently, each responsible for one process. All task managers are the same general purpose program, located on each processor, thus allowing each processor to become a task manager. This reduces the possibility of bottleneck associated with finding a free processor to act as a task manager.

The unique features of each process are kept at the process level and are passed to the appropriate task managers when awakened. Although the main function of the task manager is to control the sequencing of tasks within a process, at times it may also execute some tasks, if a situation permits.

In this design all the static features unique to the system, as well as the system conditions, are stored on tables. The executive utilizes these tables in an interactive fashion to control the complete system operation. By storing static information unique to a system on a read only memory, the system is fixed in its capabilities, as it cannot execute any new processes unless reprogrammed. By storing this information on a read/write memory which can be loaded from an external source, the system can be made to execute new processes, as long as they are based on tasks which are programmed in the system. For larger systems, one can use several of these executives, each controlling its own local system, which in turn are controlled by a central unit.

The various process information is prepared by the central unit and is then transferred into the proper executive for execution.

In systems which require fast processing, in order to reduce processing overhead, one can incorporate the task sequencing relations within a process with the task manager program. Thus when awakened, the task manager can immediately start requesting task execution.

By allowing each processor to have its own local data memory, one reduces the amount of contention on global memory. Also, the probability of global data corruption, in the event of processor error or failure, is reduced. However, in order to eliminate the possibility of one processor from corrupting the data used by other processors, some sort of controlled memory write access must be implemented. One such scheme may involve the writing of data into a global scratch pad and specifying the actual address in global memory where data is to be written. The memory management module may then rewrite this data in the proper location, thus eliminating any possibility of data corruption.

The problems of task partitioning and allocation were only briefly considered in this Chapter due to the scope and complexity of the subject. However, in many systems one can start with a rough trial task segmentation, and by using a simulation procedure as outlined before, find system bottlenecks. Using the statistical information from the simulation, one can decide upon a second task segmentation. These steps can be repeated until acceptable system characteristics are obtained.

3.9 REFERENCES

BOWE 80 B.A. Bowen and R.J.A. Buhr, "The Logical Design of Multiple-Microprocessor Systems ", Pretice Hall Inc., Englewood Cliffs, N.J. U.S.A,. 1980.

BRIN 78 P.Brinch-Hansen, "Distributed Processes: A Concurrent Programming Concept", CACM, Vol. 21,No.11, Nov. 1978, PP 934-941.

BUCK 79 B.P. Buckles and D.M. Hardin, "Partitioning and Allocation of Logical Resources in a distributed computing Environment", Tutorial: Distributed System Design, IEEE Computer Society, 1979, PP 247-276.

CHIE 79 Y.P. Chien, "Multitasking Executive Simplifies Realtime Microprocessor System Design", Computer Design, Jan. 1979, PP 109-117.

CHU 80 W.W. Chu et al, "Task Allocation In Distributed Data Processing", IEEE Computer, Nov. 1980, PP 57-69.

GALL 81 J. Gallacher, "Microcomputer Project Management", Microprocessors and Microsystems, Vol. 5, No. 1, Jan./Feb. 1981, PP 19-22.

GYLY 76 V.B. Gylys and J.A. Edwards, "Optimal Partitioning of workload for Distributed Systems ", Digest of Papers-COMPCON Fall '76', Sept. 1976, PP 353-357.

JENN 77 C.J. Jenny, "Process Partitioning in Distributed Systems", Digest of Papers-NTC '77', 1977, PP 31:1-1-10.

JENS 77 E.D. Jensen, "Problem Partitioning For Distributed Systems", Digest of Papers-NTC '77', 1977, PP 347 - 352.

KAHN 78 K.C. Kahn, "A Small-Scale Operating System Foundation For Microprocessor Applications", Proc. IEEE Feb. 1978.

KRIE 79 M. Krieger, "Task Driven Multi-Microprocessor System", Proc. of the First Canadian Workshop on the Design and Development of Computer Systems, May 1979, PP 81-88

LAWS 78 J.T. Lawson and M.P. Mariani, "Distributed Data Processing System Design - A Look at the Partitioning Problem", Proc. COMPSAC 78, 1978.

LIND 80 F.V.D. Linden and I. Wilson, "Real-Time Executive For Microprocessors", Microprocessors and Microsytems, Vol.4 NO.6, July/Aug. 1980, PP 211-218.

MARI 79a M.P.Mariani and D.F. Palmer, "Distributed System Design Allocation Steps", Tutorial: Distributed System Design, IEEE Computer Society, 1979, PP 277-289

MARI 79b M.P. Mariani and D.F. Palmer, "Distributed System Design Partitioning Steps", Tutorial: Distributed System Design, IEEE Computer Society 1979, PP 221 - 223.

TAVO 80 C.J. Tavora, "A Basic Technique For Realtime System Design", Computer Design, Oct. 1980, PP 147-152.

TOWN 77 D.A. Townzen, "A Task-Scheduling Executive Program For Microcomputer Systems", Computer Design, June 1977, PP 194-202.

4:0 SEGMENTED BUS ARCHITECUTRE FOR MULTI-MICROCOMPUTER SYSTEMS

4.1 OVERVIEW

One of the most important requirements of a multi-microcomputer system is to provide interprocessor communication with a high degree of availability and reliability. In this chapter, the segmented bus concept, which can accomodate simultaneous mutually exclusive transfers with minimal modifications to a standard bus, is introduced. The principle of the segmented bus is that two communicating elements require only that portion of the bus which is physically between them. The section between any two communicating elements can be isolated by inserting bidirectional switches between adjacent elements on the bus. This will permit the remaining sections of the bus to be used for other transfers. To allow for orderly communication on a segmented bus, the switches can be controlled either centrally or in a distributed fashion. The basic characteristics of two segmented bus architectures are presented: one with distributed control and the other with central control. For each architecture, the communication protocol, as well as various implementation aspects, are described. The centrally controlled segmented bus will be emphasized, as it provides a more flexible system. Also, in order to sustain the various functional requirements assigned to the central bus controller, and to maintain high reliability, it must be implemented as a multi-microcomputer system. A conceptual design of the bus controller is presented.

4.2 INTRODUCTION

One of the important aspects of multiple-processor systems is interprocessor communication, which can be done via thin line or parallel bus. In this chapter the parallel busing approach will be considered. The interprocessor communication scheme should be chosen such as to optimize system performance in terms of system throughput. Factors which should be considered when deciding on the bus organization are: flexibility, expandability, availability, reliability, and cost effectiveness.

The fully connected interconnection scheme, which provides a direct point-to-point connection, is a conflict free busing scheme. This option is not practical for large systems, since to provide complete interconnection among N elements, N!/2[(N-2)!] connections are required. Furthermore, the control aspect of such an interconnection scheme increases in complexity as N increases.

In order to reduce the number of interconnections yet maintain an effective throughput on the bus, some kind of bus sharing must be used. However, once any type of shared bus is considered, one cannot guarantee conflict free communication between the various elements. Note that in this context an element may represent a complete processing unit and/or a resource, with its associated controller.

There are different bus sharing methods which can satisfy the above conditions. Here, only the following two will be considered:

(1) Time Shared Bus

(2) Line Multiplexed Bus

Large computer networks communicating over long physical distances may use frequency division multiplexing as a form of line sharing scheme. This method is not applicable to this discussion as the cost and complexity of the modem may far exceed the cost and complexity of a given element in the system.

In any time sharing scheme, there is generally a single resource which is being shared among the different elements of the system, on the basis of a time slot assignment. The two basic time sharing protocols are:

(a) Synchronous - where each element is given a fixed time slot during which it has the sole use of the resource. If the element has no need for the resource, its time slot is wasted.

(b) Asynchronous - Only the element that has a need requests the use of the resource. When a request is granted, it can be either for the duration of the message or for only a portion of the message. Note that a more elaborate control is required for implementing this option, as some control information must be sent in order to identify message originator and message length.

The time-shared bus, as it uses a single resource, is relatively inexpensive, but as the number of elements increases, so does the complexity of the control scheme. The time shared bus has two major drawbacks:

(i) contention increases rapidly with the number of elements, and
(ii) since it is a single resource system, it is very susceptible to failure.

In a line multiplexed bus, separate physical connection between any two elements is provided via a switching system. Networks utilizing this scheme have high throughput as they can support simultaneous, mutually exclusive transfers. The most complex part of the communication system is the switching unit which provides the required connections. The major drawback of this scheme is the cost and the susceptibility of the switching system to failure.

In practice, there are a large number of busing schemes which are various combinations of time and line sharing. Further details about these schemes can be found in references [WEIT 80],[ANDE 75],[THUR 72].

4.3 THE SEGMENTED BUS CONCEPT

Busing schemes are either customized or general in nature. In both cases, the motivation behind the various organizations is to find a structure which can be easily implemented, be cost effective, and optimize system performance.

Here, a method of upgrading a standard single bus to accommodate simultaneous, mutually exclusive communication with minimal change to a standard bus, is presented. The principle of the segmented bus is that two communicating elements only require that portion of the bus physically between them. This will permit the remaining sections of the bus to be used for other communication.

Freeing the "unused" sections of the bus is accomplished by inserting bidirectional switches between adjacent elements on the bus. By using these switches, a communicating pair of elements can isolate themselves from the rest of the bus. The ability to segment the bus into separate sections led to the term "Segmented Bus" [THOM 75].

The organization of an open ended segmented bus (OESB) is shown in Figure 4.1.

During communication, by using the bidirectional switches, the bus can be effectively divided into a maximum of N/2 segments, (N = Even). This maximum value occurs only if all requests are between adjacent elements and are mutually exclusive. Note that in this context a link is defined as the direct physical distance between two elements, whereas a path or segment may consist of one or more links. The OESB does not provide maximum resource utilization. For example, if E_1 has to communicate with E_N, granting the use of the bus to E_1 causes the total length of the bus to be occupied, and no other communication can take place.

(The notation used for identifying each of the elements on the segmented bus also signifies their relative position with respect to each other).

If the physical distances are not large, by connecting both ends of the bus via a bidirectional switch, one can obtain better resource utilization.

The new organization obtained is called the Closed Loop Segmented Bus (CLSB) and is shown in Figure 4.2. In this organization each element controls the nearest switch to its clockwise (CW)direction. Signalling information is transmitted and received via two dedicated signalling lines connecting each two adjacent elements.

In the case of the segmented bus, it makes sense to define distance between two elements as the number of intermediate nodes. The distance between element i and element j for OESB is given by $|i-j|-1$ and for CLSB by min $[(\pm|i-j|)MOD_n-1]$.

Although the segmented bus does not guarantee conflict free communication, it does reduce the total number of possible conflicts in a time-shared bus. Requests for communication can be checked for conflicts by comparing their source and destination specifications. Consider two communication pairs P_1 (ij) and P_2 (kl) where $i<j$, and $k<l$. Without loss of generality, it can be assumed that $i<k$. A conflict will occur in the OESB when $k<j$ and in the CLSB when $k<j<l$. The closed bound for the CLSB implies that it is less subject to conflicts than the OESB.

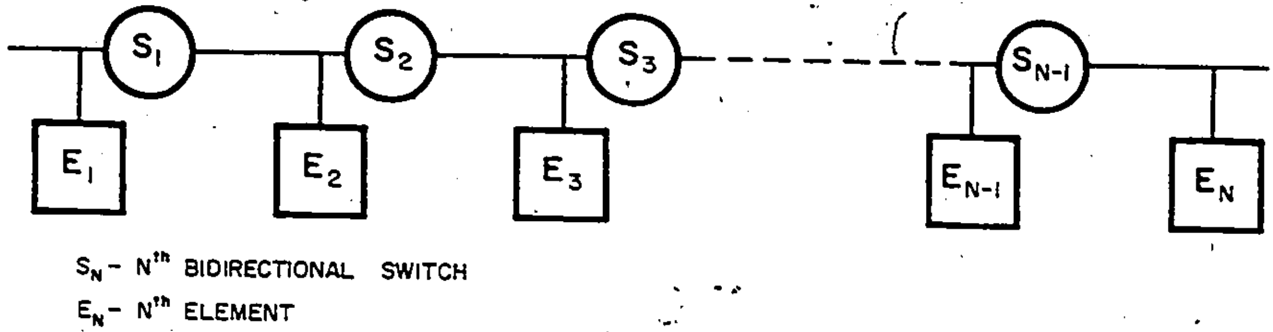


FIGURE 4.1 - OPEN ENDED SEGMENTED BUS

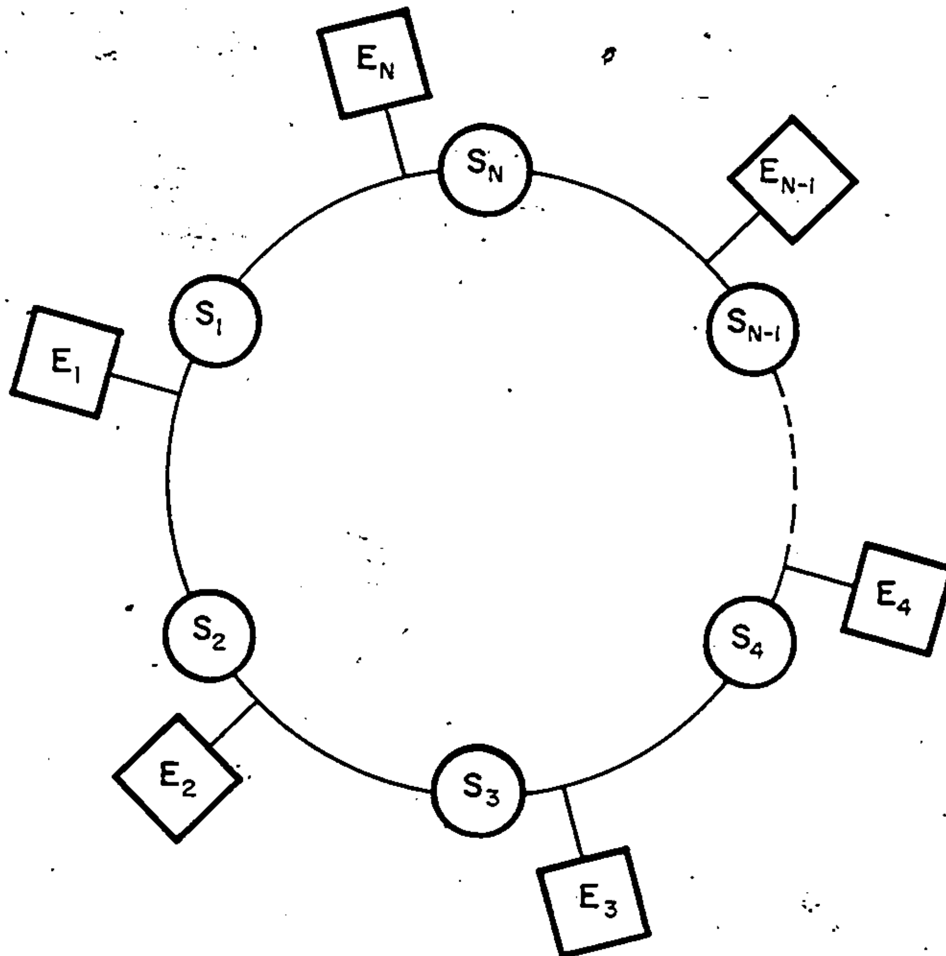


FIGURE 4.2 - CLOSED LOOP SEGMENTED BUS

4.4 SEGMENTED BUS ARCHITECTURES

The various control functions associated with the segmented bus can be done either centrally or in a distributed fashion. Aside from the physical control of the switches, to allow for controlled communication on the segmented bus, the following basic control information must be provided with each transaction:

(a) Request initiation.

(b) Source and destination codes.

(c) Start of transmission.

(d) End of transmission.

In the following, two segmented bus architectures will be described: a distributed control segmented bus (DCSB) architecture and a centrally controlled segmented bus (CCSB) architecture. For each architecture, a possible communication protocol and some implementation aspects will be given.

4.4.1 DISTRIBUTED CONTROL SEGMENTED BUS ARCHITECTURE

In the proposed distributed control segmented bus architecture, it is assumed that each element is responsible for all control functions associated with interprocessor communication on the segmented bus. Also, each element has a bus interface which is a hardwired logic circuit which includes some customized portions for individual elements. This establishes a very rigid, but fast interprocessor communication

procedure, as all decisions are made by hardware. As will be discussed in the implementation aspects section, some intelligence may be added to each interface to enhance the performance of this architecture and to make it more flexible.

(a) COMMUNICATION PROTOCOL FOR THE DISTRIBUTED CONTROL SEGMENTED BUS ARCHITECTURE

A possible asynchronous communication protocol for the DCSB is presented below. Note that in this context, the term asynchronous is used to specify that requests can be initiated at any time.

(1) Initially all the bidirectional switches are set to provide a path in one direction, say CW.

(2) A requesting element first verifies whether it is part of a communication path, by checking the closest switch status to its CW direction.

(3) If the switch is being used, it holds its request for a fixed time period.

(4) If the switch is not being used, the requesting element signals its neighbour to the counter-clockwise (CCW) direction to open both of its bidirectional switches. This avoids contention from the CCW elements.

(5) The requesting element issues the destination address on the bus.

(6) If the request reaches the destination element, the code is recognized and acknowledged via the signalling lines. The destination element signals all elements along the path, and the source, to maintain the line in the proper direction. In order to complete the path isolation from the rest of the bus, the destination element also opens both of its bidirectional switches in the CW direction.

(7) If the requesting element does not receive an acknowledgement, it signals its neighbour to the CCW direction to reconnect to the bus, and hold its request for a fixed time period.

(8) If the requesting element is to receive information, it transmits its request instead of data, but does not send end of transmission signal (this maintains the communication path already established).

(9) The destination element, via the signalling lines, requests all elements along the path to turn the line around, and transmits data as per step 10.

(10) The requesting element, upon receiving an acknowledgement, if it is a sender, sends the following information:

(i) Start of message.

(ii) Data.

(iii) End of message.

(11) Upon end of message information the proper signals are passed for each element along the communication path to resume the CW connection.

(12) While the path is being used, elements which are not part of it can establish other communication paths in the CW direction.

The above protocol is asynchronous protocol and therefore can support request only in one direction at any given time. This leads to an inefficient bus utilization since one may not always establish the shortest possible path. To overcome this limitation, one can synchronize the requests so that the protocol will support requests in both directions. In the synchronous protocol, to establish additional communication paths in the CCW direction, the above described steps are executed in a complementary fashion. The synchronous protocol is best suited for short, well-defined message lengths.

(b) IMPLEMENTATION ASPECTS

The utilization of hardwired logic to perform all the control aspects of an DCSB has the following drawbacks:

(i) Once a communication path has been established, it becomes difficult to interrupt. This makes the communication paths very rigid.

(ii) In this organization, it is hard to include priority schemes. One way would be to delay the low priority requests.

(iii) In the case of a line break or a complete switch failure, the part of the bus between source and destination may be out of service since end of message information was not received. This could be overcome by including length of message information and a time out procedure in each of the elements. This requires additional hardware but improves reliability, as failures are localized to a single link.

(iv) The DCSB does not necessarily provide the shortest possible path, since paths are being established unidirectionally. Also, the bus is being utilized inefficiently since intermediate elements cannot communicate and have to wait until the bus is free.

(v) As a direct result of (iv) above, the DCSB does not always provide the maximum number of connections.

Some of the drawbacks of the above simple distributed scheme can be overcome by adding intelligence to each interface. This can be done by either using a dedicated processor for each interface, or by time sharing the processor of the element.

The use of a dedicated processor for each interface may not be cost effective if one or both of the following conditions exist:

(a) The communication activities for an element are very minimal.

(b) The bus interface complexity is on the same level or exceeds that of the element itself.

The use of the element processing power may prove to be costly if the communication activities consume too much of the processor time.

It should be noted that the DCSB is a probabilistic communication system. An extensive study of the probabilistic nature of the DCSB is given in reference [DAVI 78].

4.4.2 CENTRALLY CONTROLLED SEGMENTED BUS ARCHITECTURE

In the case of short, physical distances between elements, a centralized control may be more appropriate. As opposed to the DCSB, in a centrally controlled segmented bus (CCSB) all the interconnections are controlled from a central location. The basic CCSB architecture consists of two layers in the form of rings: The outer ring is occupied by the data layer whereas the inner ring is occupied by the control layer. The data layer consists of a closed loop segmented data bus. The various elements, excluding the bus controller, are connected to the bus via bidirectional switches. Each element has one set of bidirectional switches in its physical space.

The control layer consists of a closed loop control bus and some dedicated control lines. The various elements are attached to the control bus via individual control interface (CI) units. The bus controller has its own dedicated interface card which is connected both to the control bus and to all the control interface units in the system. A simple block diagram of the CCSB architecture is shown in Figure 4.3

(a) THE FUNCTIONS OF THE CENTRAL BUS CONTROLLER

The most important element of the CCSB architecture is the central bus controller which performs all the control functions associated with the interprocesser communications in the system. The major responsibilities of the central bus controller are in the areas of: communication, arbitration, allocation, diagnostics and general chores. The functions associated with each area will be described next.

COMMUNICATION

(a) Accepts requests for communication from all the elements in the system.

(b) For fast messages (under 12 bytes) or in the case of heavy traffic and/or link failure, acts as mail delivery service.

(c) In the case of element and/or link failure, advises the other elements in the system directly affected to switch to their stand alone routines.

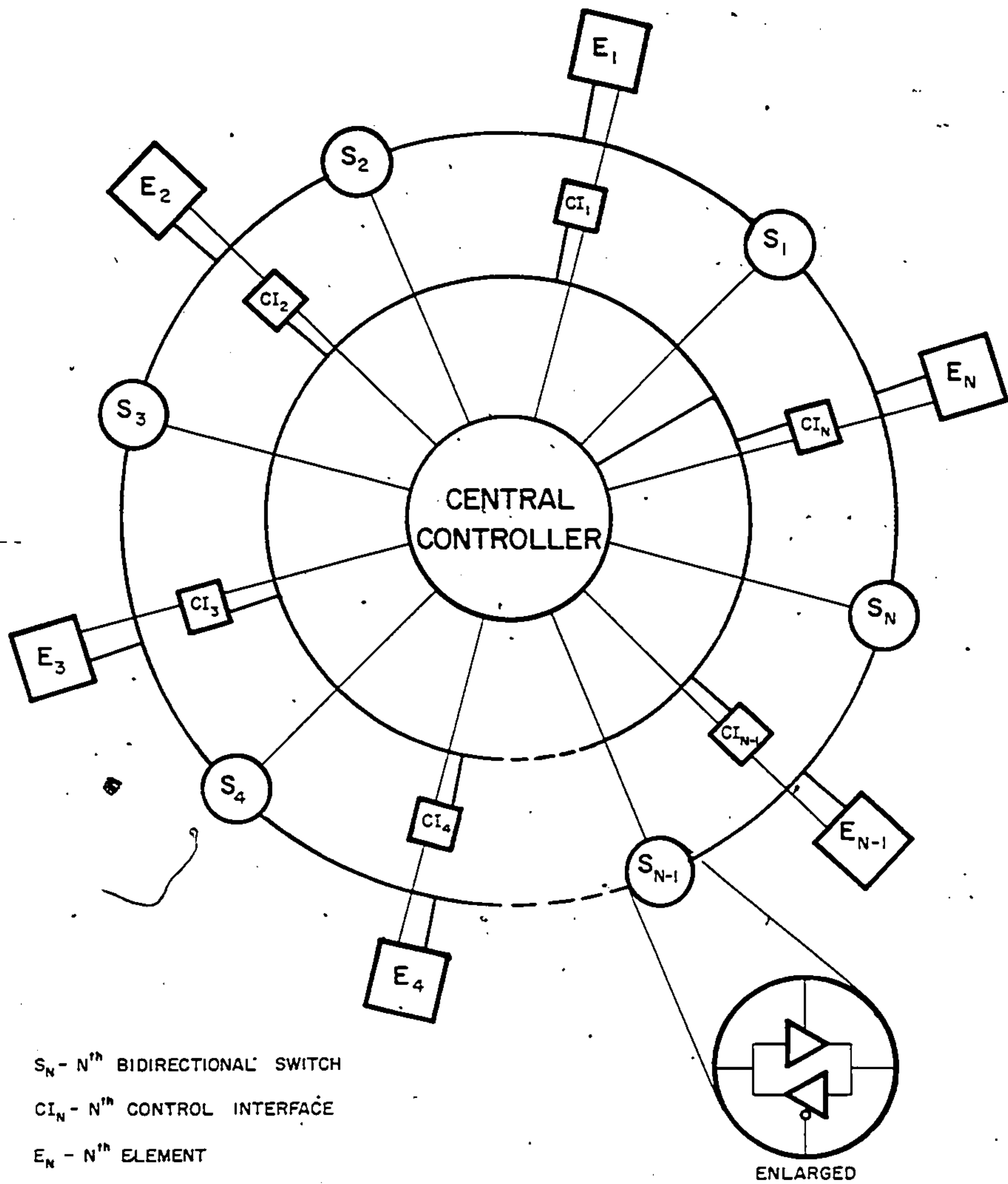


FIGURE 4.3 - CENTRALLY CONTROLLED SEGMENTED BUS

ARBITRATION

(a) Processes new requests, taking into account their software settable and hardware communication priorities.
(b) Rearranges existing paths (if possible) to accommodate new requests.
(c) Finds the shortest path from source to destination.
(d) Maintains and supervises system communication access control by checking request validity (A request for access to a classified data base may be rejected).

ALLOCATION

(a) Assigns communication paths by physically controlling the bidirectional switch control lines.

DIAGNOSTICS

(a) Utilizes a time-out option on every communication link for detecting hardware failure.
(b) Checks unused communication paths to verify that they are operational.

GENERAL CHORES

(a) Keeps a real-time clock information and passes it upon request to other elements in the system.
(b) Gathers statistical data about bus controller and system operation.

Due to the large and diverse number of functions the bus controller has to perform, and the fact that it is done by a single entity, makes the bus controller the most critical element in the system. This in turn makes the CCSB architecture vulnerable to failures as a failure of the bus controller may lead to complete system failure. To reduce the probability of major system failure, the central bus controller must be designed and implemented in the most reliable fashion possible. A conceptual design of the bus controller as a multi-microcomputer system is described in Section 4.5.

(b) COMMUNICATION PROTOCOL FOR THE CENTRALLY CONTROLLED SEGMENTED BUS ARCHITECTURE

A possible communication protocol for the CCSB architecture is presented below in a point form.

(1) Any element may issue a request via a dedicated request line through the CI. Prior to this, the requesting element places in the control interface the following control information:

(i) Source code.

(ii) Destination code.

(iii) Maximum length of message.

(iv) Type of transaction - read/write.

(2) The central controller reads the information in the control interface of the highest priority element and clears its flag. This allows for simultaneous requests.

(3) The central controller continues to read, in order of their priority, all pending requests.

(4) Meanwhile, the central controller processes stored requests and decides on proper routing schemes.

(5) Once the controller determines the route, it signals the source and destination elements and sets the proper switches on the segmented data bus. It also updates the routing tables.

(6) The central controller sets a byte counter corresponding to the maximum message length to protect against:

(i) Switch/link/element failure.

(ii) Inefficient use of the bus.

(7) If a high priority element requests use of an occupied resource or bus, the controller may execute one of the following:

(i) Revoke the existing communication links and grant the link to the high priority element.

(ii) If the timer indicates that the message is about to be completed soon, the controller may wait for the end of the message and then grant use of the bus to the high priority element.

(iii) The controller may act as a mail delivery service for the high priority element by accepting the message and passing it to/from the proper destination.

(c) IMPLEMENTATION ASPECTS

The three main entities of the CCSB architecture are the data and control buses, the central bus controller and its interface to the system, and the control interface of individual elements. In the following, some of the most important design and implementation aspects as related to each of the above entities will be described.

DATA AND CONTROL BUSES

There are two main buses in the CCSB architecture: a data bus which is dedicated entirely to transfer of data, and a control bus which may carry both control information and data. The data bus is a closed loop segmented bus, where the bidirectional-switches of the segmented bus could be a set of two tristate buffers connected in a back to back fashion. The normal switch position is the off position. Whenever there is a need for communication, the proper switches are turned on. The bidirectional switches are controlled, via two dedicated control lines, directly by the central bus controller. The two control lines are needed in order to support three possible conditions of the Bus: CW path, CCW path and element isolation. The various states of the control lines with the corresponding data bus status is presented in the form of a truth table shown in Table 4.1.

CONTROL LINE OF CW SWITCH	CONTROL LINE OF CCW SWITCH	SWITCH STATUS
0	0	CCW DATA FLOW
0	1	ISOLATED
1	0	NOT ALLOWED
1	1	CW DATA FLOW

TABLE 4.1 – BIDIRECTIONAL DATA SWITCH STATUS TRUTH TABLE.

The various states of the two control lines can be used for detecting improper switch setting. Thus before any data exchange, the communicating elements verify that the switches are set properly by checking the status of the appropriate switches.

The control bus is also a closed loop bus which supports bidirectional data flow. By tying together both ends of an open-ended bus, one obtains a closed loop bus which provides an alternate path in the event of an open circuit. The control bus is utilized primarily for transferring control information. The secondary function of the control bus is for mail delivery service, which is performed either in the cases of high priority short messages, broadcasting of a message, or in the event of a partial failure of the data bus.

CENTRAL BUS CONTROLLER AND ITS INTERFACE

The various functions executed by the bus controller were described previously. A conceptual design of a multi-microcomputer implementation of the bus controller will be discussed in Section 4.5. Here, only the aspects of element request identification and the access into a memory of a particular element control interface are considered. These aspects are very important since the initiation of asynchronous requests by multiple elements in a system may lead to a processing bottleneck. This in turn may also lengthen the response time to a request. In order to overcome these problems, the bus controller is

designed to have a fast request identification procedure implemented completely in hardware. However, in order to make the bus controller response to various requests as flexible as possible, some software control is also introduced into the design.

In the proposed design, each control interface has two dedicated control lines which are connected directly to the bus controller. The first control line is the request line which is directed from the control interface towards the bus controller. The request line is hardwired, based upon a predetermined fixed priority, to a priority encoder latch, whose outputs are expanded by a decoder. The second control line is the clear request line which is one of the outputs of the decoder, and is directed from the bus controller towards the element control interface.

The circuit operates in the following manner. The status of the various request lines are continuously sampled. As soon as a request is initiated, the clock is inhibited and the appropriate clear request line changes its state to a logical low. In the event of multiple simultaneous requests, the highest priority request is latched. Upon detection of the request initiation, the bus controller is interrupted and subsequently responds by reading the proper information from the memory selected by the clear request line. Upon completion of the read cycle, the clock is re-enabled so as new requests may be loaded into the priority encoder.

In order to allow the bus controller to access the various memories in different control interfaces on its own initiative, it can force one or more clear request lines into a particular state. A block diagram of the hardwired priority encoder and the interface handshake control is shown in Figure 4.4. The proposed design minimizes the requesting element indentification time thus reducing the overall response time to a request. However, in order to make the bus controller response more flexible, in addition to the hardwired priority, each element is assigned a software settable priority.

The software settable priority may be different from the hardwired priority and always takes precedence over it. The various priorities are stored in the form of a look-up table which is managed by the bus controller. In general, the hardwired priority is utilized primarily for the process of request identification. The software settable priority on the other hand is used only for the arbitration process when deciding which elements are to be granted communication paths. Finally, it should be noted that the utilization of a software settable priority facilitates the fine tuning of the overall system performance.

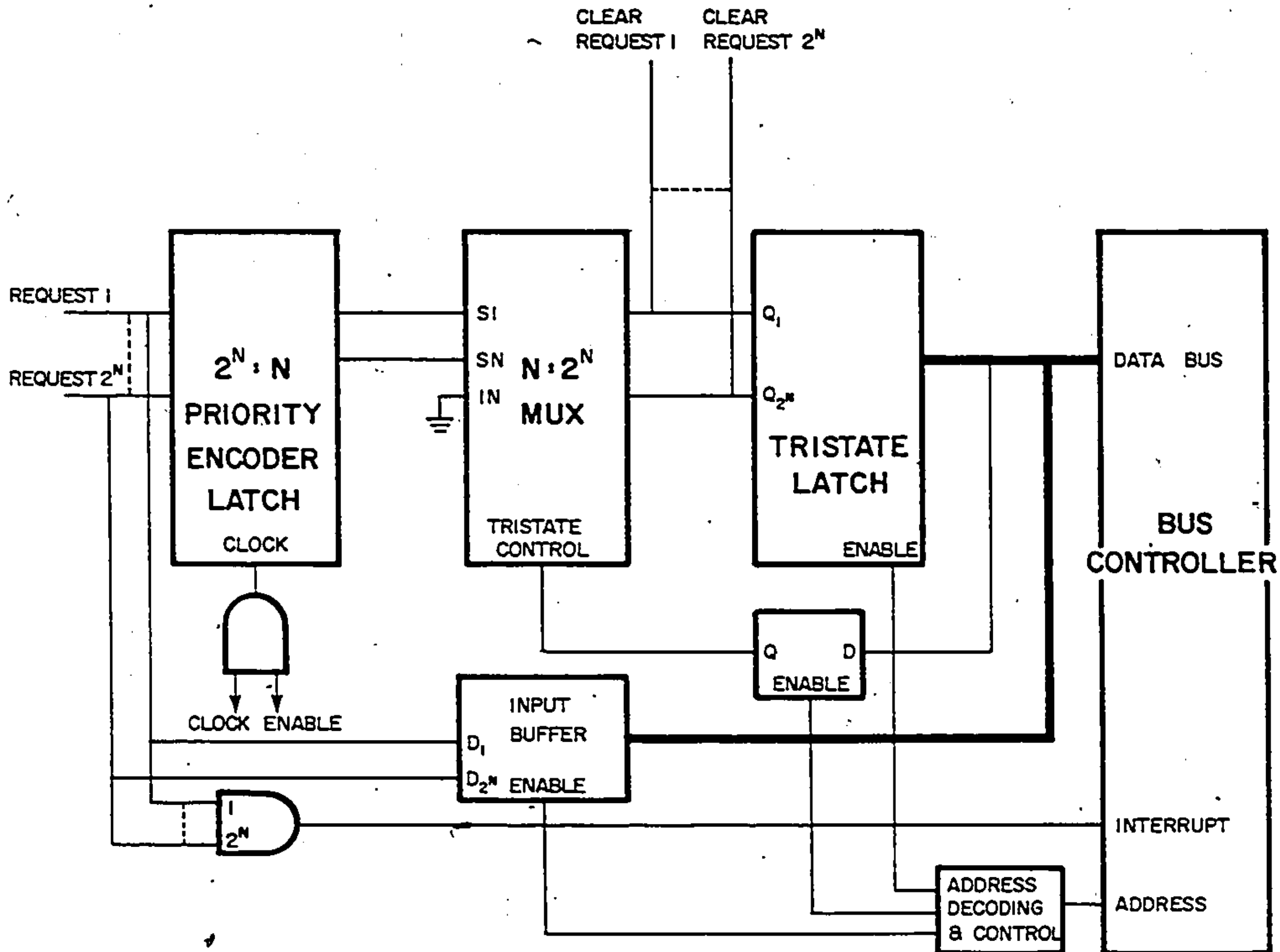


FIGURE 4.4- HARDWIRED PRIORITY ENCODER & INTERFACE HANDSHAKE CONTROL

CONTROL INTERFACE

The transfer of control information between each element and the central bus controller is done via a control interface whose block diagram is shown in Figure 4.5. The central element of this interface is a 16 byte read/write memory. An element requesting service places the proper control information, as defined in the protocol, at locations 0 to 3 of this memory. The rest of this memory is used as a "mail box" when the controller acts as a mail delivery service or when suspending communication paths.

Under normal conditions, the central controller, upon detecting a request, is given automatic access to the memory in the interface circuit of the requesting elements. The controller maintains control until it completes reading the control information from locations 0 to 3. At the end of the read cycle, control is returned to the element so that it can initiate other requests.

Whenever the bus controller wishes to deliver mail to an element or to suspend established communication paths, it has to perform the following steps:

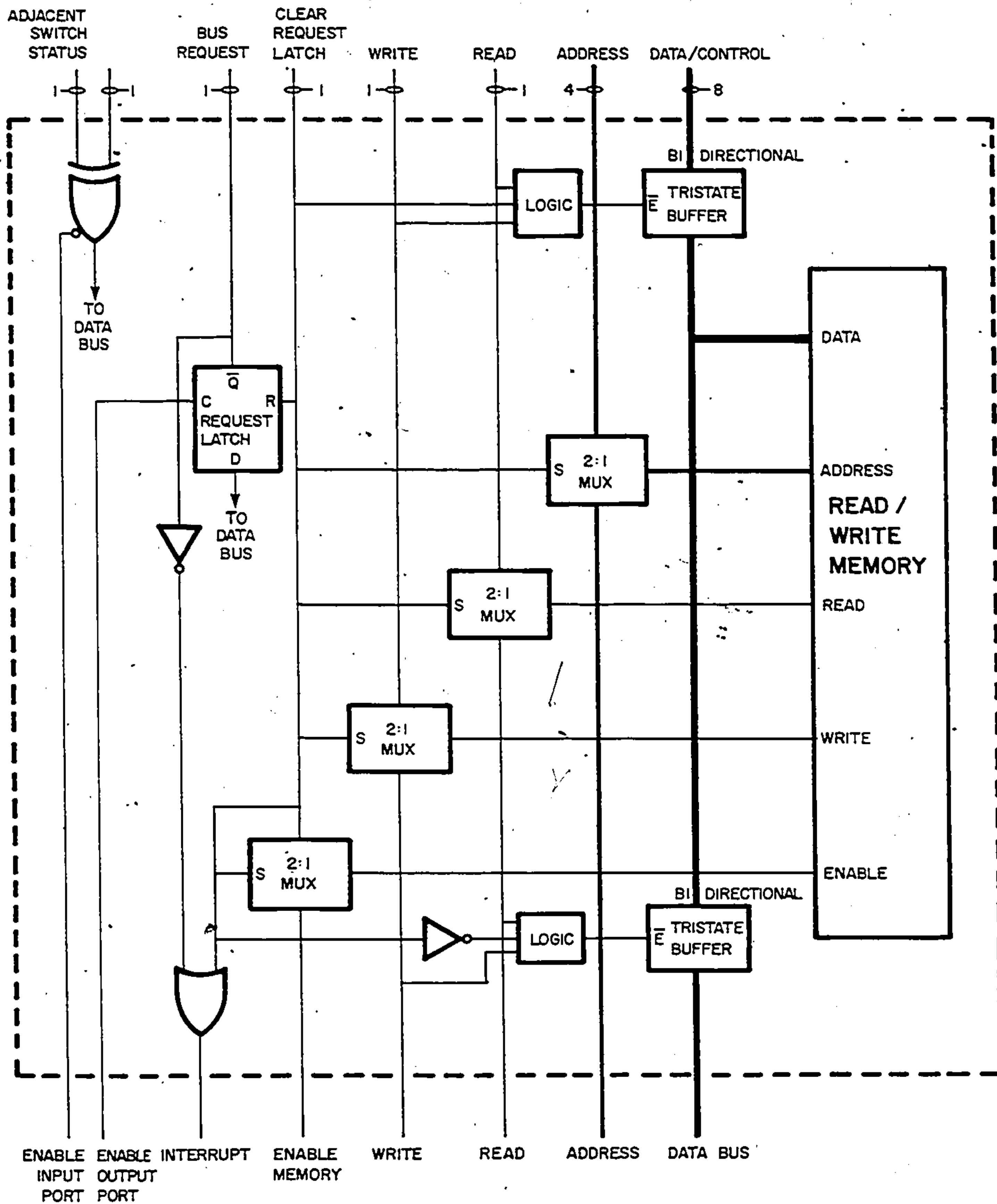


FIGURE 4.5 - CONTROL INTERFACE BLOCK DIAGRAM

(1) Inhibit the clock to the priority encoder latch.

(2) Record the current status of any request lines for those elements involved in the transaction. The number of elements involved in one transaction may range from one for the case of delivering a fast message, to the entire set of elements as in the case of a message broadcasting.

(3) Tristate all clear request lines.

(4) Force to logical zero a clear request line corresponding to one of the elements involved in the transaction. This clears the request latch and permits the bus controller to access the memory of the appropriate element.

(5) If a request was initiated, the bus controller reads the request information from locations 0 to 3.

(6) If the request was not initiated or after executing Step 5, the bus controller writes the proper control information (read, write, type of information e.g. suspension of communication) in locations 4 and 5.

(7) When delivering mail, the bus controller writes into or reads from the appropriate memory locations. If performing suspension of communication, it skips this step.

(8) Enable momentarily the clock to the priority encoder latch.

(9) Record for the remaining elements involved in the transaction, the status of their request lines to verify if new requests were initiated. This is necessary in order to maintain control information integrity and to avoid losing requests, as when accessing the memory the request latch is cleared.

(10) Relinquish control over the memory by re-enabling the clear request lines.

(11) Repeat steps 3 to 10 inclusive until all elements involved in the transaction are reached and are informed. This is done to maintain data integrity and control information integrity.

If the controller is to receive "mail" from an element, a regular request is transmitted. The only difference is that after reading the control information (which indicates this) the controller continues to read data from locations 6 to 15. With the proposed implementation, the controller has the ability to broadcast a message simultaneously to all or selective groups of elements.

Each time the bus controller accesses the control interface memory, an interrupt is generated, which indicates to the element that a transaction in which it is involved took place. Following an interrupt, the element reads the control information in positions 4 and 5. If it is to receive mail, it reads the data from locations 6 to 15. If it is to transmit data, it places

the requested data in locations 6 to 15 and uses the normal request procedure to indicate this to the central controller. If the control information indicated suspension of a communication path, the element takes the proper action .

In order to reduce the number of dedicated direct control lines, the clear request line has a dual function. Thus, the memory access selection and the element interrupt can be represented in the form of a truth table as shown in Table 4.2. From this truth table, one can observe that Select = Clear Request where as Interrupt = Request + Clear Request.

REQUEST LINE	CLEAR REQUEST-LATCH LINE	MEMORY ACCESS SELECTION		ELEMENT INTERRUPT	
0	0	CONTROLLER	0	NO	1
0	1	ELEMENT	1	NO	1
1	0	CONTROLLER	0	YES	0
1	1	ELEMENT	1	NO	1

TABLE 4.2 – MEMORY ACCESS SELECTION AND ELEMENT INTERRUPT TRUTH TABLE.

Each control interface also includes a simple error detection logic for checking adjacent switch status. It consists of an exclusive OR gate whose inputs are the switch control lines. The element, prior to data transaction, may read the output of the gate to find out whether the switch is in isolation mode. However, using this simple circuit, the element cannot find out whether

the switch is in the forbidden state and/or the actual flow direction of data allowed by the switch. In order to get the complete picture, the element must examine the control lines of both the adjacent switch as well as the switch located in its physical space, prior to data transaction. In the event that the switch setting is not in agreement with the requested setting, it can inform the bus controller to check the switch status. This in effect also provides an indirect way of checking the operation of the bus controller.

4.5 CONCEPTUAL DESIGN OF THE BUS CONTROLLER

In general, for continuous operation of any centrally controlled system, the central control unit must be designed to have fail soft or fail safe capability. Furthermore, for the CCSB architecture, the bus controller needs to execute a large and diverse number of functions. Thus, to achieve the desired response characteristics, the bus controller must be able to support concurrent operation. Therefore, a conceptual design of the bus controller as a multi-microcomputer organization is presented. In the following, the partitioning of the main processes of the bus controller as well as a first rough task-processor allocation, will be described.

4.5.1 PARTITIONING OF BUS CONTROLLER MAIN PROCESSES INTO TASKS

The essential functions of the bus controller can be grouped into the following four processes: communication, arbitration, allocation, and diagnostics. For a given transaction, the relation among these four processes is sequential, with limited

degree of parallelism within the processes themselves. However, with respect to multiple requests, even though they are accepted one at a time, they are processed in a concurrent fashion by pipelining through the processes in the system. With respect to the complete bus controller operation, there is a fifth process of general chores as well as some aspects of the diagnostics process, as shown in Figure 4.6. These can be executed in parallel with the other four processes because they include tasks which do not directly affect the main bus controller response to communication requests. For example, the general chores process contains tasks such as a real-time clock task and a statistical data gathering task which will not be considered any further.

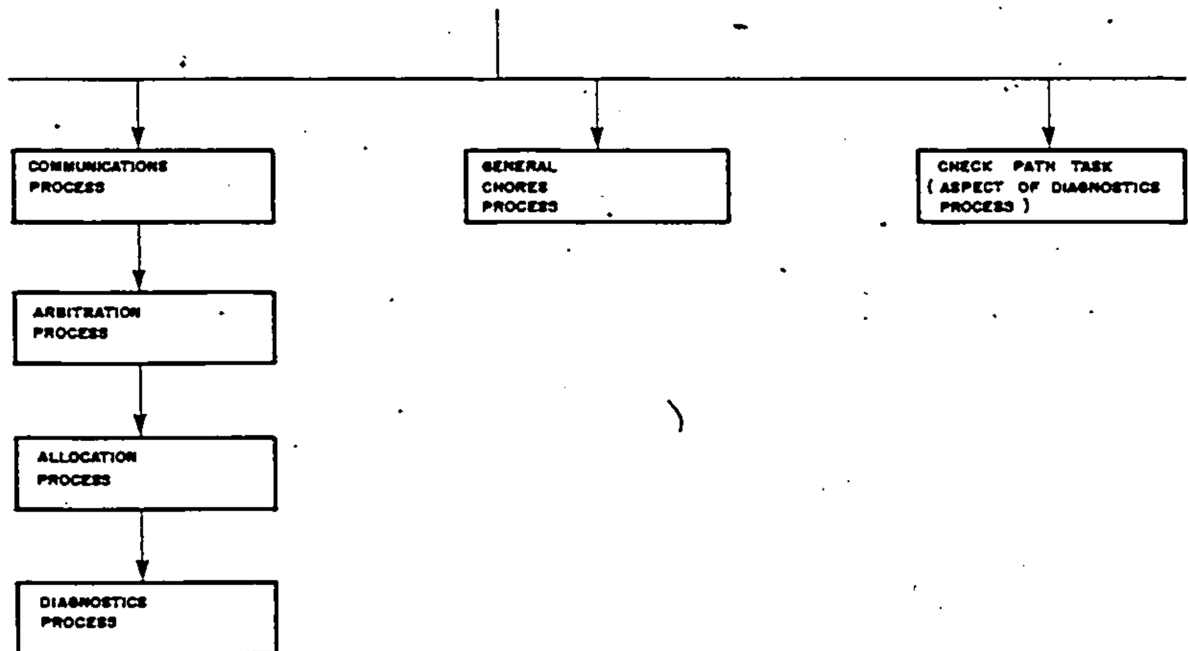


FIGURE 4·6 – BUS CONTROLLER PROCESS SEQUENCING RELATIONS.

(a) THE COMMUNICATION PROCESS

The communication process is responsible for all information transfers between the bus controller and the system. These transfers can be initiated as a result of internal requests from the other processes or external requests from the elements. Internal requests deal mainly with control messages sent to the elements following a transaction, in order to keep them informed about the bus controller operation. On the other hand, external requests are generated mainly when there is a need for communication paths. Internal requests are recognized through a polling procedure and always take precedence over external requests which are identified through an interrupt procedure. The communication process can be subdivided into the following four tasks:

(1) Request Handler (RH)
(2) Read Control Information (RCI)
(3) Read/Write Message (R/Wm)
(4) Request Analysis (RA)

The control flow of the communication process is described below and is shown graphically in Figure 4.7.

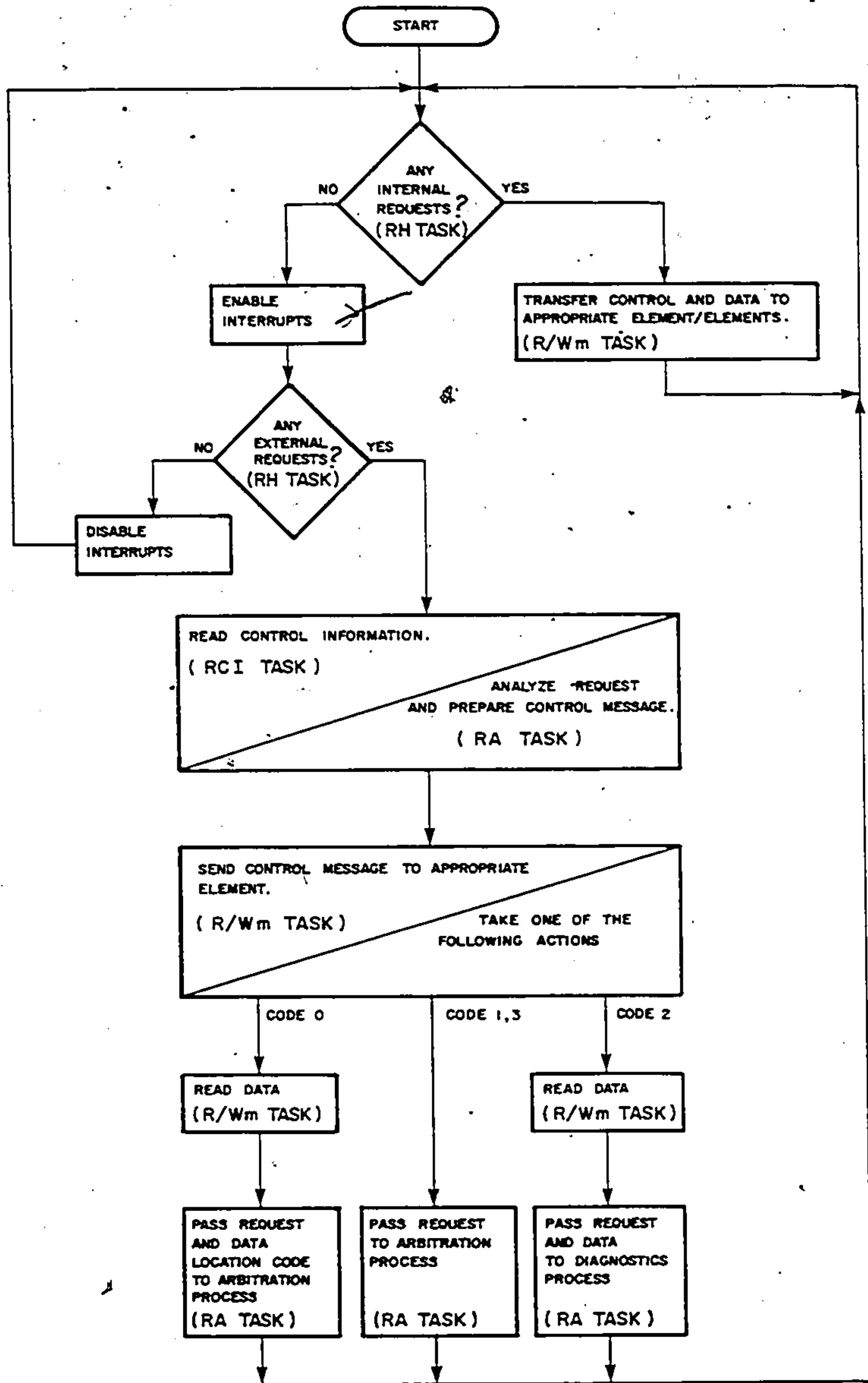


FIGURE 4.7 - CONTROL FLOW DIAGRAM OF THE COMMUNICATION PROCESS.

The RH task is a background task which is continuously looking for newly generated requests by polling the processes for internal requests, and responding to interrupts generated by the elements. The relative time spent polling each process is a function of its relative priority to the other processes, as well as proportional to the number of requests it generates. All internal requests are placed on a queue and are read by the R/Wm message task on a first come first serve basis. The types of internal control messages sent from each process to the communication process will be described when discussing each process.

In the case where there are no internal requests pending and/or on periodic time intervals, the RH task enables the interrupt structure so it can react to external requests. In response to an interrupt, the RCI task is activated which reads the data from locations 0 to 3 of the control interface memory. The control information is passed simultaneously to the RA task which examines the request and decides on the next step to be taken. The format of the control information is as follows:

Address 0 - contains information about

(i) Transaction types, (bits 0-1) where

Code 0 - requested data is ready in memory .

Code 1 - request for communication path

Code 2 - Diagnostic data is returned

Code 3 - Confirmation of transaction completion

(ii) Source identification codes (bits 2-5). Here for illustration purposes, it is assumed that only 16 elements are controlled by the bus controller.

(iii) Number of destinations involved in the transaction (bits 6-7) where

Code 0 - Select bus controller as destination

Code 1 - 1 destination

Code 2 - 2 destinations

Code 3 - 15 Destinations (broadcasting a message)

<u>Address 1</u> - contains information about the starting address (bits 0-3) and ending address (bits 4-7) of data for transaction code 0 and 2. For transaction code 1, this location specifies the estimated length of transaction in terms of data bytes, to a maximum of 256 bytes.

<u>Address 2</u> - contains destination addresses; (destination #1, bits 0-3) and (destination #2, bits 4-7).

<u>Address 3</u> - future expansion.

Note that for code 3, only the transaction type code and source specification is necessary.

At the conclusion of the analysis, an appropriate control message is sent to the elements involved via the R/Wm task. The message acknowledges the reception of the request and specifies the actions taken in response to the request. Actions taken can be one of the following:

(1) For transaction code 0, the R/Wm task reads the data from the appropriate memory locations and stores it at specified memory locations. The request is then passed to the arbitration process together with the code identification for the storage location of the data.

(2) For transaction code 1 and 3, the R/Wm task passes the request to the arbitration process.

(3) For transaction code 2, the R/Wm task reads the data from the appropriate memory locations and passes both data and request to the diagnostics process.

Following these activities the control is returned to the RH task which continues to poll the processes for newly generated requests.

(b) THE ARBITRATION PROCESS

The arbitration process is responsible for all control aspects associated with the granting of communication paths and consists of the following five tasks:

(i) Request Validation (RV)
(ii) Message Generator (MG)
(iii) Request Management (RM)
(iv) Path Identification (PI)
(v) Path Suspension (PS)

The control flow of the arbitration process is described below and is shown graphically in Figure 4.8.

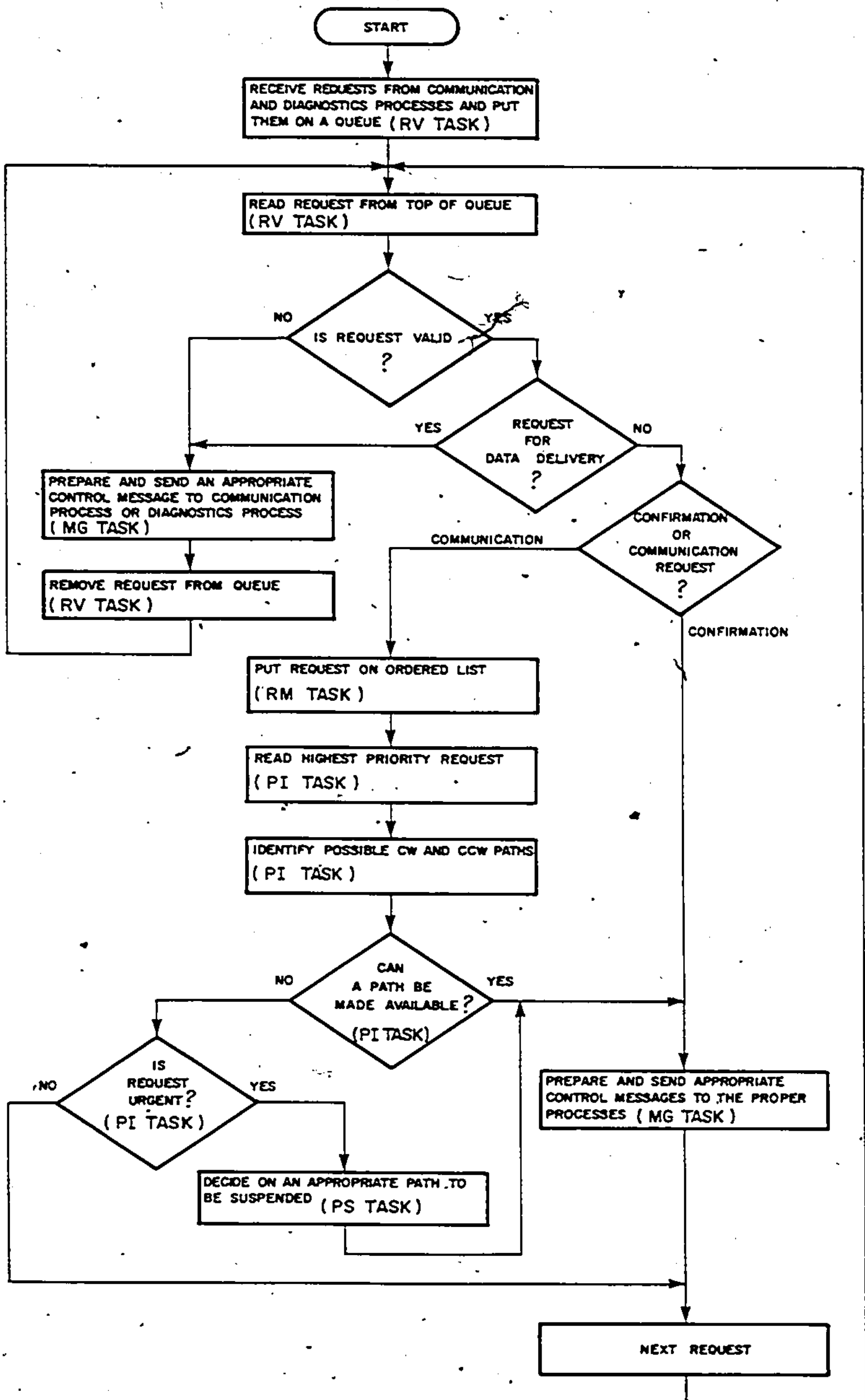


FIGURE 4.8 – CONTROL FLOW DIAGRAM OF THE ARBITRATION PROCESS.

All requests arriving at the arbitration process from the communication and diagnostics processes are received by the RV task which verifies that the requests are valid. Invalid requests, like a request to write into an output device, are rejected. An error message in the form of internal request is prepared by the MG task and is sent to the communication process for transfer to the appropriate element. If the request is for data delivery (either diagnostics data or data sent from another element) an appropriate internal request specifying the data storage identification code, is prepared by the MG task and sent to the communication process. A confirmation request, stating that the transaction is completed is sent, after validation, to the diagnostics process. Also, a request by the CHECK PATH task, after validation, is transferred to the communication process.

A valid communication request is passed to the RM task, which utilizes a look-up table of the element's software settable priority to insert the request in its proper location. The information stored on the ordered list is the source code, destination code/(s) and whether the request is urgent.

The PI task, reads the highest priority request and tries to find an available CW path, a CCW path, or if a path can be made available by rearranging existing paths. If a path cannot be found or be made available and the request is not urgent, the request is returned to the queue. However, if the request is urgent, it is passed to the PS task. It utilizes element priority information and data obtained from the diagnostics

process, about estimated remaining time for the various transactions, to decide on the most appropriate path to be suspended.

In the case of finding a path without the need for suspension, the MG task prepares appropriate control messages, which are sent to the communication process and allocation process. The communication process transfers these messages, which contain information about the changes to the communication path or check path information, to all involved elements. The allocation process uses this information to set the switches along a path into their proper position.

(c) THE ALLOCATION PROCESS

The allocation process is responsible for setting the switches in the proper mode and it consists of the following two tasks:

(i) Path Specification (PS_p)
(ii) Switch Control (SC)

The control flow of the allocation process is described below, but due to its simplicity no flow diagram is shown.

Requests are received from the arbitration process which consist of two bytes with the following format:

Byte 1 - contains information about:

(i) Whether the request is for setting a path, (bit 0=0), or for dismanteling a path, (bit 0=1).

(ii) Whether path direction is CW (bit 1=0) or CCW, (bit 1=1).

(iii) Source specification, (bits 2-5), with bits 6-7 retained for future consideration.

Byte 2 - contains the specifications of the further set destination along the path (bits 0-3) with bits 4-7 reserved for future expansion.

All requests are received by the PS_p task which puts them on a queue and processes them one at a time on a first come first serve basis. For each request, the PS_p task indicates the switch setting for each of the elements along the identified path. This information, after being specified for the first element on the path, is used by the SC task to physically set the switches to the proper mode. Upon completion of setting all switches for a specified path, a control message stating this fact is sent to the diagnostics process. It in turn, computes the transactions relative length and inserts it on the ordered list.

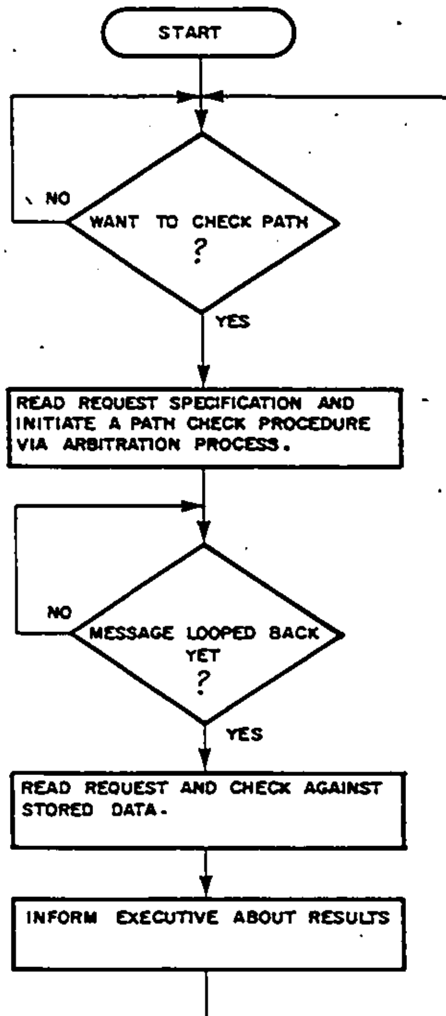
(d) THE DIAGNOSTICS PROCESS

The diagnostics process is responsible for verifying that the communication hardware and software are functioning properly. It consists of two distinct parts which can run in parallel: path diagnostics and communication diagnostics. The path diagnostics part consists of only the CHECK PATH task whose flow diagram is shown in Figure 4.9, whereas the communication diagnostics part consists of the following four tasks:

(i) Set-up (S-U)
(ii) Dismantle (DIS)
(iii) Transaction Length Timer (TLT)
(iv) Transaction Completion Timer (TCT)

The control flow of the diagnostics process is described below and is shown graphically for each task in Figure 4.10 and 4.11.

The check path requests are initiated by the system executive, either on a periodic basis or as a result of a complaint by an element. The requests are initiated one at a time and are received by the CHECK PATH task which sends a control message to the arbitration process. The control message also includes a known data pattern which is checked, upon looping back from element, against the original pattern for errors. The results of this check are sent to the executive in terms of a handshake signal and/or control message, which takes the appropriate control actions.



CHECK PATH TASK

FIGURE 4.9 - CONTROL FLOW DIAGRAM OF THE DIAGNOSTICS PROCESS IN TERMS OF TASKS.

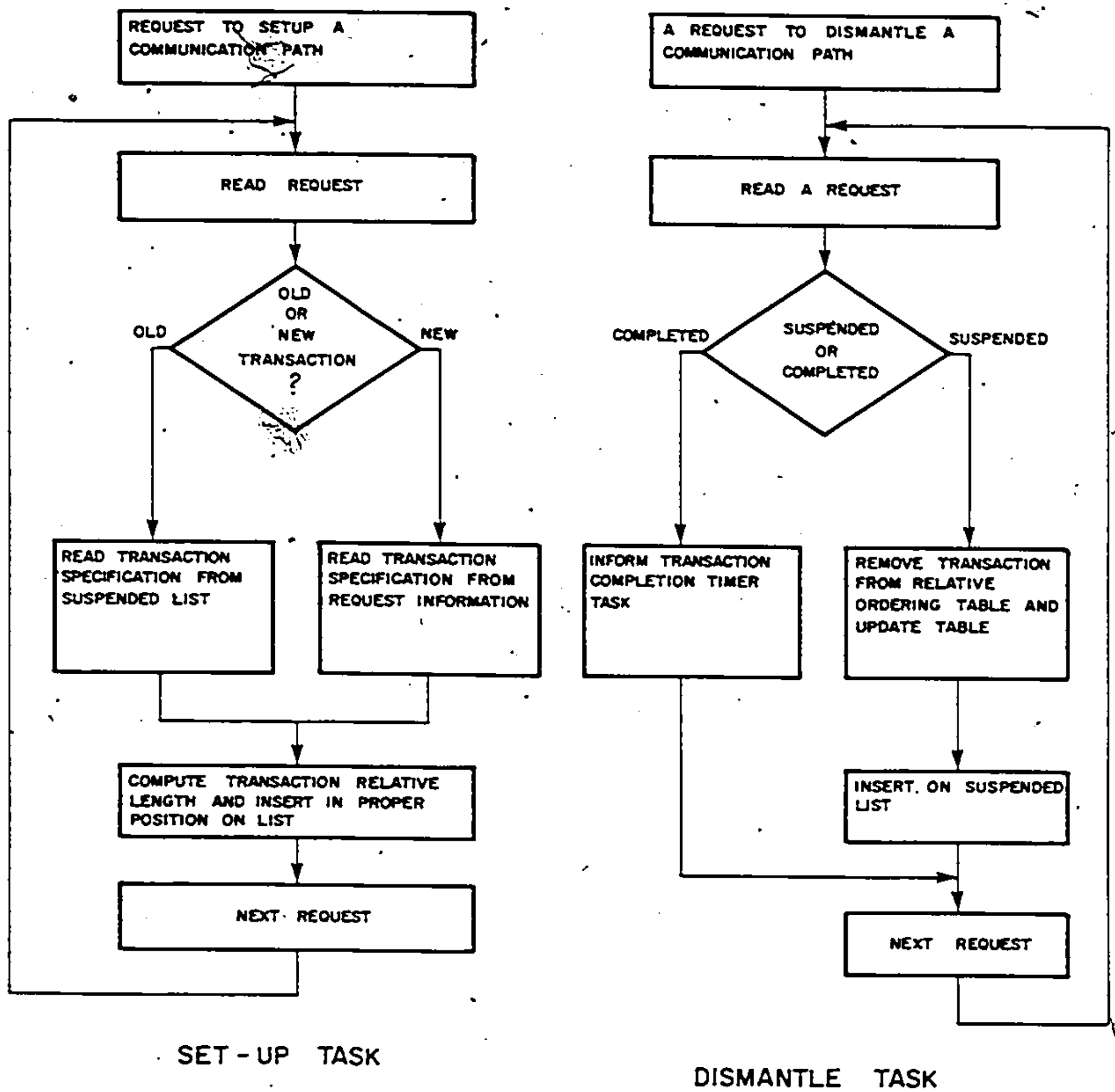
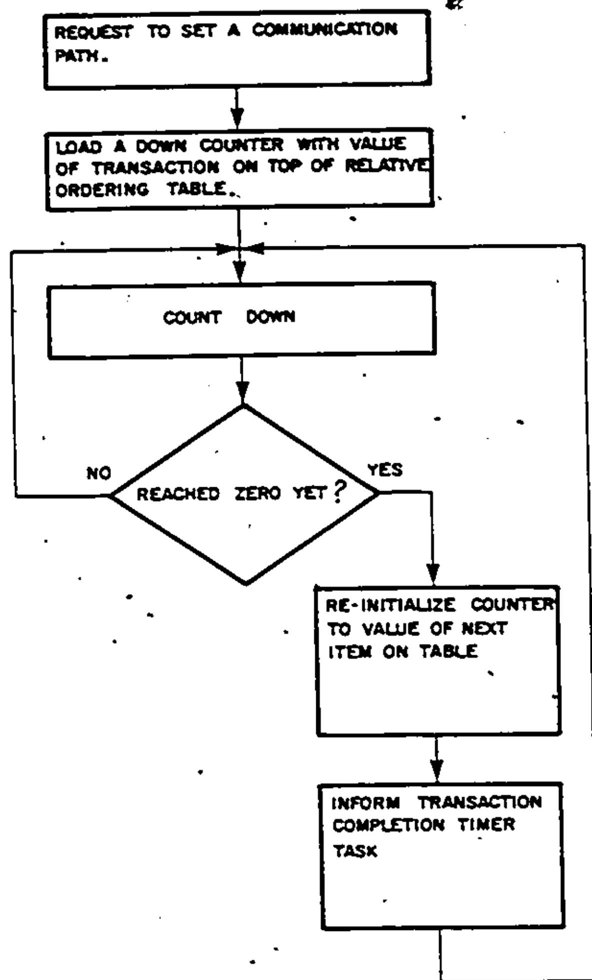
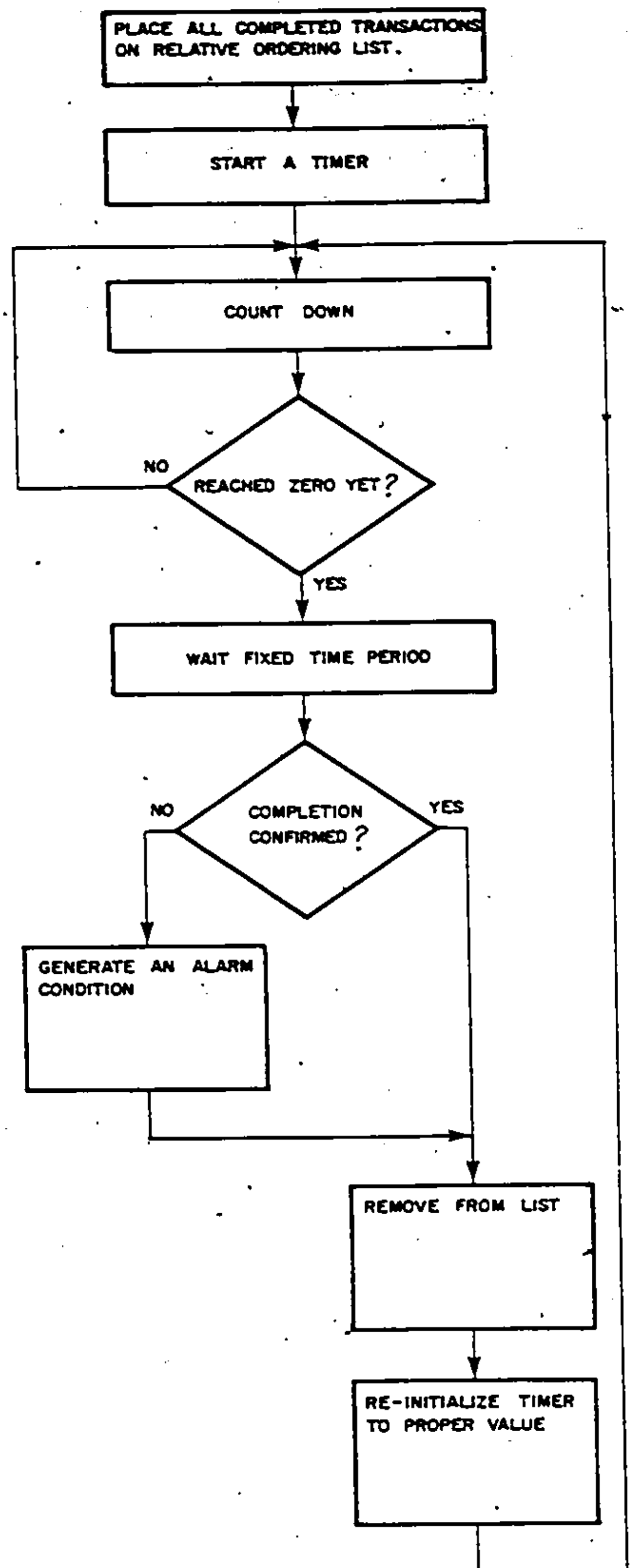


FIGURE 4.10 – CONTROL FLOW DIAGRAM OF THE DIAGNOSTICS PROCESS IN TERMS OF TASKS.



TRANSACTION LENGTH TIMER



TRANSACTION COMPLETION TIMER

FIGURE 4.11 – CONTROL FLOW DIAGRAM OF THE DIAGNOSTICS PROCESS IN TERMS OF TASKS.

All requests for communication paths, as received from the allocation process are directed either to the S-U task, which maintains a RELATIVE MESSAGE length list, or to the DIS task. The set-up requests are processed by setting a time-out counter through the TLT task, which continuously counts down the remaining time for transaction proportional to the message length. Requests for dismantling paths are processed by the DIS task by removing them from the active list and placing them on COMPLETED or SUSPENDED lists. When a request to set-up a suspended path is received, information from the suspended list is transferred into the relative message length list.

In order to check element operation a second timer is used by the TCT task. It in turn, awaits the completion message from the elements via the arbitration process. All completed transactions are placed on an RELATIVE ORDERING COMPLETED list. If confirmation about completion is not received from the element which requested the paths before the time-out period, an error message is generated.

4.5.2 TASK-PROCESSOR ALLOCATION, A ROUGH DRAFT

Referring to Figure 4.6, one can observe three parallel streams associated with the operation of the bus controller. From the point of view of task-processor allocation, each of these streams can be treated separately. In the case of the general chores process, a seperate processor can be dedicated to each of its tasks. Also, the check path task described with the diagnostics process, is assigned its own processor. The

task-processor allocation of the third path is more elaborate and must be given greater consideration. There are four processes associated with the third path, which have to be executed in a sequential manner for each communication request. Thus, the simplest allocation scheme is to use a single processor. However, multiple requests can be executed in concurrent fashion by pipelining them through the various processes if using multiple processors.

As the first step of task allocation one has to consider the relative execution times of each task. Table 4.3 lists the estimated execution times relative to the RCI task. The relative time for each task is based on its complexity relative to that of the RCI task.

If all the tasks specified in Table 4.3 are executed sequentially, it will take a maximum of 120 time units to complete one transaction. In practice not all tasks need be executed in response to a request thus total execution time is shorter. Allowing a 30% error in the estimated execution time of each task, yields a total execution time ranging from a minimum of 80 time units to a maximum of 160 time units.

The total execution time allocated for each task is calculated by multiplying the number of time units by the actual execution time of the RCI task. Using an intel 8085 microprocessor instruction set, one can write the corresponding code to read four memory locations and transfer the date into other four locations. Table 4.4 list this code along with the number of machine states required for the execution of each instruction.

TASK NAME	ESTIMATED RELATIVE TASK EXECUTION TIMES
REQUEST HANDLER	6
READ CONTROL INFORMATION	1
READ / WRITE MESSAGE	4
REQUEST ANALYSIS	9
REQUEST VALIDATION	10
MESSAGE GENERATOR	6
REQUEST MANAGEMENT	12
PATH IDENTIFICATION	15
PATH SUSPENSION	16
PATH SPECIFICATION	10
SWITCH CONTROL	3
SET - UP	10
DISMANTLE	8
TRANSACTION LENGTH TIMER	4
TRANSACTION COMPLETION TIMER	6
TOTAL # OF TIME UNITS	120

TABLE 4.3 – ESTIMATED RELATIVE TASK EXECUTION TIMES.

Using a clock rate of 3.072 MHz, the actual execution time for the code shown in Table 4.4 is about 25 ms. Thus, the maximum sequential execution time of the four processes is .4ms. This implies that theoretically one processor can handle up to about 250 requests per second. Whenever the bus controller has to handle more than 250 requests per second, one has to resort to a multiple processor system. However, the restrictions imposed by the system must be considered first.

(1) The various tasks, specified for each process, are all unique within the system. Thus each task can be used only within one process and should be controlled locally.

(2) Sequencing relations between the four processes are sequential. Thus to reduce intertask communication, each process is a self-contained unit having one or more processors allocated to it.

In line with the above, all intertask communications within each process are done via local memory and only interprocess communications are done via global memory. Concurrency in the operation of this path is achieved by pipelining the request processing within the system. Also, for better results, a second arbitration processing unit is used in parallel since it requires much longer execution times than the other processes. For reliability purposes, one can either duplicate each unit, or use one back-up processor which is capable of executing all processes.

LABEL	INSTRUCTION	MACHINE STATES
START :	LXI H, X	10
	LXI D, Y	10
	MVI B, 4	7
LOOP :	MOV A, M	7
	INX H	6
	XCHG	4
	MOV M, A	7
	INX H	6
	XCHG	4
	DCR B	4
	JNZ LOOP	10
	RET	
TOTAL # OF MACHINE STATES		75

TABLE 4.4 – READ CONTROL INFORMATION TASK CODE.

The various processing units can be controlled by an executive similar to the one described in Chapter 3 with the following characteristics:

(i) Each processing unit has its own task manager which contains the sequencing relations within the process in its code. It also may execute one of the tasks within the process and is fully responsible for the interprocess communication done via global memory.

(ii) All inter-unit communication (internal to the process), are done via a local memory within the processing unit.

(iii) All unit sequencing is done via a process manager which is responsible for the pipelining of the requests through the system.

(iv) The process manager supervises the execution of the other two paths, running in parallel to the process dealing with communication requests.

The process manager itself should be implemented with a back-up processor for reliability reasons.

The actual task-processor allocation within each processing unit is as follows:

(i) The communication process is assigned to two processors. To allow for continuous response to newly generated requests, the first processor

contains the code for the RH and RCI tasks. The second processor contains the code for the task manager, the RA task, and the R/Wm task.

(ii) The arbitration process is subdivided into three parts. The first processor has the task manager code and the MG task code. The second processor is responsible for the RM task and the RV task. The third processor is responsible for the PI and PS tasks. Also, due to its long execution time, the arbitration process is partially duplicated by having a fourth processor obtain the codes of the last two processors.

(iii) The allocation process has two processors in order to take advantage of the concurrency associated with the PS_p and SC tasks. The first processor is responsible for the PS_p task whereas the second processor contains the code for the task manager and the SC task.

(iv) The diagnostics process is subdivided into two parts. One processor is responsible for the task manager and the S-U and DIS tasks. The second processor has the code for the TLT task and TCT tasks. Also note, that in order to avoid dedicating a processor for each time measurement, software controllable hardware counters can be used as timers.

The bus controller task processor allocation is shown in Figure 4.12.

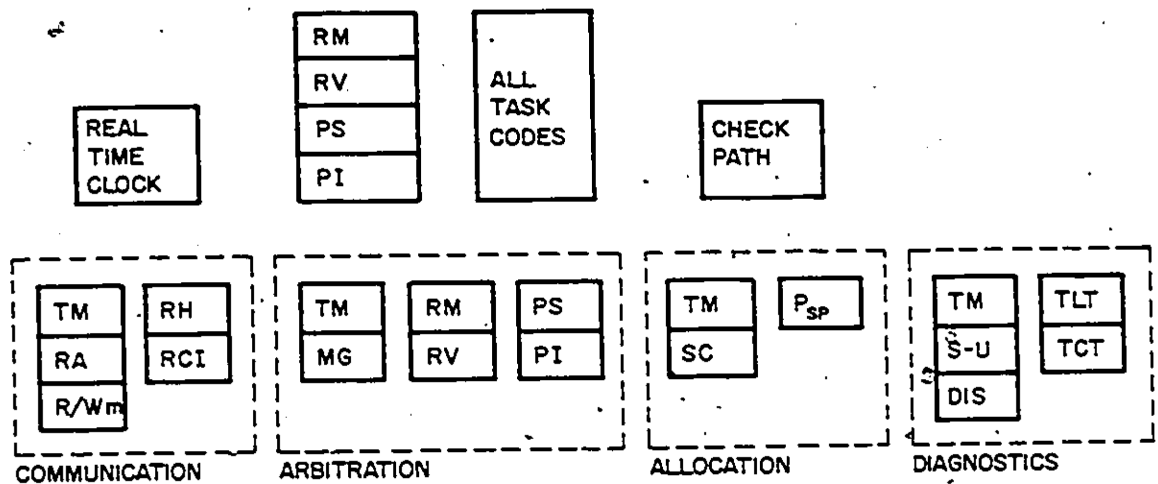


FIGURE 4.12 – TASK-PROCESSOR ALLOCATION FOR THE BUS CONTROLLER.

Finally, it should be remembered that the above allocation scheme is just a suggested first draft allocation. In order to optimize the allocation one can use the general simulation procedure outlined in Chapter 3, in an iterative fashion, until the desired system response characteristics are achieved.

4.6 CONCLUSIONS

The segmented bus allows for multiple, mutually exclusive communication to take place simultaneously, thus upgrading the capabilities of a single bus. The improvement in resource utilization is achieved by isolating a "busy" section on the bus and using the rest of the bus for other communication. The segmented bus has a fail soft capability since a single/link failure need not be a complete system failure. By isolating a faulty part from the system, the system can still operate at reduced capacity. To improve communication

and reduce contention, elements with heavy commnication activities are placed in close proximity to each other.

The suggested CCSB organization has the best properties of the segmented bus and eliminates some of the drawbacks of the DCSB organization at added complexity. The CCSB enhances the fail soft capability of the segmented bus to a limited fail safe capability. Time-out option and diagnostics routines can be used to detect a faulty switch/link. An alternate data path is provided for each element via the control/data bus. This alternative path may be used when a switch/link failure is detected. In addition, for extremely high reliability, one can allow the transformation of CCSB architecture into a DCSB architecture. Thus, by properly modifying the control interface, the centrally controlled organization can be switched to distributed control in the case of controller failure.

Generally, the CCSB is best suited for highly interactive networks with short physical distances among its nodes. In the case of large distances, the use of separate control/data bus and additional control lines may not be as cost effective.

A network with large numbers of elements may require an additional segmented data bus, or dedicated data bus if there are heavy communication activities. The option of an additional dedicated data bus may be used, as the switches along the data bus increase the total propagation delay through the communication path.

The CCSB organization has the advantage that bus controller capability can be enhanced without disturbing the rest of the system. In the first phase of implementation, it is good practice to provide all the necessary hardware connections from the elements to the bus controller. It is not essential to include all the functions described for the bus controller during this phase. Only the functions, specified in terms of tasks which are vital to control and maintenance of basic communication, must be implemented. As the bus controller design is based on a multi-microcomputer architecture the remaining functions may be added in later stages of the development, or as required.

Finally, from a cost point of view, the DCSB organization can be regarded at about the same level as a time shared bus but with improved communication. The CCSB organization can be compared to the crossbar switch scheme. Although the CCSB organization cannot provide the same system throughput as the crossbar switch, it has other features which cannot be implemented by the crossbar switch. Thus the cost of the CCSB organization is a direct function of how many of the bus controller functions are implemented.

4.7 REFERENCES

ANDE 75 G.A. Anderson and E.D. Jensen, "Computer Interconnection Structures: Taxonomy Characteristics and Examples", ACM Computing Surveys, Vol. 7, No. 4, DEC 1975, PP 197-214.

DAVI 78 J.W.E. Davidson, "The Design of Fault Tolerant Interconnection Networks", Ph.D. Thesis, University of Illinois, Urbana - Champaign, U.S.A., 1978.

THOM 75 P.M. Thompson and Z.H. Glanz, "A Data Sorting System Using High Speed Bus", AFIPS Conference preceeding 1975.

THUR 72 K.J. Thurber et al, "A Systematic Approach to The Design of Digital Busing Structures", Proc. IEEE Fall Joint Computer Con. 1972, PP 719-740.

WEIT 80 C. Weitzman, "Distributed Micro/Minicomputer Systems, Structure, Implementation and Applications". Prentice Hall Inc., Englewood Cliffs, N.J., U.S.A., 1980.

5.0 SUMMARY AND RELATED RESEARCH TOPICS

In this thesis three main problems related to multiple microprocessor systems were treated. As stated in the introduction, each of these problems is considered separately in a self-contained chapter. In this chapter, the major ideas presented in the thesis are summarized followed by an outline of a number of recommended related research topics.

5.1 THESIS SUMMARY

A general hardware architecture suitable for multiple microprocessor systems was defined in chapter two. The main characteristics of three prominent multiple processor organizations: multiprocessors, computer networks, and multi-microcomputers, were described. Next, to provide a proper reference, the various motivations behind using multiple processor systems were considered in general terms. Following that, the discussion was restricted to microprocessors and their role in multi-microcomputer systems. In order to provide proper insight into the design and implementation aspects of a multi-microcomputer system, its functional design was presented. First the system's physical and logical structures were described, followed by a general presentation about arbitration, allocation, and coordination of the system's resources. Finally, the main characteristics of a Task-Driven multi-microcomputer architecture, were described.

The design procedures and implementation aspects of an executive for a Task-Driven system were presented in chapter three. First, the design philosophy behind this

executive was discussed. Next, a two level (process and task) hierarchical structure executive, with its functional elements, was developed. The executive outer layer, which includes the process sequencer and task allocator, provides the communication with the external world. The inner layer, which includes task managers and service tasks, provides the direct hardware control. Following that discussion, the communication protocol used during the request, assignment and execution of a given task was described. In order to provide a systematic presentation about task interrelations, task hierarchies were discussed in general, followed by a presentation of task interrelations in terms of state diagram and control primitives. To provide additional insight into the implementation aspects of the executive, an outline of the task allocator program was presented, followed by the presentation of the system's static and dynamic conditions, in terms of look-up tables . Finally, some of the most important ideas regarding task partitioning and allocations were outlined.

The general interconnection scheme for multi-micro-computer systems was defined in chapter four. After brief discussion about bus sharing schemes, the concept of the segmented bus, which accomodates simultaneous, mutually exclusive transfers on a data bus, was introduced . As the segmented bus can be controlled either in a distributed or centeralized fashion, two respective architectures were identified.

For each of the two architectures (distributed control segmented bus and centrally controlled segmented bus) a commuinication protocol and the main implementation aspects were described. Also, after identifying the major functions needed to be executed by the central bus controller, its conceptual design as a multi-microcomputer system was described.

THESIS LIMITATIONS

To conclude this section, the two main limitations of the Task-Driven multi-microcomputer system are presented. A Task-Driven system assumes that a given job can be partitioned into a number of independent tasks. However, if the partitioning process yields either tasks with high interdependency, or just very few independent tasks, the applicability of the Task-Driven system is questionable. The other limitation is that for any given application, a complete job partitioning and task allocation must be performed. This may lead indirectly to an increase in development time and system cost. However, one should mention that both of the above limitations apply equally to all distributed data processing systems.

5.2 RECOMMENDED RELATED RESEARCH TOPICS

In the following, additional research ideas as related to each of the chapters will be presented. The coverage of the second chapter, which is based on a literature survey, can be extended into the following three areas:

(1) The relationship of multi-microcomputer architecture to the concept of distributed data

processing should be investigated in more detail. Also general guide lines should be given as to when and how to apply this concept to various application areas.

(2) Although the Task-Driven architecture was identified as the most suitable for control applications, additional analysis must be undertaken as to its limitations and applicability to other application areas.

(3) A potential research topic would be to define the various signals, and other hardware and software aids that would facilitate the interconnection of a multi-microcomputer system. In line with this, a critical survey of various hardware and software aids on existing microprocessors and their applicability may also be provided.

Related research ideas with respect to the third chapter are in the following areas:

(1) For systems with extensive amount of communication with the external world, the possibility of converting the two level executive to a three level executive can be investigated. By assigning the external communications to a third level, all interface with the external world can be done through this level. In this case, one could have general purpose programs at the process and task levels, and provide the required system customizing at the third level.

(2) Research is required to define the maximum number of transactions a single executive can handle, thus indirectly the number of processing elements. Based on that, a system designer knowing the given system load, may be able to better judge the executive's adequacy.

(3) The possibility of implementing the executive, or parts of, it in firmware and/or hardware should be investigated. This is important as it affects both speed of operation, and cost of the executive, and thus indirectly, its applicability to small systems.

All the above points could be part of a more general research area, which checks the possibility of designing a general purpose executive with simulator and optimizer programs. Such a general purpose executive would be customized to a particular system by specifying the system's basic and unique parameters. This general purpose executive program can be provided by the industry and offered to users as any other multitasking real-time executive for microprocessors.

(4) Another extension would be to allow for dynamic loading of the system's static conditions. In this case, instead of having fixed system parameters, they are updated according to new requirements, and are down loaded from a central location. Thus, the system is made to execute a wide variety of processes, as long as they are based on tasks which are programmed in the system. For larger systems, this idea can be further extended by forming a

network with a number of these executives. Each of the executives controls a local network and receives the appropriate system parameters from a central unit, which supervises all of these executives.

(5) Additional research needs to be done with respect to the problems of job partitioning and task allocation as related to Task-Driven systems.

With respect to the segmented bus architecture and the bus controller, covered in chapter four, the following research topics are of interest:

(1) One needs to establish the maximum number of transfers that the segmented bus and the bus controller can handle. This also defines the number of elements which can be connected to the segmented bus. Such an investigation must consider various parameters such as size of transactions, the distance between communicating elements, and the variety of communication. In line with this, the ultimate design would be to provide a general purpose segmented bus controller. This controller should be able to accommodate a variety of communication needs with customizing done by specifying the number of elements in the system and their software priority. Also, a general simulation program must be provided to determine the number of elements which can be effectively controlled by the bus controller.

(2) One can extend the capabilities of a system by forming a network of rings. Each local ring, being

a segmented bus, is controlled by a bus controller and is also connected to a global ring which is a high speed serial bus. Each bus controller, aside from its defined activities and interfaces, whould have an additional interface to the global ring. Through this interface, one element from one local ring can be connected to another element on a different local ring.

(3) For high reliability systems, one should investigate the idea of designing a system with both distributed and central control. Normally, the system operates as a centrally controlled system, and upon failure of the bus controller, switches automatically to the distributed control scheme. The main research here has to do with the detection of the bus controller failure and the mechanism of automatic switch over to the distributed control scheme.

(4) A way of combining the executive design with the segmented bus architecture, with the bus controller being controlled by the executive, should be investigated. For such design, the capabilities and applicability of the design to various areas must also be studied.

BIBLIOGRAPHY

ATWO 76 J.W. Atwood, "Concurrency in Operating Systems", IEEE Computer, Oct. 1976, PP 18-25.

BART 80 J.P. Barthmaier, "Multiprocessing System Mixes 8-and 16-Bit Microcomputers", Computer Design, Vol.19, No. 2, Feb. 1980, PP 137-144.

BELL 81 J.R. Bell, "Future Directions in Computing", Computer Design, Vol. 20, No. 3, March 1981, PP 95-102.

BRIN 79 B.L. Brinkman, "A Selection of Multi-Microcomputer Systems", Mini-Micro Systems, Jan. 1979, PP 86-90.

CHIL 79 R.E. Childs, "Multiple Microprocessors Systems: Goals, Limitations and Alternatives", Exploding Technology Responsibility Growth-Digest of Papers-COMPCON, Spring 79, San Francisco, Calif., March 1979, PP 94-97.

COLE 79 D. Coleman, J.W. Hughes and M.S. Powell, "Engineering Data Processing Programs for Multiple Microprocessors", Microprocessors and Microsystems, Vol. 3, No. 2, March 1979, PP 83-85.

DOWS 79 M. Dowson, B. Collins and B. McBride, "Software Strategy For Multiprocessors", Microprocessors and Microsystems, Vol. 3, No. 6, July/Aug. 1979, PP 263-266.

HABE 76 A.N. Habermann, "Introduction to Operating System Design", Science Research Associates, Inc., U.S.A., 1976.

HARR 77 J.A. Harris and D.R. Smith, "Hierarchical Multiprocessor Organizations", Conf. Proc. 4th Ann. Symp. Computer Architecture, Silver Spring, Md., March 1977, PP 41 - 48.

HOAR 74 C.A.R. Hoare, "Monitors: An Operating System Structuring Concept", Communications of the ACM, Vol. 17, No. 10, Oct. 1974, PP 549-557.

HOOG 77 C.H. Hoggendoorn, "Reduction of Memory Interference in Multiprocessor Systems", Conf. Proc. 4th Symp. Computer Architecture, Silver Spring Md., March 1977, PP 179-183.

INTE 79 INTEL Application Note AP-61, "Multitasking For The 8086", 1979.

INTE 79 INTEL Application Note AP-33, "RMX/80 Real-Time Multitasking Executive", 1979.

KINN 78 L.L. Kinney and R.G. Arnold, "Analysis of a Multiprocessor System with a Shared Bus", Conf. Proc. 5th Ann. Symp. Computer Architecutre, Palo Alto, Calif., April 1978, PP 89-95.

LEMA 80 I. Lemir, "Microprocessors and Microcomputers," Technology Forecast, Digital Design, Dec. 1980, PP 30-34.

MAZA 77 G. Mazare', "A Few Examples of How to use a Symmetrical multi-micro-processor", Conf. Proc. 4th Ann. Symp. Computer Architecture, Silver Spring Md., March 1977, PP 57-62.

PICK 79 C. Pick, D. Hird and D. Elliot, "Multimicroprocessor Computer System", Microprocessors and Microsystems", Vol. 3, No. 4, May 1979, PP, 179-183.

SATY 80 M. Satyanarayanan, "Multiprocessing: An Annotated Bibliography", IEEE Computer, May 1980, PP 101-116.

TOON 78 H.M.D. Toong, "Multi-Microprocessor Systems", Center For Information Systems Research, M.I.T. Cambridge, Mass., 1978.