

## INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.
2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

**University  
Microfilms  
International**  
300 N. Zeeb Road  
Ann Arbor, MI 48106



8304681

Jaragh, Mansour H.

ANALYSIS OF PARALLEL MULTIPROCESSOR ARCHITECTURE

*New Mexico State University*

PH.D. 1982

University  
Microfilms  
International 300 N. Zeeb Road, Ann Arbor, MI 48106

Copyright 1983

by

Jaragh, Mansour H.

All Rights Reserved



PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark .

1. Glossy photographs or pages: \_\_\_\_\_
2. Colored illustrations, paper or print \_\_\_\_\_
3. Photographs with dark background \_\_\_\_\_
4. Illustrations are poor copy \_\_\_\_\_
5. Pages with black marks, not original copy \_\_\_\_\_
6. Print shows through as there is text on both sides of page \_\_\_\_\_
7. Indistinct, broken or small print on several pages
8. Print exceeds margin requirements \_\_\_\_\_
9. Tightly bound copy with print lost in spine \_\_\_\_\_
10. Computer printout pages with indistinct print \_\_\_\_\_
11. Page(s) \_\_\_\_\_ lacking when material received, and not available from school or author.
12. Page(s) \_\_\_\_\_ seem to be missing in numbering only as text follows.
13. Two pages numbered \_\_\_\_\_ . Text follows.
14. Curling and wrinkled pages \_\_\_\_\_
15. Other \_\_\_\_\_

University  
Microfilms  
International



ANALYSIS OF PARALLEL MULTIPROCESSOR ARCHITECTURE

BY

'MANSOUR JARAGH, B.S., M.S.'

A Dissertation Submitted to the Graduate School  
in partial fulfilment of the requirements  
for the Degree  
Doctor of Philosophy

Major Subject: Electrical and Computer Engineering  
Related Area: Computer Science

New Mexico State University

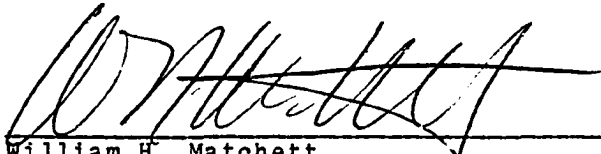
Las Cruces, New Mexico

October 1982

© Mansour Hameed Jaragh

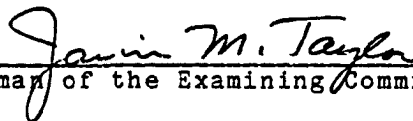


"Analysis of Parallel Multiprocessor Architecture," a dissertation prepared by Mansour Hameed Jaragh in partial fulfillment of the requirements for the degree Doctor of Philosophy, has been approved by the following:



---

William H. Matchett  
Dean of the Graduate School



---

Javin M. Taylor  
Chairman of the Examining Committee

November 12, 1982

---

Date

Committee in Charge:

Dr. Javin M. Taylor  
Dr. Gerald M. Flachs  
Dr. Robert L. Golden  
Dr. John B. Johnston  
Dr. M. Don Merrill  
Dr. Arun G. Walvekar

## ACKNOWLEDGMENTS

I would like to acknowledge the guidance and great help of my advisor Dr Javin M. Taylor both during my graduate studies and during the course of this research. The professors at NMSU and the committee members deserve a special thank. I would like to thank Mr. Foo Lam of the Instrumentation Directorate at White Sands Missile Range, and the Kuwaiti Civil Services for their support. I appreciate the cooperation of the project team and special thanks goes to Robert Widlicka.

Finally, I would like to acknowledge the encouragment and support of my family, especially my wife Fakhreyah, and my two sons Abdullah and Hamid for not seeing me among them as often as they should.

VITA

August 18, 1952 - Born in Kuwait city, Kuwait.

1975 B.S.E.E., Tulane University, New Orleans,  
Louisiana, U.S.A.

1975-1977 Engineer at Kuwait Ministry of Telecommunication,  
Kuwait.

1979 M.S.E.E., New Mexico State University, Las Cruces.

1981-1982 Research Assistant, Department of Electrical and  
Computer Engineering, New Mexico State University.

PROFESSIONAL AND HONORARY SOCIETIES

Institute of Electrical and Electronic Engineers

Association for Computing Machinery

Eta Kappa Nu

Phi Kappa Phi

FIELDS OF STUDY

Major Field: Computer Engineering

Microprocessor and multiprocessor architecture

Performance evaluation

Related Area: Computer Science

Computer system computation.

ABSTRACT

ANALYSIS OF PARALLEL MULTIPROCESSOR ARCHITECTURE

BY

MANSOUR H JARAGH, B.S.E.E., M.S.E.E.

Doctor of Philosophy in Electrical and Computer Engineering

New Mexico State University

Las Cruces, New Mexico 1982

Dr Javin M. Taylor, Chairman

This dissertation describes the author's research in 1) the design and development of small general purpose bit-slice emulators; 2) model formulation of networks using small processors or bit-slice emulators; 3) performance evaluation of the resultant network models; and 4) design methods for evaluation of these network structures.

Networks of small processing elements can have unlimited variety. Out of the many possible multiprocessing architectures, we have proposed three models to study in this dissertation. These models treat von Neumann and non-von Neumann structures and provide a basis for the

analysis and performance evaluation of various parallel configurations of small LSI processing elements, such as microprogrammable bit-slice devices. The three parallel architecture models analyzed are 1) the controlled multiserver model; 2) the array model; and 3) the data flow model.

The general purpose bit-slice emulator developed at New Mexico State University is used in the network models. The bit-slice emulator is very versatile and therefore can easily be modified to fit various requirements. Analytic and simulation techniques are employed in this study. For some models, both micro and macro analyses are performed. At the macro level, the analysis is carried out at the job level, whereas at the micro level the analysis is concerned with the behavior of the system at the instruction execution level.

Our intent is not to compare these models nor expect them to be universally applicable, but to provide building blocks and various approaches. We believe that this contribution will assist network researchers in effectively constructing and evaluating their own particular network models.

## CONTENTS

LIST OF FIGURES . . . . . xii

LIST OF TABLES . . . . . xvi

<u>Chapter</u>	<u>page</u>
I. INTRODUCTION . . . . .	1
GOALS AND BACKGROUND . . . . .	1
MOTIVATION . . . . .	4
DISSERTATION ORGANIZATION . . . . .	5
II. A GENERAL VIEW OF MULTIPROCESSING SYSTEMS AND THEIR CLASSIFICATION . . . . .	10
VON NEUMANN STRUCTURES . . . . .	10
NON-VON NEUMANN STRUCTURES . . . . .	15
Direct Execution Machine (D.E.M.) . . . . .	15
Data Flow Machine . . . . .	18
PARALLELISM IN COMPUTER ARCHITECTURE . . . . .	19
CLASSIFICATION OF COMPUTING SYSTEMS . . . . .	21
Flynn Classification Scheme . . . . .	21
Handler Classification Scheme . . . . .	25
EXAMPLE OF EACH ORGANIZATION TYPE . . . . .	28
Pipeline Processor . . . . .	29
Array Processing . . . . .	32
III. A MODULAR BIT-SLICE PROCESSING ELEMENT (PE) . . . . .	38
INTRODUCTION . . . . .	38
THE ALU UNIT . . . . .	42
THE CONTROL UNIT . . . . .	43
The Sequencer Unit (AM2910) . . . . .	45
Operation of The Control Unit . . . . .	48
The Control Store . . . . .	48
Condition Code Mux (CC) . . . . .	49
The Interrupt Control Unit (AM2914) . . . . .	49
IV. QUEUEING AND SIMULATION CONCEPTS . . . . .	51
INTRODUCTION . . . . .	51
QUEUEING PRINCIPLES . . . . .	53
Basic Relationships . . . . .	54

	Markov Process . . . . .	56
	Networks of Queues . . . . .	57
	SIMULATION PRINCIPLES . . . . .	59
<b>V.</b>	<b>THE CONTROLLED MULTISERVER MODEL . . . . .</b>	<b>63</b>
	INTRODUCTION . . . . .	63
	SYSTEM ORGANIZATION . . . . .	63
	SYSTEM ANALYSIS . . . . .	65
	The Analytic Model . . . . .	67
	State diagram derivation . . . . .	70
	Derivation of the state equations . . . . .	71
	Solution of the analytic technique . . . . .	81
	The Simulation Model . . . . .	82
	ANALYSIS OF RESULTS . . . . .	86
<b>VI.</b>	<b>A PROGRAMMABLE ARRAY MODEL . . . . .</b>	<b>98</b>
	INTRODUCTION . . . . .	98
	SYSTEM ORGANIZATION . . . . .	99
	HARDWARE ARCHITECTURE . . . . .	100
	Processing Elements (PE's) . . . . .	100
	The Control Section . . . . .	101
	ANALYSIS OF THE ARRAY MACHINE . . . . .	110
	The Analytic Model . . . . .	111
	The Simulation Model . . . . .	120
	POSSIBLE APPLICATION EXAMPLES . . . . .	124
	ANALYSIS OF RESULTS . . . . .	132
	Macro-Analysis of the Array System . . . . .	133
	Micro-Analysis of the Array Model . . . . .	142
	REMARKS . . . . .	147
<b>VII.</b>	<b>DATA FLOW MODEL . . . . .</b>	<b>148</b>
	DATA FLOW CONCEPTS . . . . .	148
	DATA FLOW PROGRAM EXAMPLE . . . . .	149
	THE BASIC HARDWARE UNITS FOR A DATA FLOW SYSTEM . . . . .	152
	The PEQUE to PE Connection . . . . .	153
	The Processing Element (PE) . . . . .	153
	The Queueing Circuit . . . . .	155
	The Memory Section . . . . .	157
	THE DATA FLOW MODEL . . . . .	159
	The Analytic Model . . . . .	161
	The Simulation Analysis . . . . .	168
	Case 1 . . . . .	169
	Case 2 . . . . .	169
	ANALYSIS OF RESULTS . . . . .	174



VIII. SUMMARY AND CONCLUSION . . . . .	181
SUMMARY . . . . .	181
FUTURE APPLICATION . . . . .	185

LIST OF REFERENCES . . . . .	186
------------------------------	-----

Appendix

	<u>page</u>
A. SOLUTION FOR THE CASE N=3 AND C=3 OF THE CONTROLLED MULTISERVER MODEL OF CHAPTER V	192
B. THE ANALYTIC PROGRAM FOR THE CONTROLLED MULTISERVER MODEL OF CHAPTER V . . . . .	194
C. THE SIMULATION PROGRAM FOR THE CONTROLLED MULTISERVER MODEL OF CHAPTER V . . . . .	197
D. THE ANALYTIC PROGRAM FOR THE ARRAY PROCESSING MODEL OF CHAPTER VI . . . . .	203
E. THE SIMULATION PROGRAMS FOR BOTH ARRAY MODELS OF CHAPTER VI . . . . .	206
F. THE ANALYTIC PROGRAM FOR THE DATA FLOW MODEL OF CHAPTER VII . . . . .	212
G. THE SIMULATION PROGRAMS FOR BOTH MODELS OF THE DATA FLOW SYSTEM OF CHAPTER VII . . . . .	215

## LIST OF FIGURES

<u>Figure</u>	<u>page</u>
1. The Basic Organization of a von Neumann Machine . . .	12
2. Basic Task Execution Flow. . . . .	13
3. Organization of a Direct-Execution Computer. . . . .	16
4. The Four Flynn Classifications . . . . .	23
5. The Possible Diagram for $C=(2,5,-)$ . . . . .	27
6. A Four-Stage Pipeline System . . . . .	31
7. The Throughput Versus the Number of Stages in a Pipeline System . . . . .	33
8. The Relation Between the Percent Throughput and the Number of Tasks Waiting . . . . .	34
9. Basic Configuration of an Array System . . . . .	36
10. The Organization of the Complete ILLIAC IV Computer	37
11. The Microinstruction Field Layout . . . . .	41
12. The Basic Bit-Slice Design . . . . .	44
13. The Modified ALU Board . . . . .	46
14. The Basic Block Diagram of the AM2910 . . . . .	47
15. Some Random Variables Used in the Queueing System .	55
16. The Basic Flowchart for Simulating a Model . . . . .	61
17. The System Block Diagram . . . . .	64
18. A Detailed System Organization . . . . .	66
19. The Overall Queueing Structure . . . . .	67

20.	The Simplified System Model . . . . .	69
21.	State Transition Diagram for the Controlled Multiserver System . . . . .	72
22.	The State Transition Diagram for the Second Case . .	84
23.	Sub-State Transition Diagram for $n=5$ . . . . .	85
24.	The Simulation Flowchart for the Controlled Multiserver Model . . . . .	87
25.	The Utilization Versus the System Capacity . . . . .	89
26.	The Throughput Versus the System Capacity . . . . .	90
27.	The Utilization as a Function of $n$ . . . . .	93
28.	The SPE Utilization as a Function of $n$ . . . . .	95
29.	System Throughput as a Function of $n$ . . . . .	96
30.	The Average Queue Length as a Function of $n$ . . . .	97
31.	A Block Diagram of the System . . . . .	99
32.	The Simplified PE Block Diagram . . . . .	102
33.	The Microinstruction Fields . . . . .	103
34.	The Complete System Organization . . . . .	105
35.	The Interconnection Network . . . . .	106
36.	The Arrangement of the Data Mux for a 4-PE Array System . . . . .	107
37.	The Two Interconnection Networks . . . . .	109
38.	The Queueing System . . . . .	110
39.	The State Diagram for the System . . . . .	112
40.	The Simulation Flowchart for the Macro Model . . .	122
41.	The Composite Flowchart for the Array Model . . .	123
42.	The Flowchart for the First Example . . . . .	126
43.	The Uniprocessor Versus the Array System Cycle Requirements . . . . .	128

44.	The Flowchart for the Second Example . . . . .	130
45.	The Main and Local Memories Contents . . . . .	132
46.	The PE and System Utilization as a Function of $\lambda$ . . . . .	134
47.	The Average Queue Length as a Function of $\lambda$ . . . . .	136
48.	The Utilization as a Function of $n$ . . . . .	137
49.	The Average Queue Length as a Function of $n$ . . . . .	138
50.	The Utilizations as a Function of $c$ . . . . .	139
51.	The Average Queue Length as a Function of $c$ . . . . .	140
52.	The Utilization as a Function of $n$ and $\lambda$ . . . . .	141
53.	The Utilization as a Function of $c$ . . . . .	144
54.	The Utilization as a Function of the Arrival Rate . . . . .	145
55.	The Average Queue Length as a Function of $\lambda$ . . . . .	146
56.	The Data Flow Graph . . . . .	150
57.	The Basic Blocks of a Data Flow System . . . . .	152
58.	The PEQUE to PE connection . . . . .	154
59.	The PE Circuit Diagram . . . . .	156
60.	The Basic Blocks of the Memory Section . . . . .	158
61.	The Queueing Model of a Data Flow System . . . . .	161
62.	The State Transition Diagram . . . . .	162
63.	The Simulation Flow Chart for Case 1 . . . . .	170
64.	The Example Program to be Simulated . . . . .	171
65.	The Simulation Flowchart for Case 2 . . . . .	172
66.	The Memory Cells for the Program of Figure 64 . . . . .	173
67.	The Processing Elements Utilization as a Function of $n$ . . . . .	175
68.	The Throughputs as a Function of $n$ . . . . .	176

69.	Average Queue Contents as a Function of $n$ . . . .	177
70.	The PE Utilization for the Second Case . . . . .	179
71.	The Average Queue Contents for the Second Case . .	180

LIST OF TABLES

<u>Table</u>	<u>page</u>
1. The Utilization of Individual SPE's ( $\psi=0.616$ ) . . . .	91
2. The Utilization of Individual SPE's ( $\psi=1.5152$ ) . . . .	92
3. The Total Throughput of the SPE's . . . . .	92
4. Definitions of the Signals . . . . .	155

Chapter I  
INTRODUCTION

1.1 GOALS AND BACKGROUND

The purpose of this research is to develop models and procedures for analyzing the performance of networks consisting of large numbers of small processing elements. We envision these small processing elements as, perhaps, microprogrammable bit-slice devices or single board microprocessors. However, this is not a restriction as to the use of the models presented.

The motivation for this research grew out of studies and development of reconfigurable architecture and universal cascadable bit-slice emulators for the US Army White Sands Missile Range. In this research, a bit-slice emulator was developed to emulate 8-bit and 16-bit microprocessors, as well as special purpose networks such as array processors. It became obvious that if these bit-slice emulators had the desired reconfigurability, the next step was that of developing generalized procedures for constructing models and analyzing networks comprised of these emulators.

This dissertation describes the author's contribution to this research, which includes 1) design and development of the control portion of the bit-slice emulator, 2) model formulation and analysis for three types of networks comprised of bit-slice emulators, and 3) design methods for evaluation of these network structures.

Networks of small processing elements can have unlimited variety. Consequently, in this research the scope has been narrowed to the study of three network models. A practical methodology is developed and applied to the performance investigation of these three structures. Each structure is studied by itself. The main interest is the construction and analysis of a mathematical model for each structure. Analytic and simulation programs are developed to enhance our study. This research is particularly appropriate due to recent research efforts in VLSI, the introduction of 16-bit microprocessors, and the development of radically different LSI architectures, such as the recently announced Intel 432.

Since its invention, the computer has gone through many developmental stages. As more complex jobs are created, the need for developing faster and faster machines becomes more imminent. With the continuously decreasing cost of microprocessors and other LSI chips, the notion of using large arrays of these devices to perform in parallel becomes a practical one.



In parallel processing more than one processor is used to accomplish the processing of a given job, provided that this job is applicable in a parallel processing environment. Parallel processing is often referred to as multiprocessing or multiprogramming. Multiprocessing is defined as the simultaneous processing of two or more portions of the same program by two (or more) processing units. Multiprogramming is defined as the time and resource sharing of a computer by two (or more) programs residing simultaneously in primary memory. Parallel processing can include either of the above or a combination of them.

Why are there so many different computer configurations and by what criteria can the performance of a computer be judged? Some typical criteria are the speed, reliability, versatility, programming convenience, cost, and most importantly the computing power. The computing power includes parameters such as the number of bits per word, and the size of the main memory, plus others.

Many of the computer architecture innovations are mainly motivated by the need for processor speed. The processor speed and computing power are critical to some real time analysis, such as pattern recognition, image analysis, and information gathering from satellites.

Over the last twenty years, a great deal of research effort has been devoted to the development of performance

evaluation of large computer systems [ROBL81],[SAST73], and [DENN78]. Now, with the interest in networking, the time has come to study performance models of parallel configurations of LSI processing elements [WONG78].

## 1.2 MOTIVATION

The term "performance evaluation" typically implies the evaluation of large computers. While it is generally true that it is not cost effective to analyze the performance evaluation of small systems employing a few microprocessing elements, it does become cost effective to evaluate the performance of such systems when considering the implementation of hundreds of these devices in a parallel architecture. Due to the availability and the lower cost associated with LSI chips, future systems will emerge that use these microprocessing elements, [BRIG79], and [HWAN81]. Thus, devising algorithms to analyze and evaluate their performance is important.

Out of the many possible multiprocessing architectures, we have proposed three models to study in this dissertation. These models provide a basis for the analysis and performance evaluation of various parallel configurations of small LSI processing elements, such as microprogrammable bit-slice devices. The three parallel architecture models analyzed are

1. The controlled multiserver model

2. The array model
3. The data flow model.

The general purpose bit-slice emulator developed at New Mexico State University is used in the above models. The bit-slice emulator is very versatile and therefore can easily be modified to fit our needs. Some minor modifications are discussed in order to make the designed system applicable to a parallel processing environment. Von Neumann and non-von Neumann architectures are pursued. Simulation and analytic techniques are developed in order to analyze the different systems studied.

The basic measures of performance that are considered are as follows:

1. Processing element utilization.
2. The control unit utilization.
3. The total system throughput.
4. The average queue length (where applicable).

### 1.3 DISSERTATION ORGANIZATION

The material presented in this dissertation is divided into two main parts. The first part, chapters II through IV, discusses the basic issues of parallel architecture, such as the different classes of system organization and the different classification schemes, basic microprogrammed emulation, and performance evaluation analysis techniques.

The second part, chapters V through VII, focuses in on several different models proposed for parallel processing and the resulting performance evaluation analysis.

Chapter II discusses the basic types of parallel computer architectures. Two architectures in general are viewed: the von Neumann and the non-von Neumann architectures. The direct execution computer and the data flow computer are used as examples of the non-von Neumann machine, whereas the array and pipeline machines are used as examples of the von Neumann machines. The schemes used for classifying computing systems are also reviewed. Basically two classification schemes are discussed, the Flynn classification scheme and the Handler classification scheme.

The basic processing element that is used in the models of chapters V through VII is discussed in chapter III. The principal blocks of the control unit and the processing unit sections are presented. The processing element consists of an ALU unit and a control unit. The basic ALU unit ( which can be increased by orders of 8 ) is an 8-bit bit-slice machine using the Am2903 (the super slice).

Chapter IV reviews the basic concepts of queueing such as the arrival rate, the service rate, and the queueing discipline. Due to its importance in this analysis, the "network of queues" method is presented. The general simulation flow graph is also discussed. The APL language

is used for the analytic case while GPSS language is used for the simulation study. Poisson arrival and exponential service times are employed in most of the models studied.

In chapter V, the proposed model for the controlled multiserver system is presented. In the controlled multiserver model, several processing elements (PE's), each with a specific function, are employed. All the PE's are controlled by a single central control unit and are activated via an instruction analyzer. The mathematical model is studied using two methods: analytic and simulation. This model is particularly good in application programs where there are only a few number of operations and each group of instructions is executed repeatedly.

The second model, the array model, is presented in chapter VI. Unlike the model of chapter V, all the PE's or a subset of the PE set are utilized in executing an incoming job. The PE's allocated to the incoming job are assigned using a probability selection vector of size  $(n \times 1)$ , where  $n$  is the number of PE's. In this analysis we assume that there are always a sufficient number of processing elements to serve the incoming job. An alternative assumption is that there are not enough processing elements to serve the incoming job. This assumption is not necessary in our network research, but the model proposed can be extended to this case.

The array model is analyzed from two different points of view. The macro-model is associated with the job execution in the system. That is, the entity unit in the system is the job as a whole. The macro-model is analyzed with both analytic and simulation methods. The results obtained in the two cases are shown to be similar. The second model, the micro-model, is associated with the instructions in the system, and a fixed number of jobs are considered. The micro-model is analyzed using simulation techniques only. The array model is good for vector-type and matrix-like problems. In general the array systems are very specialized and are tailored for specific application and environment. Failure of any processing element in the system will bring the whole system to a halt.

Lastly, the data flow model is presented in chapter VII. Unlike the two models of chapters V and VI, the data flow model is an example of a non-von Neumann machine. The parallelism in the data flow system lies in the fact that all the processing elements can be busy simultaneously performing distinct operations. Similar techniques are used to analyze the data flow system. The simulation analysis is performed in two parts: macro and micro-analysis. The macro-model which is also analyzed by the analytic technique yields similar results. The micro-analysis, on the other hand, simulates an actual program execution thus providing a

good tool for the performance evaluation of the real hardware. This kind of system is suitable for programs that employ parallelism. Failure of any PE should have negligible effect on the overall throughput of the system.

Finally, chapter VIII summarizes the results obtained in this dissertation and discusses the future research of multiprocessors.

Part of the research described in this dissertation was supported by the Instrumentation Directorate at White Sands Missile Range under contract #DAAD07-81-C-0094, "Demonstration Prototype Cascadable Microcomputer Module."

## Chapter II

### A GENERAL VIEW OF MULTIPROCESSING SYSTEMS AND THEIR CLASSIFICATION

#### 2.1 VON NEUMANN STRUCTURES

The basic characteristics and architecture of digital computers was first set forth in a systematic manner by the mathematician John von Neumann in 1945. A computer that follows the von Neumann structure is referred to as a von Neumann machine. Figure 1 shows the basic organization of a typical von Neumann machine. A von Neumann machine is said to have the following properties:

1. A single sequential memory.

A program and its data are stored intermixed in a single memory, and the memory is referenced with sequential addresses.

2. A linear memory.

The memory is one-dimensional, that is, it has the appearance of a vector of words.

3. No explicit distinction between instructions and data.



4. Meaning is not an inherent part of data.

The meaning of data is assigned by program logic. That is, the interpretation of a pattern as an instruction or a datum depends on the state of the machine when the code is fetched from the memory. If the state of the machine dictates that the code should be transferred to the control unit, then that code is interpreted as an instruction. On the other hand, if the code is transferred to a register, then it is treated as data.

5. A program counter.

A register which is used to indicate the location of the next instruction to be executed and which is automatically incremented with each instruction fetch.

A typical von Neumann machine consists of three basic parts:

- 1) A central processing unit.
- 2) A program storage unit.
- 3) A tube connecting the cpu to the store. The address to the store is sent through this tube. This tube is sometimes referred to as the von Neumann bottleneck [BACK78].

Thus the von Neumann machine employs a single instruction stream which will operate on a single data stream.

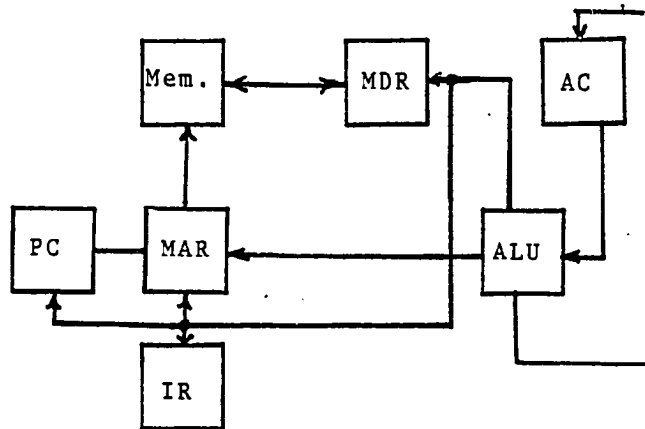


Figure 1: The Basic Organization of a von Neumann Machine

These kinds of machines are usually known as uniprocessors. The speed of processing a certain process (or task) is, therefore, dependent on the speed of this single CPU. The execution of a task in the von Neumann machine follows the flowchart shown in Figure 2.

In order to overcome the execution restriction in the von Neumann architecture, various multiprocessing schemes

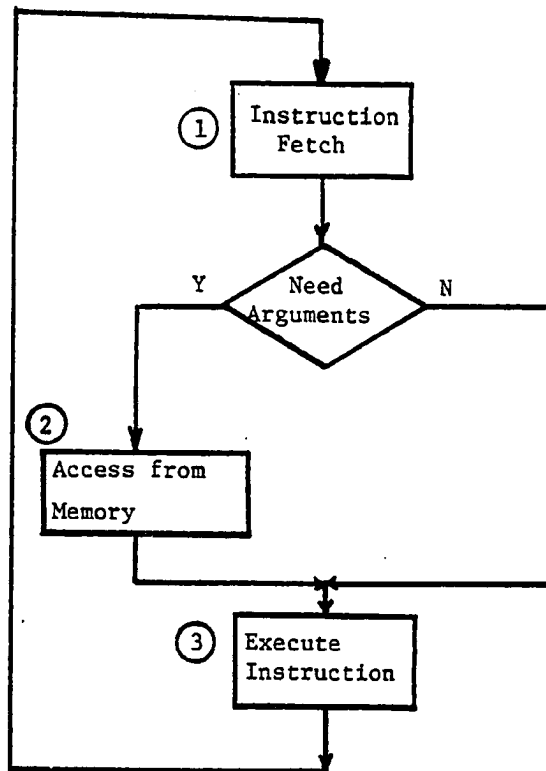


Figure 2: Basic Task Execution Flow.

have been devised. We will briefly discuss the different classes of multiprocessing systems that exist. Also a brief discussion of the different multiprocessing classification schemes will be outlined in the next section.

Before going on with the details, let us define the following terms so that whenever they are referred to in the context of the discussion in this dissertation, these definitions will apply.

1. **Multiprocessor:** A group of computing units each with its own instruction stream and data stream sharing a common memory and control unit. The ANSI (American National Standard X8.-1970) definition of multiprocessor is given as: "A computer employing two or more processing units under integrated control."
2. **Multicomputer:** Independent computers often with one acting as a supervisor in performing a common task at a single location.
3. **Computer Network:** Independent computers at different geographical locations connected by a communication channel. A unique resource at one site can be available to all the members of the network.
4. **Concurrency:** This property is associated with two or more activities that are in progress simultaneously; e.g., the central processing unit can be executing instructions from some task at the same time a peripheral processor is carrying out an input/output operation.

## 2.2 NON-VON NEUMANN STRUCTURES

The concept of non-von Neumann machines has been the concern of research in computer architecture for many years. Some interesting machines have been proposed and built. Examples are the data flow computer [DENN80a], [KELL80], [COTE78], and [JOHN80], the direct execution machine [CHU81], and the early SYMBOL computer [DITZ81]. In chapter VII we will concentrate more on the data flow computers and will attempt to model a basic machine. A simulation program is developed by which the performance of such machines can be studied.

### 2.2.1 Direct Execution Machine (D.E.M.)

An example of a non-von Neumann architecture is the direct execution machine discussed in [CHU81]. The basic organization of a direct execution machine is shown to consist of three processors, which are as follows:

1. The lexical processor
2. The control processor
3. The data processor

The lexical processor assembles source program (of a high level language) characters from the program memory into

tokens such as operators, reserved words, names, and numbers, then delivers them to the language processor (which in turn consists of the control and the data processor). The control processor executes tokens which are part of the control flow, whereas the data processor executes tokens which are part of the data flow. Figure 3 below shows the basic organization of a direct-execution computer (D.E.C.).

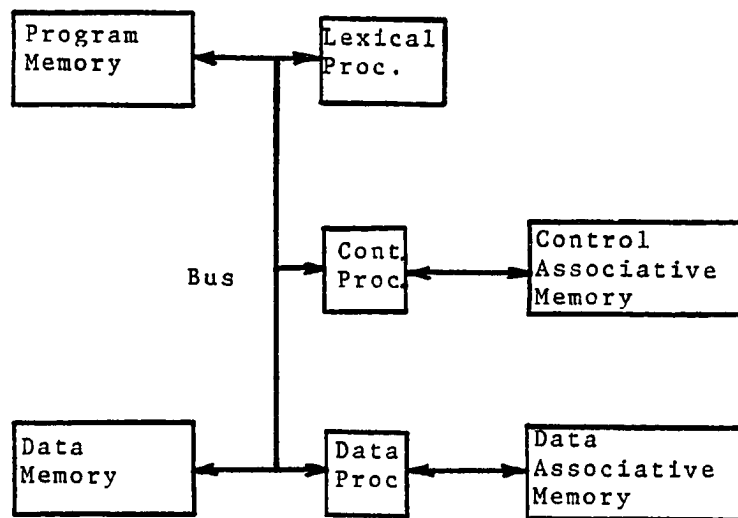


Figure 3: Organization of a Direct-Execution Computer.

Examples of a detailed program storage and organization of a direct-execution machine are found in [CHU81].

A direct-execution computer executes the tokens of a high level language rather than a compiler-generated machine code. In contrast, an indirect-execution computer first translates a high level language into an intermediate language such as polish string, and then executes it by hardware.

The direct-execution machine differs from the von Neumann architecture in the following way:

1. In the basic von Neumann architecture, the program code and data are stored in a single memory, whereas in the D.E.C. the program code is stored in the program memory and data is stored in the data memory.
2. The conventional cpu of a von Neumann machine has been split into two separate processors: the control processor and the data processor. Therefore the control processor executes tokens which are part of the program control flow, while the data processor executes tokens which are part of the data flow. In the von Neumann architecture, the cpu executes both the control instructions such as branch, jump to subroutine, and the data manipulation instructions such as add, multiply.

3. The D.E.C. offers an increase in the instruction execution rate; the lexical processor operates in parallel with the data and the control processor.

### 2.2.2 Data Flow Machine

The data flow architecture is considered a non-von Neumann architecture, as was mentioned earlier. Like any computer, the data flow also sequences through the instructions. However, there is no program counter (PC) to be updated each time an instruction is fetched. Instead, the sequencing of instruction execution depends only on the availability of the operands required by the instruction.

Data flow machines are constructed of modules and the communication between these modules is asynchronous. In his dissertation, Rumbaugh clearly outlines the construction of a basic data flow multiprocessor [RUMB75]. The principle advantage of the data flow multiprocessing system over conventional multiprocessing systems is reduced complexity of the processor memory connection. The instructions in a data flow machine reside in the main memory, and as the operands for a particular instruction (which are specified either initially or are provided during the course of execution of other instructions) are available, then that



instruction is ready, regardless of the states of other instructions. When an instruction execution is completed, the result is provided to all other instructions that require it. We will explain this principle in more detail with illustrative examples in chapter VII.

The kind of algorithms executed in a data flow machine are assumed to have enough parallelism in order to fully utilize the processors. As it will be shown later, the data flow concept will be beneficial only for a particular class of programs.

The architecture of a data flow computer resembles that of a pipeline system. In a pipeline machine, each preceding unit passes on the ready task (or portion of a task) to the next unit for attention. Unlike the conventional pipeline, the instructions in a data flow pipeline must come back to where they started from, i.e., the first unit. Thus the data flow machine could be considered as a circular pipeline.

### 2.3 PARALLELISM IN COMPUTER ARCHITECTURE

When speaking of parallelism in computer architecture, one should be aware of the different levels and kinds of parallelism within a specific system. For example, there can exist the following:

1. Parallelism between jobs

2. Parallelism between subroutines in a run
3. Parallelism between instructions
4. Parallelism between stages in an instruction  
(Hardware level)

For instance, if two equations in an algebraic system are independent of each other and the solution of one variable does not provide an input to the other equation, then these two equations could be processed simultaneously. Therefore, parallel execution of tasks within the same job has occurred. In other situations, parallel execution of two independent jobs concurrently can take place. In a multiprocessor system where more than one processor is controlled by the same control unit, several jobs can be processed within the same time frame. In other parallel executions, the control unit which controls several identical processing elements is used to direct all the PE units with the same instructions, but each PE does its own data manipulation. This type of parallel system is called "array processing."

Various configurations of more than one processing element has resulted in a variety of parallel systems. Each system has its own characteristics, which leads to certain advantages and disadvantages.

## 2.4 CLASSIFICATION OF COMPUTING SYSTEMS

Due to the growth of contemporary computer technology, it seems a little difficult to create a classification scheme for the different types of computing systems. Several classification schemes have been proposed in the past two decades: Flynn classification in 1966 [FLYN66], Feng classification in 1972 [FENG72], and The Handler scheme in 1977 [HAND77]. Each of the above schemes tries to include most of the contemporary systems; nevertheless each suffers from some deficiency. We will discuss the first and the last scheme in some detail.

### 2.4.1 Flynn Classification Scheme

In 1966 M.J. Flynn published a paper in which he classified the different computer structures using the stream concept. Stream simply means a sequence of items (instructions or data as executed or operated on by a processor). The four broad classifications of machine organizations which he defined are given below:

#### 1. SISD:

Single-instruction stream - single-data stream organization. This kind of organization represents the basic von Neumann structure and includes the class of uniprocessors.

## 2. SIMD:

Single-instruction stream - multiple-data stream organization. This kind of organization includes most array processing systems, e.g., ILLIAC IV, Goodyear corporation STARAN, and SOLOMON. The array processing systems employ a single control unit along with a single instruction stream to serve a group of processing units.

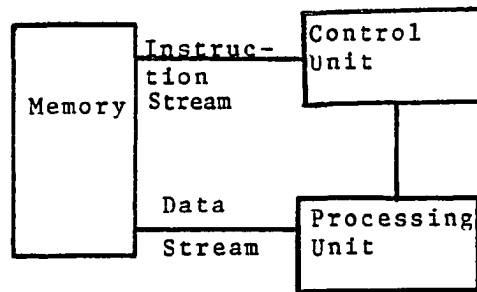
## 3. MISD:

Multiple-instruction stream - single-data stream organization. Some authors tend to include pipeline processors in this category. The data passes through different consecutive stages where in each stage a separate instruction stream is applied.

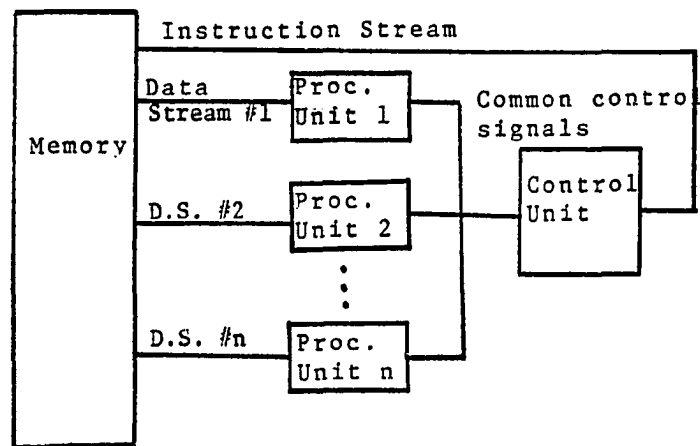
## 4. MIMD:

Multiple-instruction stream - multiple data-stream organization. This includes most multiprocessing organizations. Univac has proposed many different MIMD organizations.

The above classes could be quantified somewhat by specifying the number of streams of each type in the organization or the number of instruction streams per data stream or vice versa [FLYN72]. A pictorial of the four different organizations is shown in Figure 4.

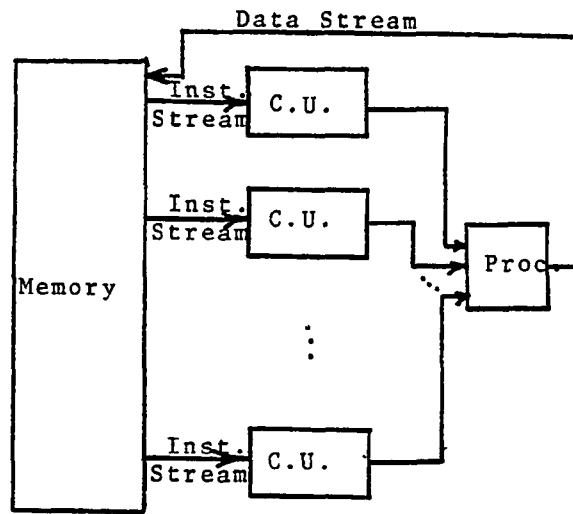


a) SISD

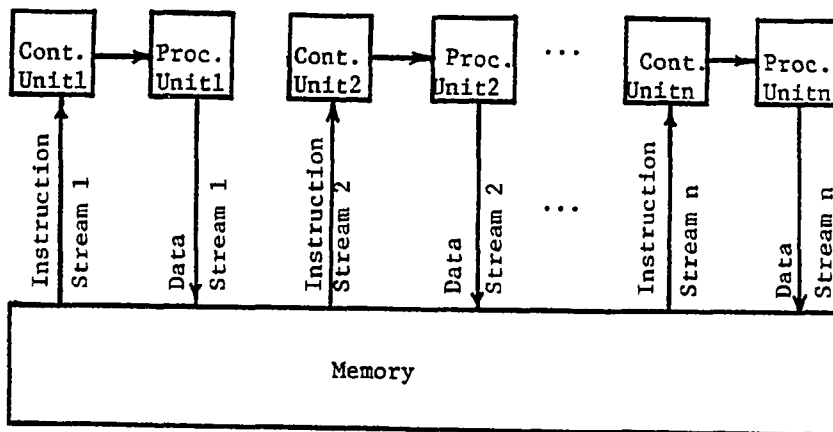


b) SIMD

Figure 4: The Four Flynn Classifications



c) MISD



d) MIMD

Figure 4 (Continued)

The limitation of the Flynn classification lies in the definition of MIMD. The MIMD is very broad in the sense that it does not mention the type of connection used, i.e., whether the processors are connected via a bus system or can access multiport memory. Furthermore, some authors tend to include various types of pipeline computers in the MISD class. It is inappropriate to do this since the different types of pipelining are not distinguishable.

#### 2.4.2 Handler Classification Scheme

Handler has proposed a different classification scheme for computing systems. In some regards, the Handler scheme is more explicit than the Flynn scheme. It is also known as the Erlangen Classification Scheme, (E.C.S.) [HAND77]. Each system is represented by a triple:

$$C = (K, D, W)$$

k = The number of control units

D = The number of ALU's controlled by each CU

W = The i-unit length of the entities managed by the D's

For example, for the following systems the triples are given as:

<u>System</u>	<u>Triple</u>
IBM System/370	(1,1,32)
CDC 6400	(1,1,60)
IBM System/360 dual processor	(2,1,32)

ILLIAC V	(1,64,64)
C.mmp	(16,1,16)

Further, the classification could be written in the form  $i x i'$  where  $i'$  indicates the parallelism or the number of pipeline stages in the  $i$ th component. Then, in  $K x K'$ ,  $K'$  is the number of independent computers of the same type of processing programs. In  $D x D'$ ,  $D'$  is the number of functional units or ALU's per processor. And in  $W x W'$ ,  $W'$  is the number of stages in a pipelined ALU.

Examples:

TI ASC	=	(1,4,64x8)
CDC 6600	=	(1,1x10,60)

Clearly the  $K$  and  $D$  parameters would indicate the type of computer system. For example,  $K=1$  and  $D=1$  would be equivalent to an SISD structure, and  $K=1$  and  $D=n$  where  $n.1$  would be an SIMD system. The Handler classification suffers from the following limitation: It does not explicitly or implicitly define the kind of parallelism used. It only specifies the number of units of each kind and does not mention the interconnection. For instance,  $C=(2,5,-)$  signifies a system consisting of two CU's each having five PE's, as indicated in Figure 5.

In this regard, the Flynn classification is more specific. The other problem from which the ECS suffers is the inherently binary nature of the definition of  $W$  (the



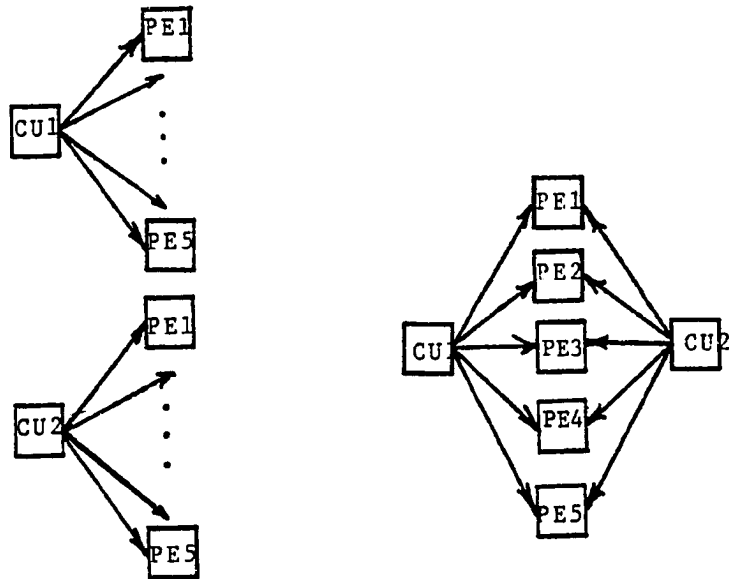


Figure 5: The Possible Diagram for  $C=(2,5,-)$

word length). That is, if a computer is based on another modulo number system, then the ECS should be modified.

## 2.5 EXAMPLE OF EACH ORGANIZATION TYPE

The currently existing parallel machines are pipeline machines, array machines, associative processors, cellular machines, and multiprocessors. The first three machines are useful for a restricted class of problems, whereas the fourth is extremely hard to program and as a result is not very popular. Different multiprocessing systems exist, but the software for these processors takes a considerable effort to build.

The major problem confronting the parallel systems in general is how to use such systems as efficiently as possible, i.e., utilizing the full power of the PE's. At present, technology can provide the necessary hardware to develop a super multiprocessing system, but such a system is futile without the required software. The efficiency (throughput) of a computer system can be increased by intensifying the usage of system resources which are either active or passive. Active resources such as processors (both for data and I/O) perform the calculations and move information to other parts of the system; passive resources such as memories, registers, and bulk storage devices hold information produced by the active resources for later use. Furthermore, as the number of PE's increases, the processor-to-memory interconnection grows and becomes more complex. The interconnection problem is an extensive

research area by itself and will not be discussed here. The December 1981 issue of COMPUTER magazine is devoted to the problem of interconnection in computer networks.

In this section we will discuss two types of organizations: the pipeline and the array system. The array system will be discussed in more detail in chapter VI. Somewhat detailed analysis of pipeline system is done in this section. Moreover, more examples of multiprocessing systems are found in [ENSL74].

Recently two articles have been written that are totally devoted to the bibliographies on the subject of multiprocessing systems: [LOUI81], and [SATY80]; more than 250 citations are referenced in these articles, and are considered a valuable source in the subject.

### 2.5.1 Pipeline Processor

The basic philosophy in pipelining is to break up the task into a number of subtasks which in turn are operated on in a manner similar to the assembly line technique. In effect the system is divided into several functional units, where each unit is assigned a specific task. A speed-up factor of more than two orders of magnitude can be obtained compared with the uniprocessor. Typical pipeline systems are uniprocessor systems with concurrent SISD organization. Some examples of pipeline systems are the Control Data

Star(CDC Star-100) and the Texas Instrument ASC system. The IBM 370-195 employs a pipeline floating-point multiply unit. In TI ASC and CDC STAR-100 pipelining is employed to perform the same arithmetic operations on a series of operands as they progress down the pipeline. Keeping the pipeline full is the main desire in exploiting the system characteristics. Since the pipeline system is not modeled in this study, it is worthwhile at this point to get some insight into the basic performance of a typical pipeline organization. We will investigate the throughput as a function of the number of stages and the jobs in the system.

A pipelined process is decomposed into a series of sequential subprocesses and each subprocess uses one stage of the pipeline. Each stage is isolated from its neighbors; therefore, overlapping will occur. For example, consider a pipeline system consisting of four stages. Starting with the system empty, then at  $t=t_1$ , stage 1 is busy on the first job, whereas stages 2-4 are idle. At  $t=t_2$ , stage 2 will be busy serving job 1 while stage 1 is receiving process 2, and so on. The diagram below illustrates this mechanism.

Let  $k$  = the number of processes waiting in the system,

$n$  = the number of stages (or system capacity),

and let  $k=n=4$  for this particular example.

Then the total execution time is given as

$$\begin{aligned} T &= n \times t + (k-1) \times t \\ &= 4t + 3t = 7t \end{aligned}$$

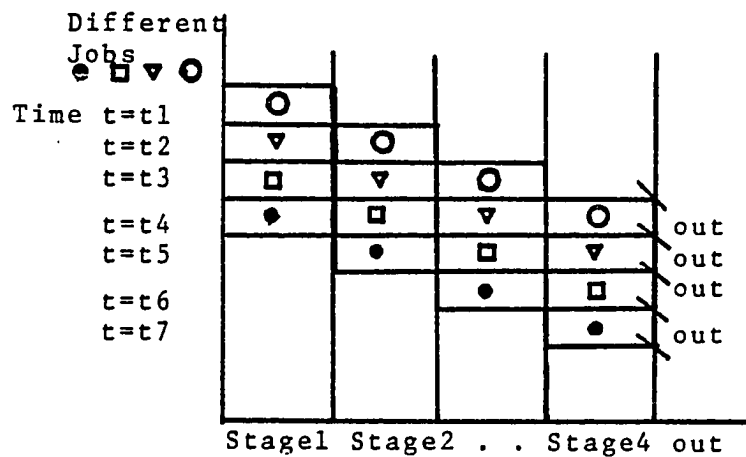


Figure 6: A Four-Stage Pipeline System

or  $(7/4)$  time units per stage. Therefore, it will take 7 time units to flush out the system. Comparing this pipeline system with one that consists of only one stage, with each process taking 4 time units, it will take 16 time units to process all four jobs. A considerable gain is achieved in the throughput for the pipeline system. The improvement factor is over 50% for this simple case. Note that the overall throughput performance depends also on the availability of jobs in the system.

The curves of Figures 7 and 8 illustrate that the throughput is directly proportional to the number of stages. The assumption that is made in all of the above cases is the

availability of jobs. In Figure 7, the curves show the throughput of a pipeline system versus the number of stages in the system. Since the principle time unit is the same for all the cases (i.e., the time a process spends from stage 1 to stage n is the same), it should be expected that as the number of stages increases then the probability of overlapping would increase. Thus the utilization of the stages would increase, which in turn results in a higher degree of throughput. However, Figure 8 illustrates the relationship between the percent throughput and the number of tasks waiting in the system.

### 2.5.2 Array Processing

In the SIMD system, as mentioned before, the control unit CU dispatches the instructions to the processing elements PE's; consequently, all active PE's execute the same instruction simultaneously. Each active PE executes the instruction on its own data in its own memory. The interconnection network provides communication among the processing elements. This type of machine structure is designed to exploit the parallelism of tasks such as vector and matrix-like problems.

The array systems, however, do have some drawbacks which we will mention in brief. The major drawback of the existing array systems is their ineffective use of the

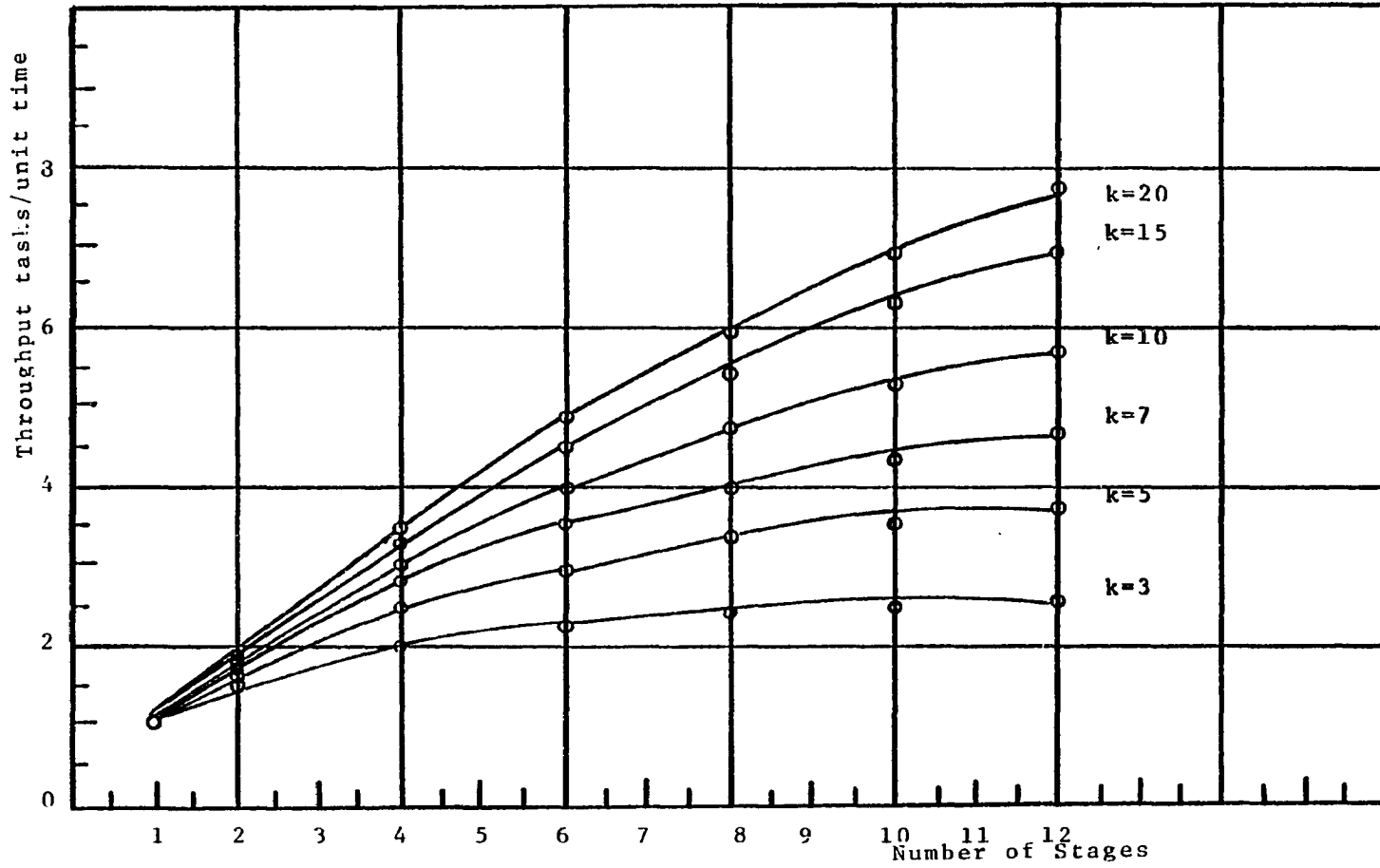


Figure 7: The Throughput Versus the Number of Stages in a Pipeline System

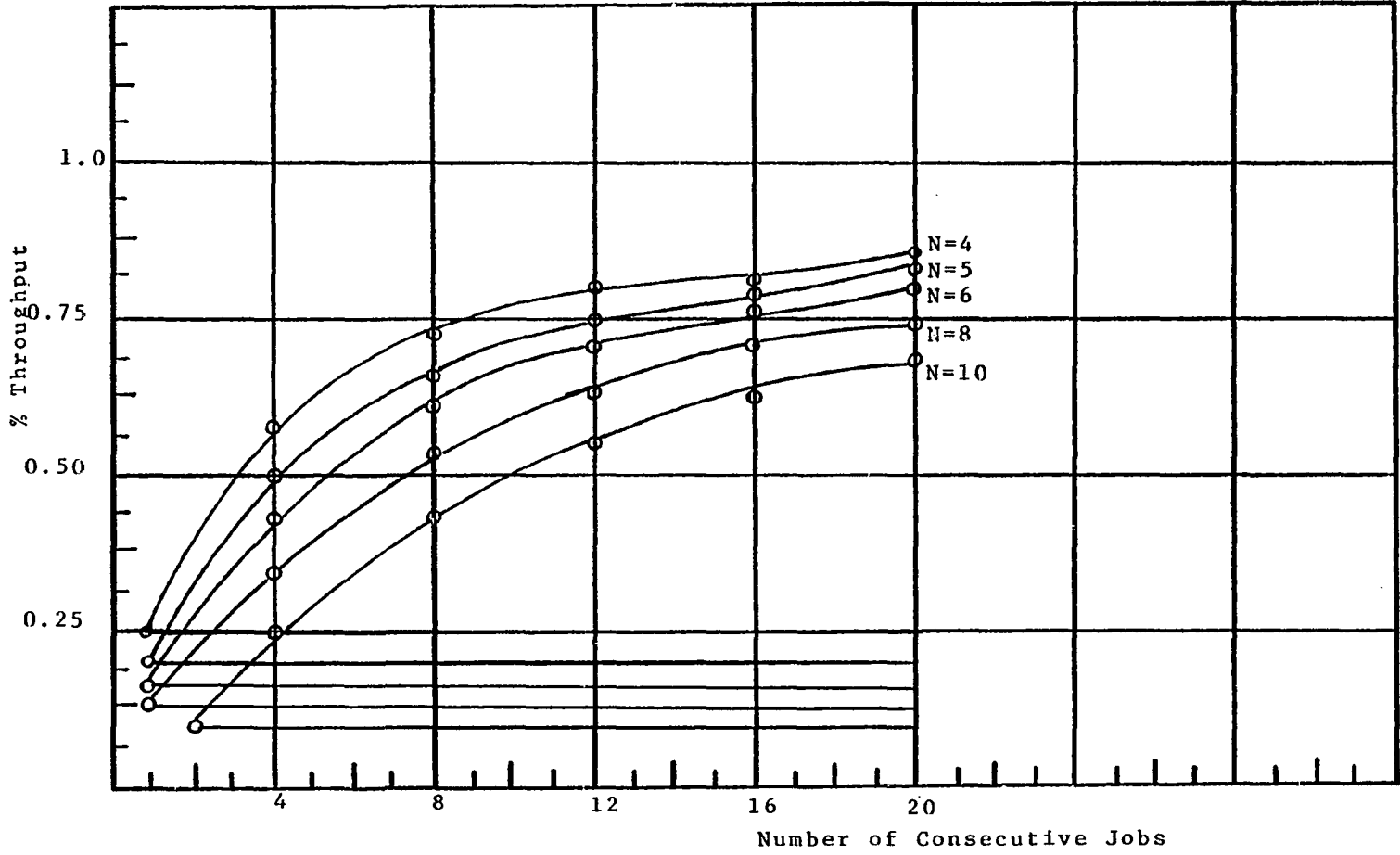


Figure 8: The Relation Between the Percent Throughput and the Number of Tasks Waiting



available hardware resources. This fact is due to the synchronous operation of all the processors working in the array. The number of parallel data streams are not fixed, and in the case where the number of data streams decreases, there will be some idle processors. Thus far it has not been possible to put the idle processors to work ~~REDD76~~. Furthermore, the cost of such systems is quite high. It is worthwhile to note that the typical number of processing elements in an array system is greater than 64, whereas in the pipeline processor system, the typical number of processors is 1. The basic configuration of an array processor is shown in Figure 9.

The first work done in the array processing area was on SOLOMON I and II. This work led to the ILLIAC IV system. SOLOMON contains 1024 processing elements in an array of 32x32 PE's. All the PE's are under the control of a single control processor. On the other hand, the ILLIAC IV includes 256 PE's, each more powerful than the PE's of SOLOMON. The 256 processing elements are arranged in four quadrants, of 8x8 PE each, with a separate CU for each quadrant. Each PE has a private memory of 2k 64-bits words. Figure 10 illustrates the organization of the complete ILLIAC IV computer system.

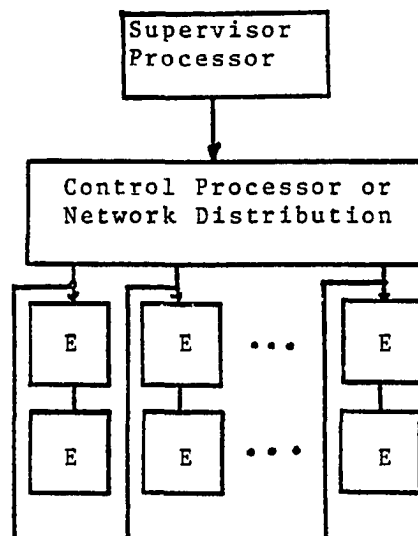


Figure 9: Basic Configuration of an Array System

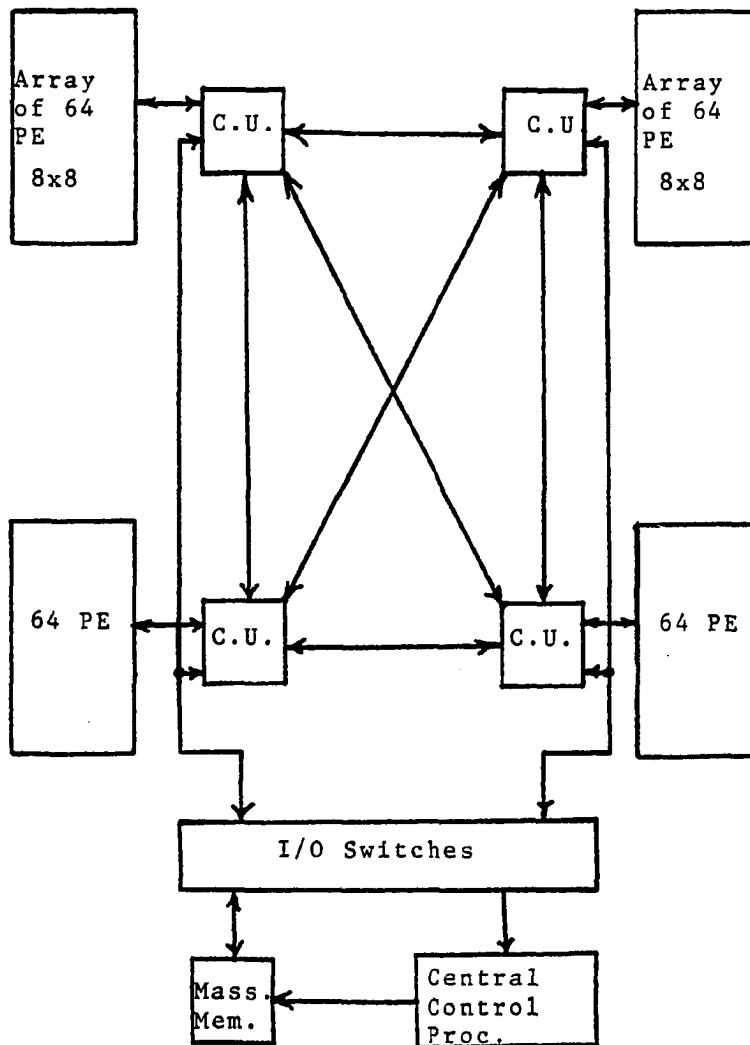


Figure 10: The Organization of the Complete ILLIAC IV Computer

## Chapter III

### A MODULAR BIT-SLICE PROCESSING ELEMENT (PE)

#### 3.1 INTRODUCTION

The NMSU modular bit-slice emulator ( MBSE ) which the author helped develop will be used as the basic processing element in our investigation of parallel multiprocessor architectures. Due to its microprogramming ability, the NMSU-MBSE system provides an excellent machine to be used in the various models proposed.

Bit-slice microprocessors in general are more versatile than the single chip microprocessors. On the other hand, they are more difficult to program. Their use is often somewhat limited to the application for which they are designed. But if they are designed to emulate another processor, then the user should see little difference between the real processor and the emulated one except in instruction execution time. Bit-slice microprocessors are known as variable instruction set microprocessors, in contrast to the fixed instruction set microprocessors. The flexibility associated with the bit-slice machines makes them very desirable in certain engineering applications.

Emulation is a combined hardware-software (firmware)

approach to the process of modeling the characteristic of machine Y (the target) on machine X (the host). The code in machine X makes it appear as machine Y to the user. Consequently, the user can write the software for machine Y by using only machine X. In other words, emulation is a complete set of instructions which, when stored in the control store of a bit-slice microprocessor, defines a new machine.

Due to their microprogramming ability, bit-slice elements play a significant role in process emulation. In this chapter, the basic elements of a general purpose bit-slice emulator designed at New Mexico State University are discussed and explained in some detail.

Due to their nature, bit-slice elements play a significant role in microprogramming. Husson [HUSS70] proposed the following definition for microprogramming:

"Microprogramming is a technique for designing and implementing the control function of a data processing system as a sequence of control signals, to interpret fixed or dynamically changeable data processing functions. These control signals, organized on a word basis and stored in a fixed or dynamically changeable control memory, represent the states of the signals which control the flow of information between the executing functions and the orderly transition between these signal states,"p 20. A description of recent applications of microprogramming is found in

[RAUS80]. Furthermore, the advantages and disadvantages of microprogrammed implementation compared to hardware implementation is nicely outlined. Two microprogramming techniques can essentially be specified:

- 1) vertical microprogramming
- 2) horizontal microprogramming

In vertical microprogramming, a shorter field is used, but it takes many microinstructions to accomplish the desired functions, whereas in horizontal microprogramming, larger fields are used and thus fewer microinstructions are required to perform the necessary function. The latter, of course, has a disadvantage: the microbits are not used as efficiently as in the former. Thus, horizontal microprogramming is not very economical when compared with the vertical microprogramming. Nevertheless, the horizontal microprogramming is used whenever speed is of concern and importance. Maximal parallelism at hardware level can be exploited by horizontal microprogramming. Generating the microinstructions can be cumbersome and sometimes time consuming.

In the NMSU-MBSE design horizontal microprogramming is used. The microinstruction length is 108 bits long. Out of the 108, 104 bits are pipelined and the other 4 bits come directly out of the control store and are used to control the 4-phase clock chip Am2925. The microinstruction fields are shown in Figure 11.



### 3.2 THE ALU UNIT

A single ALU board constitutes an 8-bit-wide data bus. Each additional ALU board added to the system increases the data bus bits by a factor of 8. The ALU'S are designed in such a way that when connecting more than one ALU together, the proper signals are generated, which in turn specify which board is the most significant slice (MSS), and which is the least significant slice (LSS). In effect, each board has two neighbors, right and left. If a board does not have a left neighbor, then it is the MSS, and if it does not have a right neighbor, then it is the LSS. Consequently, if a board has neither left nor right neighbor, it is considered to be the MSS and LSS at the same time, e.g., as in the 8-bit machines.

#### AM2903:

The ALU chip used in this design is the AMD AM2903, a 4-bit slice. Each board has two AM2903's. The following characteristics are summarized for the AM2903:

1. Independent access to two different registers
2. Performs 16 arithmetic and logical functions
3. Left or right shift independent of ALU
4. Has four status lines: carry, overflow, zero, and negative
5. Horizontally expandable to any word length
6. Has a 16 words by 4 bits registers with two ports, expandable to any number of registers.



Figure 12 illustrates the basic organization of the bit-slice design employed. Since the processing elements used in the different models of chapters V, VI, and VII are groups of independent PE'S, it is necessary to move the status, shift, and carry control unit (the SSC) from the central controller to each of the PE boards. After this modification the general block diagram will be that of Figure 13. For an 8-bit processor, only one ALU board and one sequencer board are needed. For a 16-bit processor, two such boards are needed along with one sequencer board, and so on.

The additional 16 register bank (the AM29705) is used in order to increase the total number of registers from 16 to 32 registers. When two boards are cascaded, then multiplexers 1 and 2 in Figure 13 will bypass the AM2904 on the L.S. board.

### 3.3 THE CONTROL UNIT

In a bit-slice design the control unit performs the function of sequencing through the microinstructions. The microsequencer, the AM2910, is considered the heart of the control unit.

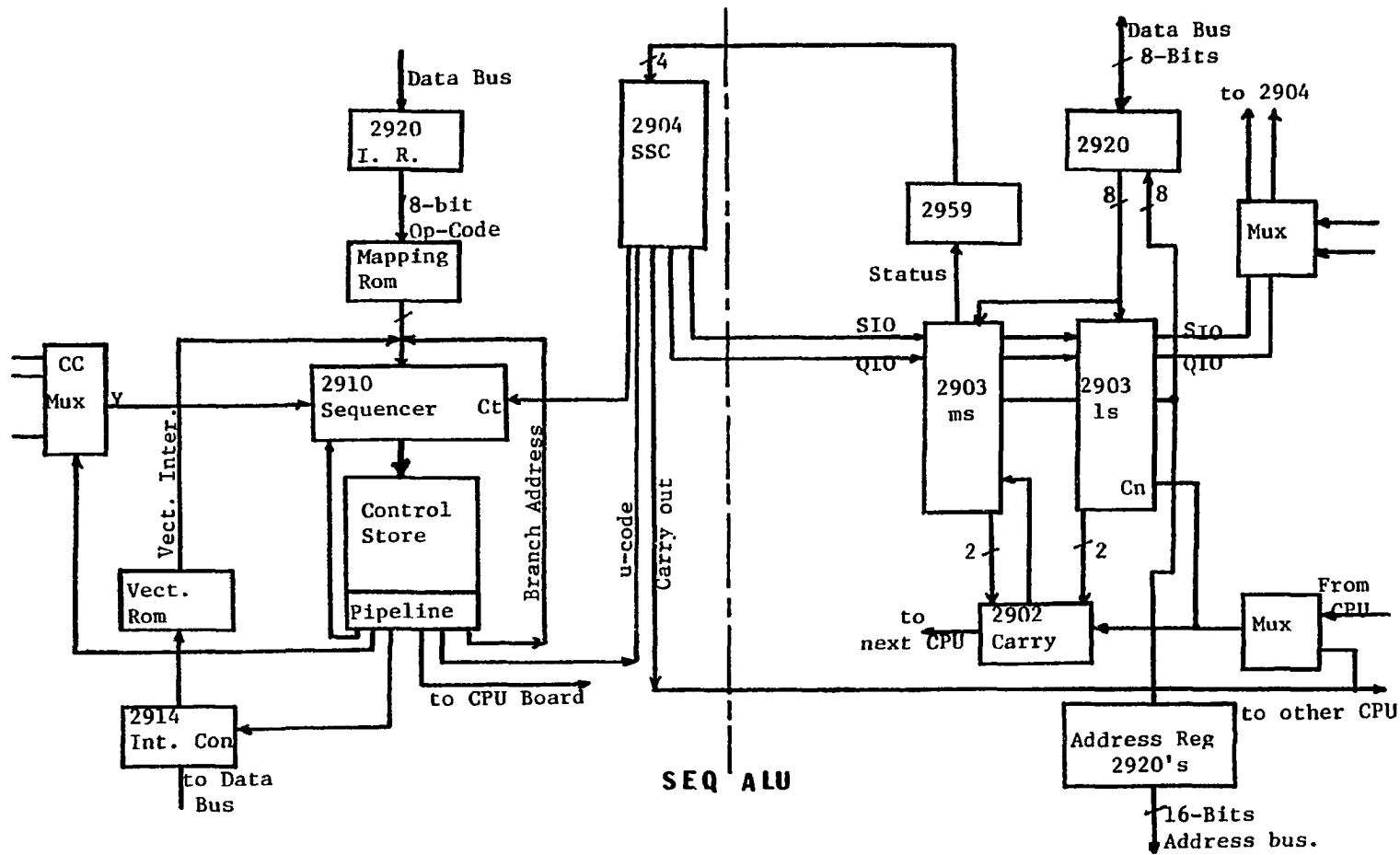


Figure 12: The Basic Bit-Slice Design

### 3.3.1 The Sequencer Unit (AM2910)

The sequencer chip, AM2910, has an address capability of up to 4K of control store words. The AM2910 essentially consists of a microprogram counter and an instruction-decoding programmed logic array on one chip. In addition to decoding instructions, the PLA provides three output signals that can be used to enable any of the three sources of the 2910's D inputs. The register counter in the AM2910 may be used to store a branch address that is used for a subroutine call. The AM2910 has a unique three-way branch instruction that is useful at the end of loops. If the input test condition is true (CT), then incrementing the program counter will cause an exit from the loop. But if the test condition is false, the loop counter is decremented and the program branches back to the top of the loop until the counter is zero, and then branches once more to the address specified on the direct input lines. The AM2910 makes the use of the 2909 and 2911 obsolete except for rare applications where more than 4K range is desired. More discussion about the AM2910 is found in [MICK78].

The 2910 provides a powerful set of instructions. It is well suited for a high performance computer control unit. The basic block diagram of the AM2910 is shown in Figure 14.

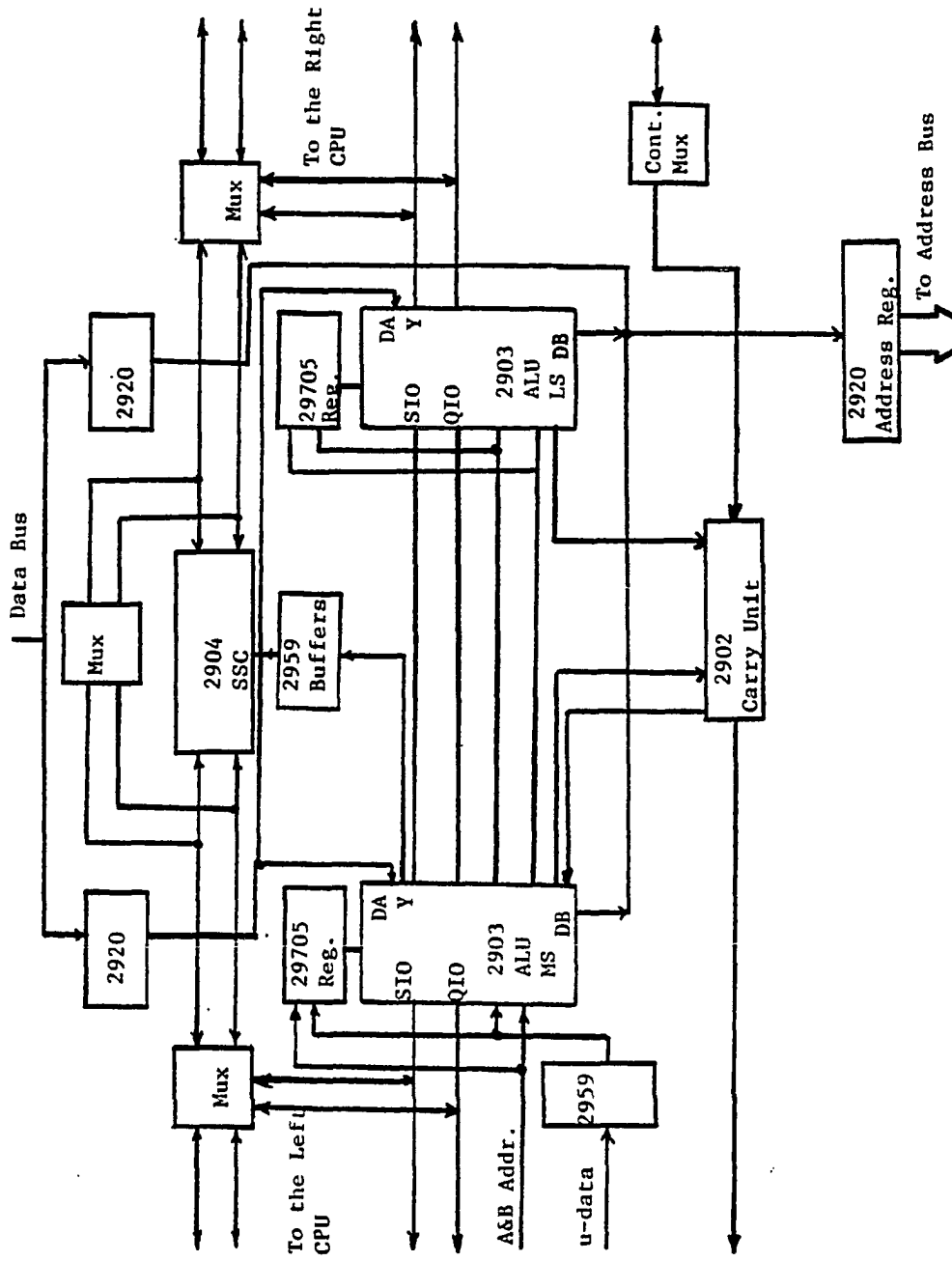


Figure 13: The Modified ALU Board

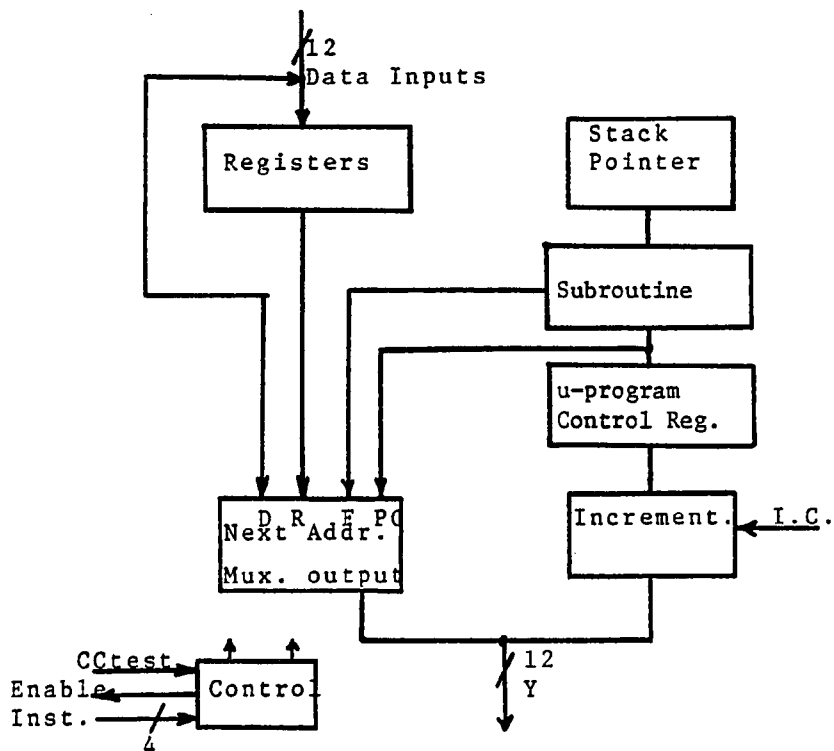


Figure 14: The Basic Block Diagram of the AM2910

### 3.3.2 Operation of The Control Unit

The macroinstructions are loaded into the Am2920 instruction register (IR). The operation code (op code) is used as an input to the mapper Roms (AM27S13'S), which are each 512X4. Up to 256 different op codes can be specified, since the AM2920 provides 8 output lines. Three AM27S13's are used to provide the 12 line inputs to the sequencer controller (AM2910). The AM2910 receives inputs from three different places, as follows:

- 1) Output of the mapping Roms
- 2) Output of the vector Roms
- 3) The branch address field of the microcode.

Depending on the test condition input to the 2910, which comes from the condition code Mux AM2922 or from the AM2904 (the SSC), then the AM2910 will decide whether to sequence through or to take the branch address.

### 3.3.3 The Control Store

The other major block on the sequencer board is the control store. The control store consists of 1K by 108 bits. As mentioned earlier, only 104 bits are fed to a pipeline register which is included in the AM27S27. The control store capacity can be increased to up to 4k if needed. The need for a larger control store arises when it is necessary to have the microcode of more than one target machine implemented.

### 3.3.4 Condition Code Mux (CC)

Since the status, shift, and carry control unit (2904) will be used to take care of overflow, zero, carry, and negative conditions (the outputs of the MS 2903 on the MSS), and produce a test signal to the Am2910, then other testing means should be provided for the other test conditions. A status register Am25LS377 along with a CC Mux Am2922 are used for this purpose. The input signals to the 25LS377 will include the interrupt request (IR) signal from the AM2914, reset, and the FULL signal output of the AM2910 plus any other test conditions. Both the registers and the CC mux are controlled by the microbits.

### 3.3.5 The Interrupt Control Unit (AM2914)

The AM2914 is an 8-bit priority interrupt circuit and is cascadable to handle any number of priority interrupt request levels. It implements an 8-bit mask register to mask individual interrupts. Only eight levels of interrupts are implemented in this design. Four microbits are specified for the operation of the AM2914.

The vector output of the AM2914 is connected to the three vector Roms (AM27S21), which are 256x4 each. The 12-bits output by the vector Roms are fed to the D-inputs of the 2910 along with the other two input sources. The interrupt request signal is fed to the CC mux, which in turn provides

a test input signal to the 2910 ( the CC input ), as mentioned earlier.



Chapter IV  
QUEUEING AND SIMULATION CONCEPTS

4.1 INTRODUCTION

In recent years, performance analysis has taken a major role in the design of computer systems. In the past two decades, several important contributions in the field of computer performance evaluation have been made [ROBL81]. The performance evaluation of computer systems may be divided into two broad categories. At the one end are the empirical studies. This covers techniques such as measurements and simulation. At the other end are the analytic methods. This covers techniques such as queueing models which depend on obtaining mathematical equations to analyze the system. The queueing models may yield some qualitative insight into the system behavior, but they cannot be trusted to provide quantitative insight to drive the architecture of the system in the desired direction [KUMA80]. The simulation technique suffers from the cost of carrying out such experiments. On the other hand, the analytic technique suffers from the time spent developing these equations. Kumar [KUMA80] gives a detailed hierarchical approach to performance evaluation of computer

systems.

Moreover, there exist a number of different algorithms to analyze the performance of queueing systems. Some of these algorithms are the mean value analysis algorithm [REIS80], the approximation technique algorithm [SAUR75], and the numerical methods. These mathematical algorithms are often used to analyze large computer systems and networks. In large computer systems, however, there is one factor that plays a significant role in analysis and this is the degree of multiprogramming. The degree of multiprogramming is not of great importance when analyzing systems of multi-microprocessors. The degree of multiprogramming tends to complicate the situation even more. With a small degree of multiprogramming the number of states grow enormously, namely,

$$\text{The number of states} = \binom{M+N-1}{M+1} \quad 4-0$$

where M equals the number of devices and N is the degree of multiprogramming. In a multiprocessing system implemented with microprocessors, more than one processor is used at a time, but always with one user or with one program in the main memory executing at any given time.

In summary, the analytic models should be used to study the effect of varying system parameters over a wide range, while simulation models should be used for more accurate

analysis of a specific configuration.

In this chapter the necessary background in analytic and simulation technique are considered. These basic concepts will provide a clearer understanding of the chapters that follow.

#### 4.2 QUEUEING PRINCIPLES

After the mathematical model of a queueing system has been formulated (by specifying all its assumptions), the model may be studied analytically in order to better understand the behavior of the system. Under certain conditions, a queueing system that has been in operation for a sufficiently long time settles down to a behavior independent of time. The system is then said to be in an equilibrium (steady state) condition. At the steady state, the following holds:

Jobs into the queue = Jobs out of the queue

The steady state is more convenient for system analysis.

The shorthand notation  $(A/B/C):(D/E/F)$  is used to describe any general queueing system, where A,B,C,D,E, and F are defined as follows:

- A represents the statistical characteristics of the customers arrival rate
- B represents the statistical characteristics of the server's service time
- C is the number of the servers in the system,  $1 < C < \infty$
- D is the service discipline
- E is the restriction (if any) on the maximum capacity of the system (in queue + in service)
- F is the size of the population from which the customers are drawn, typically finite or infinite.

A typical notation would be  $(M/M/1):(FIFO/\infty/\infty)$  where M designates that the distribution is Poisson type. Throughout the analysis, the above notation will frequently be used.

#### 4.2.1 Basic Relationships

The basic queueing parameters that will be used are summarized in Figure 15 below. The performance measures such as throughput, utilization as a function of the number of processing units, and the expected number in the system will be considered. The following relationships are defined:

$$\text{Mean service time per job} = E[x] \text{ sec/job} \quad 4-1$$

$$\text{then, Mean service rate} = 1 / E[x] \text{ jobs/sec} \quad 4-2$$

If the U (utilization) is defined as the fraction of time the resource is busy, the throughput must be equal to the service rate of the resource, when it is busy, times the fraction of time it is busy, i.e.,

$$\text{Throughput} = T = U \cdot (1 / E[x]) \quad \text{where } 0 < U < 1 \quad 4-3$$

Thus, for a given service rate, the higher the utilization, the higher the throughput will be. It is clear that for a utilization of 100% the throughput equals to the service rate, i.e.,  $1 / E[x]$ .

Furthermore, if there are k identical units of the same resource each with a utilization U, then the total throughput is given as

$$\text{Throughput} = k \cdot U / E[x] \text{ jobs/sec} \quad 4-4$$

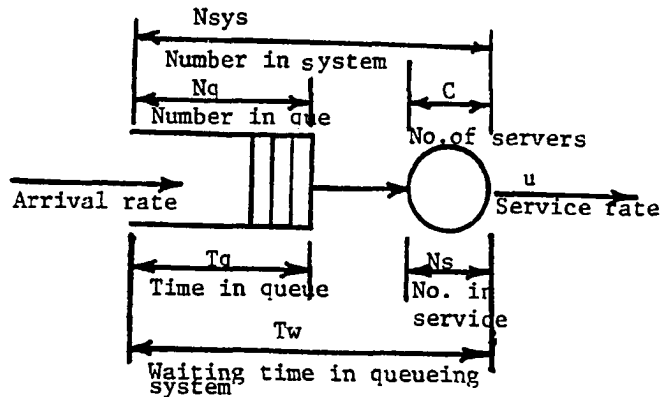


Figure 15: Some Random Variables Used in the Queueing System

The mean queue length and the queueing time are defined as follows:

$$Nq = L = E[q] = \sum_{l=1}^{\infty} lP(l) \quad 4-5$$

where  $P(l)$  is the probability that the queue has  $l$  job,

and the mean queueing time is defined as

$$Tq = E[q] = \frac{1}{\lambda} \sum_{l=0}^{\infty} lP(l) \quad 4-6$$

The mean queue length ( $Nq$ ) and time  $Tq$  are related by the well-known formula known as Little's Rule, i.e.,  $Nq = \lambda Tq$ , and it is proven in most queueing systems textbooks.

#### 4.2.2 Markov Process

In most queueing models made for computer systems analysis, the Markov process concept plays a predominant role. The customer arrival at a past or future instant does not affect the arrival or non-arrival at the present time. This lack of dependence on the past (and future) is commonly called the Markovian, or memoryless, property. That is, each event is acting independently.

A more formal definition of the Markov process is the following: A set of random variables  $[X_n]$  forms a Markov chain if the probability that the next value (state) is  $x$  depends only upon the current value (state)  $x$  and not upon any previous values. That is, the entire past history, which affects the future of the process, is completely summarized in the current state of the process.

Analytically, the Markov process is expressed as

$$\begin{aligned}
 P[X(t_{n+1})=x_{n+1} | X(t_n)=x_n, X(t_{n-1})=x_{n-1}, \dots, X(t_1)=x_1] \\
 = P[X(t_{n+1})=x_{n+1} | X(t_n)=x_n] \\
 t_1 < t_2 < \dots < t_{n+1}
 \end{aligned}
 \tag{4-7}$$

The arrival time and service time distribution used in this study are the Poisson and the exponential distribution, respectively. The Poisson arrival is simpler to treat mathematically than other arrival distributions.

The state transition concept used in developing the models is defined by the following formula:

$$\begin{aligned}
 P_{ij} = P(x_1=j | x_0=i) = P(x_2=j | x_1=i) = P(x_3=j | x_2=i) \dots \\
 \text{Or } P_{ij} \triangleq P [x_n=j | x_{n-1}=i]
 \end{aligned}
 \tag{4-8}$$

It denotes the probability that a move is made from state  $E_i$  to state  $E_j$ .  $P_{ij}$  is called a transition probability. If  $M$  is a Markov chain with  $n$  states, the transition matrix will be an  $n \times n$  matrix. If each entry in the transition matrix is a non-negative, and the sum of each row adds up to 1, then the transition matrix is also called a stochastic matrix. Thus, every transition matrix is a stochastic matrix. The converse, however, is not true.

#### 4.2.3 Networks of Queues

When a queueing system can be composed of several interconnected nodes, and each node constitutes a complete queueing system (i.e., it has its own queue and server), then the overall system is referred to as a network of queues. Burke's theorem [BURK56] plays an important role in analyzing the network of queues. Burke's theorem states that in a stable queueing system, a Poisson process driving an exponential server generates a Poisson process for the departure. In other words, if the arrival distribution for the first node is Poisson, then the arrival distribution for the successor node will also be a Poisson. Another important fact that Burke's theorem states is the following: the steady-state output of a stable (M/M/m) queue with input parameter  $\lambda$  and output parameter  $u$  for each of the  $m$  channels is in fact a Poisson process at the same rate as  $\lambda$ .

These amazing concepts associated with Burke's theorem

will help us to a great extent when analyzing the data flow model of chapter VII.

Sometimes it is necessary to work with networks of several nodes where each node is an (M/M/m) system. The  $i$ th node, then, consists of  $m_i$  exponential servers each with parameter  $\mu_i$ . Upon service completion in the  $i$ th node, the customer then proceeds to the  $j$ th node with a probability  $P_{ij}$ . The total arrival to node  $j$  is given by

$$\lambda_j = \gamma_j + \sum_{i=1}^N \lambda_i P_{ij} \quad \text{for } j=1,2,\dots,N$$

where  $\gamma_j$  is the outside arrival rate. Note that some  $P_{ij}$ 's may be zero; this is true if node  $i$  does not feed its output to node  $j$ .

Jackson [JACK57] analyzed this situation and he showed that each node in the system behaves independently and can be considered an (M/M/m) system with Poisson input  $\lambda$ , even though the total input is not a Poisson process. Therefore, in an  $N$ -node system, the state variables consist of the vector  $(k_1, k_2, \dots, k_N)$  where  $k_i$  is the number of customers in the  $i$ th node including the one in service. Applying Jackson's theorem, the state variable vector is written as

$$P(k_1, k_2, \dots, k_N) = P_1(k_1)P_2(k_2)\dots P_N(k_N) \quad 4-9$$

where  $P_i(k_i)$  is the the solution to the classical (M/M/m) system and is given by

$$P_k = P_0 (m^k) / k! \quad k < m \quad 4-10$$

$$\text{and } P_k = P_0 (p)^m / m! \quad k > m \quad 4-11$$

Again in chapter VII, these concepts will help in



treating the different subsystems as nodes in a network of queues.

#### 4.3 SIMULATION PRINCIPLES

One of the most costly analysis techniques is simulation. Given a specific model, running a simulation program will always cost more than running its analytic counterpart. Because of the limitation of queueing models, simulation can be used in conjunction. Simulation models can accurately model more complex structures than the queueing models can. The flexibility associated with simulation provides them this property.

In this study, the simulation language GPSS (General Purpose Simulation System) is used to simulate the models under investigation. This language is quite suitable for simulating queueing problems. Furthermore, it has some advantages over the other high level languages such as Fortran and Pascal in queue management and is more compact. It has a built-in random number generator (R.N.G.). This R.N.G. can be controlled so that the different runs made for the different system parameters in each case are done with the same initial random number, thus controlling the simulation environment.

In all the models under study, the basic simulation blocks are similar. The flowchart shown in Figure 16 represents the basic simulation blocks. Except for some

specific situations where some modification has to be made, the skeleton of Figure 16 will be typical.

System initialization is necessary, especially if we are using memory places to be accessed by all the transactions. As mentioned earlier, initialization of R.N.G. is sometimes very desirable. The initialization is done for more accurate analysis and comparison of specific configuration.

In the GENERATE arrival block we control the desired frequency of arrivals, their type (uniformly or Poisson-distributed), the starting of arrivals, ... etc. Note that the first operand of the generate block is the average interarrival time (IAT). Therefore, if the interarrival function used is exponentially distributed, the arrival process is in fact Poisson. The terms transaction (xact), job, process, and macroinstruction will be used interchangeably to represent the customer in the system. With each xact, a number of parameters are associated. Each parameter can be used to indicate a particular function. For example, one parameter can be used as a counter to count the number of operands, and another parameter can have the destination address for the result (for the data flow case). The real advantages of the parameters are appreciated when using indirect addressing. As the xact flow down the model different service times (for controller, PE, etc) and the different queues are all computed and stored in the system memory for print-out. By the TERMINATE block we mean

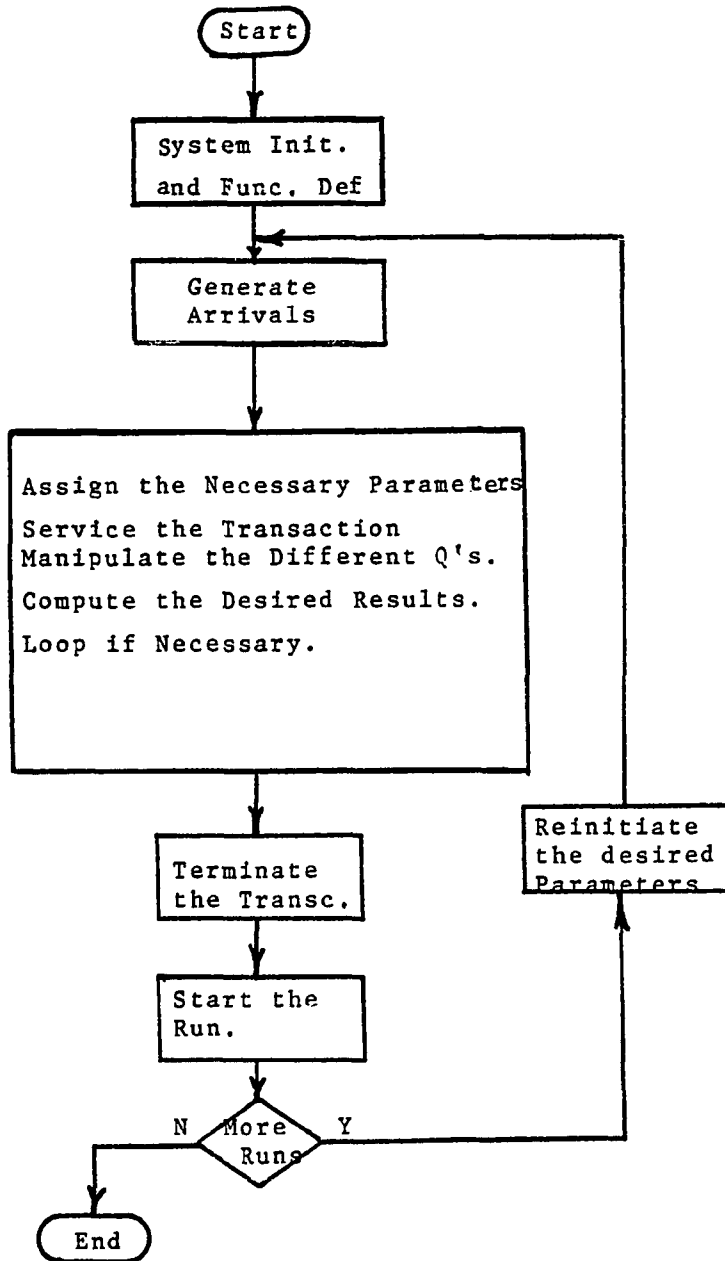


Figure 16: The Basic Flowchart for Simulating a Model

terminating that particular xact when it passes through this block. The START block initiates the run. Finally, the END block will terminate the program unless other runs are necessary for different configurations. For more information on the GPSS blocks used, the following references are recommended: [SCHR74], and [BOBI76].

## Chapter V

### THE CONTROLLED MULTISERVER MODEL

#### 5.1 INTRODUCTION

The controlled multiserver model represents an organization of processing units in such fashion that there is at least one PE for each operation code. Therefore, in executing a certain program where the operations addition, subtraction, multiplication, and division are extensively used, one or more PE could be allocated for each operation code. This will increase the reliability of the system.

#### 5.2 SYSTEM ORGANIZATION

The overall system organization can be envisioned as two parts communicating via a communication network in a master/slave environment. The master processing element (MPE) is located in one part, and the slave processing elements (SPE's) in the other. Figure 17 illustrates the principal blocks for the model. According to the notation used in chapter II, this system could be called an SIMSD (single instruction stream multiple single data stream) system.

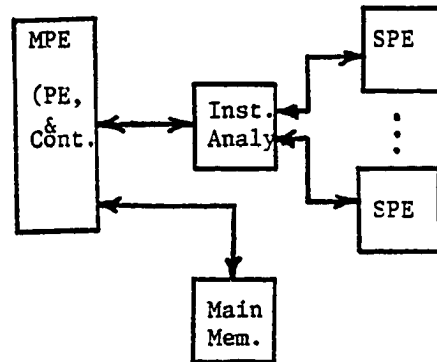


Figure 17: The System Block Diagram

The MPE can be considered as consisting of two parts, the processing elements and the controller. The microinstructions are stored in the control store of the controller. The macroinstructions (program code) are stored in the main memory, which is also part of the controller section. The PE is responsible for fetching macroinstructions from the main memory and routing them to the controller's instruction register. The op code portion of the instruction is used to address the microinstructions. The microinstruction should contain some control bits to direct the instruction analyzer (IA). Then depending upon the type of the instruction, the IA will enable the specified SPE. If the instruction is of the control type,

such as jump, subroutine call, etc., then the PE of the MPE is enabled. A more detailed organization of the system is shown in Figure 18.

### 5.3 SYSTEM ANALYSIS

The performance of this model is analyzed using queueing techniques and simulation methods. We will discuss the queueing model for this organization first; the state transition concept will be used to derive the analytic equations for the system. Once established, the state equations are used to calculate the different performance measures. The second technique used in this analysis is simulation using GPSS. Due to the flexibility associated with simulation techniques, some modifications will be performed in order to observe the effect on performance. Basically, we are interested in adding a load unit to each PE in order to store its microinstruction.

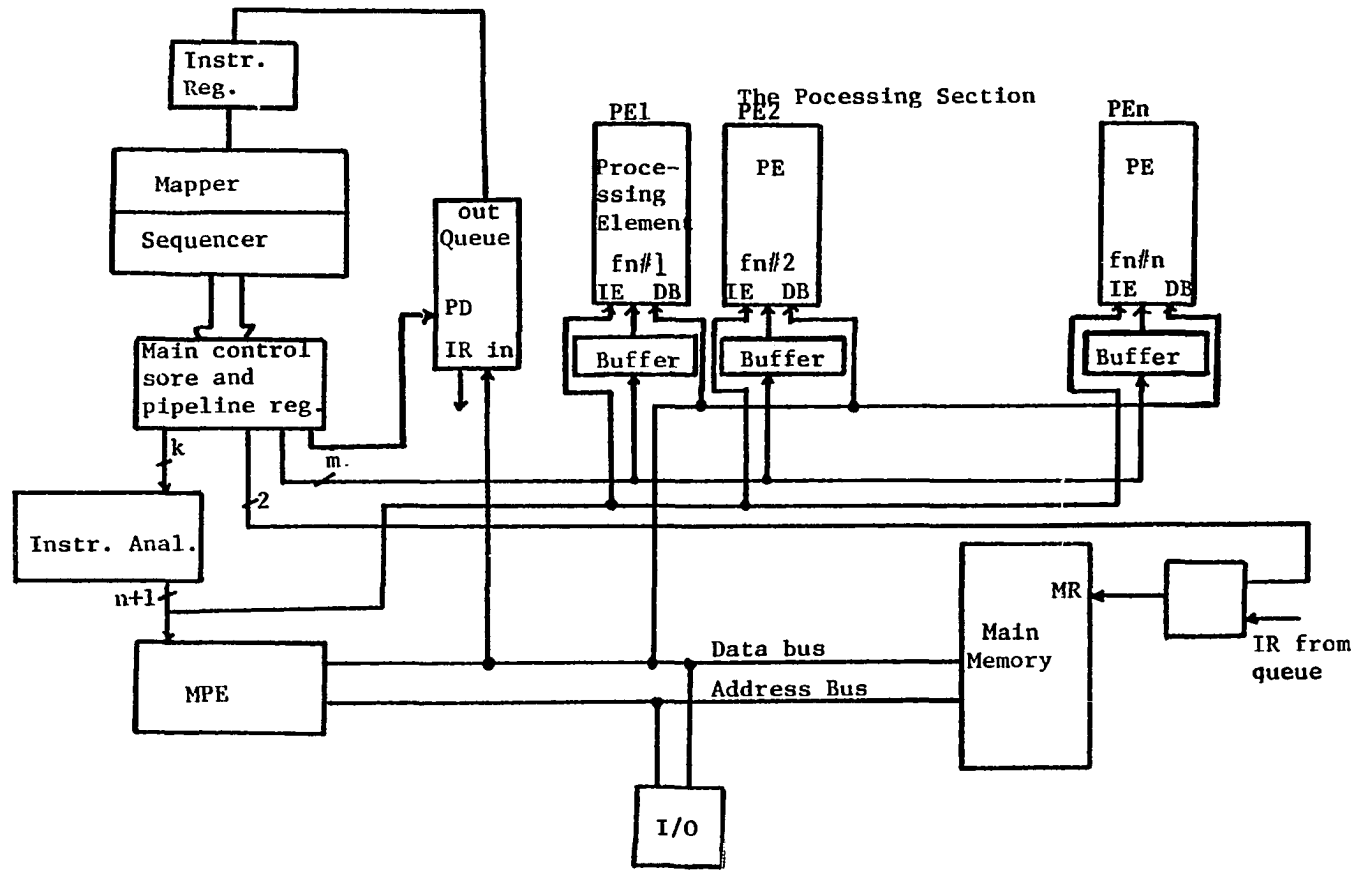


Figure 18: A Detailed System Organization



### 5.3.1 The Analytic Model

The basic queueing structure is shown in Figure 19.

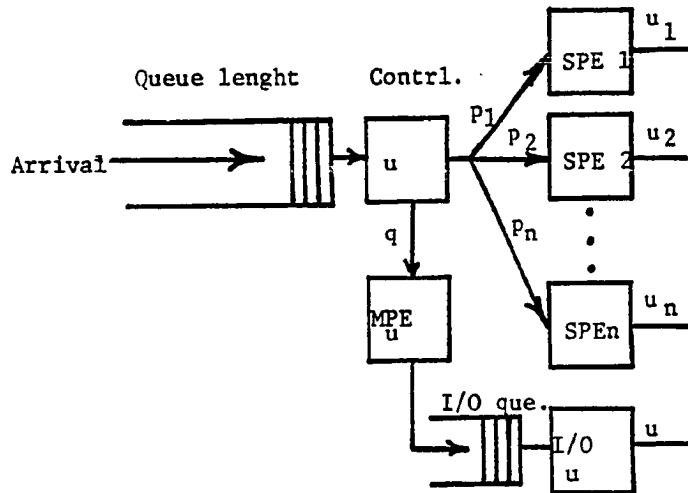


Figure 19: The Overall Queueing Structure

Clearly, it is not an easy task to analyze the model of Figure 19. Due to the complexity of this queueing model, we need to make certain assumptions. These assumptions will serve in easing analysis of the system without getting down to the fine details. As is generally true in queueing models, simple models of a particular system will not give erroneous results in comparison with more complex models for the same system [CHAN81]. The following facts are true for this system:

1. One job is executed at a time, i.e., where the master PE is busy serving a job, then this job will have the priority of getting the attention of all the processors (nonpreemptive).
2. The service time of the slave processors are measured and the overall average execution time is found to be

$$E(u) = \frac{1}{n} \sum_{i=1}^n \mu_i$$

where  $\mu_i$  is the service time for each of the slave processors, and  $n$  is the total number of PE's (this value will be used in the simulation case).

3. According to the analysis, only one SPE is enabled at a time. The slave processor will only perform one function, and while the SPE is performing that particular ALU function, the controller can not deal with the next macroinstruction.

Due to the overlapping between the operations of the SPE's, then each SPE should have its own status, shift, and carry control unit, (AM2904). This is the modified PE of chapter III.

The following assumptions are made for the queueing model so that the analysis can be carried out with less hardship. First the I/O device can be neglected, because the probability of executing an I/O instruction is always lower than the probability of executing any other instructions. Therefore, the error due to this will be negligible. The second assumption made is the possibility of combining the

service time of the controller with that of the SPE. This later assumption is justified by the following fact: In order for each SPE service time to occur, the service time of the controller ( $u_c$ ) must occur. Therefore, ( $u_i + u_c$ ) (where  $u_i$  stands for the service time for SPE $i$ ) is implemented. The model after these simplification will reduce to the one shown in Figure 20. The probabilities  $p_1$  through  $p_n$  and  $p_q$ .

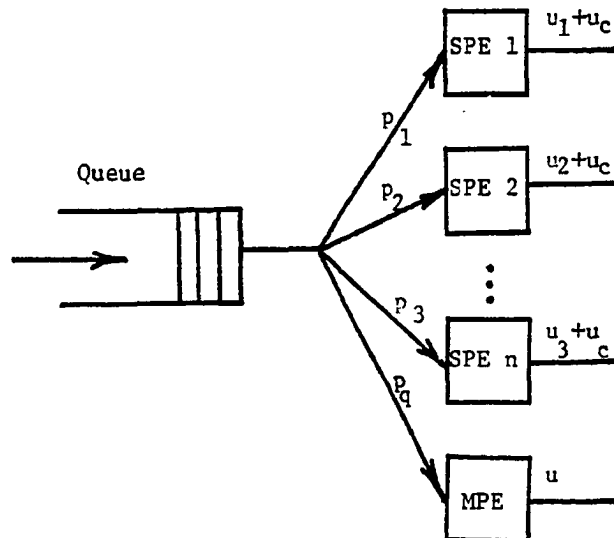


Figure 20: The Simplified System Model

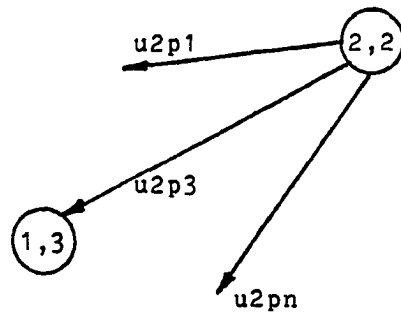
can be estimated by using a typical program that uses all or most of the functions (that is the SPE's).

This system (for the moment) can not be considered a truly parallel system, for not all the SPE's can be occupied

at the same time. More precisely, as long as the MPE's controller is busy serving one SPE, then other SPE's can not serve the waiting instructions. This latter fact differentiates this system from a normal multiple server system, and can be considered a restricted  $(M/M/n):(FCFS/c/\infty)$  model. When using simulation we will remove this last restriction and compare the results.

### 5.3.1.1 State diagram derivation

Initially the system will be empty, i.e., at state 0. As the first microinstruction arrives, depending upon the type of operation, Add, Sub, etc., a particular SPE will be selected. If for example the operation is an add, and the probability is  $p_2$ , then the next state in the state diagram is  $(1,2)$ . The notation  $(f,g)$  is interpreted as follows:  $f$  instructions are in the system and SPE- $g$  is busy executing it. While in state  $(1,2)$  and another instruction arrives, the next state will be  $(2,2)$ , i.e., 2 instructions are in the system, and the one being served is using SPE-2. Now suppose a service completion occurs while the system is in state  $(2,2)$  then we will go to state  $(1,1)$  where  $i=1,2,\dots,n$ . Depending upon the probability  $(p_1, p_2, \dots, p_n)$  the next state will be decided, e.g., if the probability is  $p_3$ , then the next state is  $(1,3)$  as shown below.



The complete state diagram is presented in Figure 21.

### 5.3.1.2 Derivation of the state equations

We will derive the equations for the general case. Afterwards, some numerical examples will be used to illustrate the results. Let  $n$  be the number of PE's; and  $c$  be the capacity of the system. The rate balance techniques [KLEI75a] as pointed out in chapter IV, states that the flow rate into a node must equal the flow rate out of the node. This principle is applied to each node in the state transition diagram.

For state  $P_0$ :

$$P_0 \lambda = P_{11} \mu_1 + P_{12} \mu_2 + \dots + P_{1n} \mu_n \quad 5-0$$

and for states  $P_{11}, P_{12}, \dots, P_{1n}$  we have:

$$P_0 \lambda p_1 + P_{21} \mu_1 p_1 + P_{22} \mu_2 p_1 + \dots + P_{2n} \mu_n p_1 = P_{11} (\lambda + \mu_1) \quad 5-1$$

$$P_0 \lambda p_2 + P_{21} \mu_1 p_2 + P_{22} \mu_2 p_2 + \dots + P_{2n} \mu_n p_2 = P_{12} (\lambda + \mu_2) \quad 5-2$$

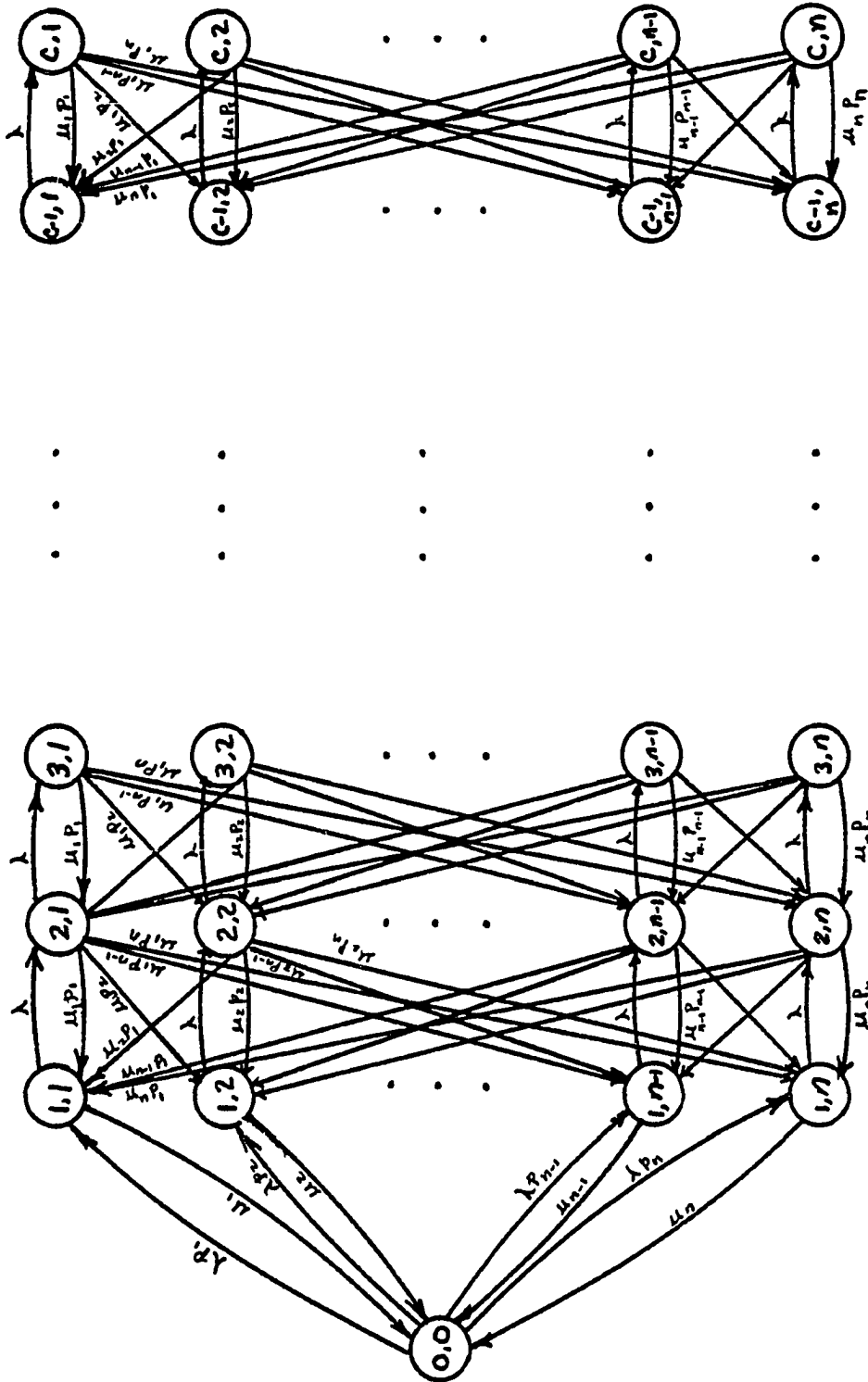


Figure 21: State Transition Diagram for the Controlled Multiserver System

$$P_o^\lambda p_n + P_{21}^\mu p_n + P_{22}^\mu p_n + \dots + P_{2n}^\mu p_n = P_{1n} (\lambda + \mu_n) \quad 5-3$$

rearranging equations 5-1, 5-2, and 5-3 ,

$$P_{21}^\mu p_1 + P_{22}^\mu p_1 + \dots + P_{2n}^\mu p_1 = P_{11} (\lambda + \mu_1) - P_o^\lambda p_1 \quad 5-4$$

$$P_{21}^\mu p_2 + P_{22}^\mu p_2 + \dots + P_{2n}^\mu p_2 = P_{12} (\lambda + \mu_2) - P_o^\lambda p_2 \quad 5-5$$

$$P_{21}^\mu p_n + P_{22}^\mu p_n + \dots + P_{2n}^\mu p_n = P_{1n} (\lambda + \mu_n) - P_o^\lambda p_n \quad 5-6$$

For states  $P_{21}, P_{22}, \dots, P_{2n}$  we have

$$\lambda P_{11} + P_{31}^\mu p_1 + P_{32}^\mu p_1 + \dots + P_{3,n-1}^\mu p_1 + P_{3n}^\mu p_1 = P_{21} (\mu_1 + \lambda) \quad 5-7$$

$$\lambda P_{12} + P_{31}^\mu p_2 + P_{32}^\mu p_2 + \dots + P_{3,n-1}^\mu p_2 + P_{3n}^\mu p_2 = P_{22} (\mu_2 + \lambda) \quad 5-8$$

$$\lambda P_{1,n-1} + P_{31}^\mu p_{n-1} + P_{32}^\mu p_{n-1} + \dots + P_{3,n-1}^\mu p_{n-1} + P_{3n}^\mu p_{n-1} = P_{2,n-1} (\mu_{n-1} + \lambda) \quad 5-9$$

$$\lambda P_{1n} + P_{31}^\mu p_n + P_{32}^\mu p_n + \dots + P_{3,n-1}^\mu p_n + P_{3n}^\mu p_n = P_{2n} (\mu_n + \lambda) \quad 5-10$$

Similarly, we can derive the equations for  $P_{31} \dots P_{3n}$  up to

$P_{c1} \dots P_{cn}$  using the same procedure.

Generally, it is possible to write the above set in a more compact form,

$$P_{1j} (\mu_j + \lambda) = P_o^\lambda p_j + p_j \sum_{i=1}^n P_{2i}^\mu p_i \quad 5-11$$

where  $j=1,2,\dots,n$ ,

$$P_{kj} (\lambda + \mu_j) = P_{k-1,j} \lambda + p_j \sum_{i=1}^n P_{k+1,i} \mu_i \tag{5-12}$$

where  $k = 2,3,\dots,c-1$  and  $j = 1,2,\dots,n$

and,

$$P_{cj} = (\lambda / \mu_j) P_{c-1,j} \tag{5-13}$$

The above general set of equations can be used to find any state probability. It is evident that the set is a recursive one. Due to the symmetry in the state diagram, one is encouraged to take advantage of matrix algebra. Consequently, by converting the above into matrices and rearranging we get:

$$\begin{bmatrix} \mu_1 P_1 & \mu_2 P_1 & \dots & \mu_n P_1 \\ \mu_1 P_2 & \mu_2 P_2 & \dots & \mu_n P_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mu_1 P_n & \mu_2 P_n & \dots & \mu_n P_n \end{bmatrix} \begin{bmatrix} P_{21} \\ P_{22} \\ \vdots \\ P_{2n} \end{bmatrix} = \begin{bmatrix} \lambda + \mu_n & 0 & \dots & 0 \\ 0 & \lambda + \mu_n & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda + \mu_n \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{12} \\ \vdots \\ P_{1n} \end{bmatrix} - \lambda P_c \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{bmatrix} \tag{5-14}$$

$$\begin{bmatrix} \mu_1 P_1 & \mu_2 P_1 & \dots & \mu_{n-1} P_1 & \mu_n P_1 \\ \mu_1 P_2 & \mu_2 P_2 & \dots & \mu_{n-1} P_2 & \mu_n P_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mu_1 P_{n-1} & \mu_2 P_{n-1} & \dots & \mu_{n-1} P_{n-1} & \mu_n P_{n-1} \\ \mu_1 P_n & \mu_2 P_n & \dots & \mu_{n-1} P_n & \mu_n P_n \end{bmatrix} \begin{bmatrix} P_{31} \\ P_{32} \\ \vdots \\ P_{3n} \end{bmatrix} = \begin{bmatrix} \mu_1 + \lambda & 0 & \dots & 0 & P_{21} \\ 0 & \mu_2 + \lambda & \dots & 0 & P_{22} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mu_{n-1} + \lambda & \dots & \dots & \dots & P_{2n} \\ \dots & \dots & \dots & \mu_n + \lambda & P_{2n} \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{12} \\ \vdots \\ P_{1n-1} \\ P_{1n} \end{bmatrix} - \lambda I \tag{5-15}$$

For convenience, let us make the following assignments,



$$\text{Let } A = \begin{bmatrix} \mu_1^{P_1} & \mu_2^{P_1} & \dots & \mu_n^{P_1} \\ \mu_1^{P_2} & \mu_2^{P_2} & \dots & \mu_n^{P_2} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_1^{P_n} & \mu_2^{P_n} & \dots & \mu_n^{P_n} \end{bmatrix} \quad \text{an } nxn \text{ matrix}$$

$$B = \begin{bmatrix} \mu_1 + \lambda & 0 & \dots & 0 \\ 0 & \mu_2 + \lambda & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & \mu_n + \lambda \end{bmatrix} \quad \text{an } nxn \text{ matrix}$$

and

$$L = \begin{bmatrix} \lambda & 0 & \dots & 0 \\ 0 & \lambda & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \dots & \lambda \end{bmatrix} = \lambda [I]$$

where  $[I]$  is an  $nxn$  identity matrix. Furthermore, at this point it is also necessary to define the  $U$  and  $P$  vectors. The  $U$ -vector ( $UV$ ) is the service rate vector and is given as:

$$UV = [\mu_1 \mu_2 \dots \mu_n]^T$$

whereas the  $P$ -vector ( $PV$ ) is the probability selection vector

and is given as:

$$PV = [P_1 P_2 \dots P_n]^T$$

The A matrix is then found by multiplying two matrices, where the first is formed by the elements of the P vector (PV) and the second is diagonal and is formed with the elements of the UV.

$$A = \begin{bmatrix} P_1 & P_1 & \cdot & \cdot & \cdot & P_1 \\ P_2 & P_2 & \cdot & \cdot & \cdot & P_2 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ P_n & P_n & \cdot & \cdot & \cdot & P_n \end{bmatrix} \times \begin{bmatrix} \mu_1 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \mu_2 & \cdot & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ 0 & 0 & \cdot & \cdot & \cdot & \mu_n \end{bmatrix}$$

The B matrix is formed as follows:

$$B = \begin{bmatrix} \lambda & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \lambda & \cdot & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ 0 & 0 & & & & \lambda \end{bmatrix} \times \begin{bmatrix} \mu_1 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \mu_2 & \cdot & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ 0 & 0 & & & & \mu_n \end{bmatrix}$$

where the diagonal matrix  $\lambda$  is of size  $n \times n$ . For the rest of the states, up to  $P_{c-1,i}$   $i=1, \dots, n$  the set of equations will look like Eq 5-15. And finally, for states  $P_{c,i}$   $i=1, \dots, n$

$$\begin{bmatrix} \mu_1^{P_{c1}} \\ \mu_2^{P_{c2}} \\ \cdot \\ \mu_n^{P_{cn}} \end{bmatrix} = \begin{bmatrix} \lambda^{P_{c-1,1}} \\ \lambda^{P_{c-1,2}} \\ \cdot \\ \lambda^{P_{c-1,n}} \end{bmatrix}$$

or

$$\begin{bmatrix} P_{c1} \\ P_{c2} \\ \cdot \\ \cdot \\ P_{c3} \end{bmatrix} = \lambda \begin{bmatrix} \mu_1 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \mu_2 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot & \cdot & \mu_n \end{bmatrix}^{-1} \begin{bmatrix} P_{c-1,1} \\ P_{c-1,2} \\ \cdot \\ \cdot \\ P_{c-1,2} \end{bmatrix}$$

5-16

Returning to 5-14 and 5-15, and solving for the state probabilities  $P_{ij}$ 's, we obtain the following general results:

$$\begin{bmatrix} P_{11} \\ P_{12} \\ \cdot \\ \cdot \\ P_{1n} \end{bmatrix} = [B]^{-1} [A] \begin{bmatrix} P_{21} \\ P_{22} \\ \cdot \\ \cdot \\ P_{2n} \end{bmatrix} + P_0 \lambda \begin{bmatrix} P_1 \\ P_2 \\ \cdot \\ \cdot \\ P_n \end{bmatrix}$$

5-17

$$\begin{bmatrix} P_{21} \\ P_{22} \\ \cdot \\ \cdot \\ P_{2n} \end{bmatrix} = [B]^{-1} [A] \begin{bmatrix} P_{31} \\ P_{32} \\ \cdot \\ \cdot \\ P_{3n} \end{bmatrix} + [L] \begin{bmatrix} P_{11} \\ P_{12} \\ \cdot \\ \cdot \\ P_{1n} \end{bmatrix}$$

5-18

$$\begin{bmatrix} P_{c-1,1} \\ P_{c-1,2} \\ \cdot \\ \cdot \\ P_{c-1,n} \end{bmatrix} = [B]^{-1} [A] \begin{bmatrix} P_{c1} \\ P_{c2} \\ \cdot \\ \cdot \\ P_{c,n} \end{bmatrix} + [L] \begin{bmatrix} P_{c-2,1} \\ P_{c-2,2} \\ \cdot \\ \cdot \\ P_{c-2,n} \end{bmatrix}$$

5-19

$$\begin{bmatrix} P_{c1} \\ P_{c2} \\ \vdots \\ P_{cn} \end{bmatrix} = \lambda \begin{bmatrix} \mu_1 & 0 & \dots & \dots & 0 \\ 0 & \mu_2 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & \mu_n \end{bmatrix}^{-1} \begin{bmatrix} P_{c-1,1} \\ P_{c-1,2} \\ \vdots \\ P_{c-1,n} \end{bmatrix} \quad 5-20$$

where the matrices  $[A]$ ,  $[B]$ , and  $[L]$  are defined as above. Now, the set of equations 5-17 through 5-20 can be solved recursively. In Appendix A we will show the solution for the case  $n=3$  and  $c=3$ . The dimensions of the above matrices and vectors depend only upon  $n$ , whereas  $c$ , the system capacity, has no effect. The role that  $c$  plays is the quantity of the  $P$  vector, i.e., we will have  $c$   $P$ -vectors each of dimension  $n \times 1$ . Note that even after the solution for  $[P_{11} \ P_{12} \ \dots \ P_{1n}]^T$  is found, equation 5-0 can not be used to compute  $P_0$ , since one equation of the set is always redundant. Instead, we could use the normalizing relation:

$$\sum_{i=0}^c \sum_{j=0}^n P_{ij} = 1 \quad 5-21$$

By inspection we see that it is not possible to have the following states:

since  $P_{i0} = 0$  for  $i=1, \dots, c$  (i.e.,  $i$  exact's are in the system and none of the SPE's are busy.)

and  $P_{0j} = 0$  for  $j=1, \dots, n$  (i.e., the probability of the system being empty with a processor busy.)

and finally  $P_{00} = P_0$

Then equation 5-21 reduces to

$$P_0 + \sum_{i=1}^c \sum_{j=1}^n P_{ij} = 1 \quad 5-22$$

where  $\sum_{i,j} P_{ij}$  are in fact the column vectors we solved for

earlier, and are in terms of  $P_0$

$$P_0 = 1 / ( 1 + \sum_{i=1}^c \sum_{j=1}^n P'_{ij} ) \quad 5-23$$

where  $P_{ij} = P_0 P'_{ij}$ . The utilization is found as

$$\rho_{\text{sys}} = ( \sum_{i=1}^c \sum_{j=1}^n P'_{ij} ) \times 100 \quad 5-24$$

where  $P_{ij} = P_0 P'_{ij}$ . The utilization of each SPE is also found:

$$\begin{aligned} \rho_{k, \text{PE}} &= ( P_{1k} + P_{2k} + P_{3k} + \dots + P_{ck} ) \times 100 \\ &= ( \sum_{i=1}^c P_{i,k} ) \times 100 \end{aligned} \quad 5-25$$

for  $k=1, 2, \dots, n$

In general, the expected number in the system is defined as

$$N_{\text{sys}} = L = \sum_{n=1}^n n P_n$$

where  $P_n$  is the probability of the system being in state  $n$ ; and  $N_{\text{sys}}$  signifies both the transactions waiting and in service. In our case the expected number in the system is not as obvious as in the above for the probability also depends on the type of service. Consequently, the expected number in the system is defined as

$$N_{\text{sys}} = 1x[P_{11} + P_{12} + \dots + P_{1n}] + 2x[P_{21} + P_{22} + \dots + P_{2n}] + \dots + cx[P_{c1} + P_{c2} + \dots + P_{cn}] \quad 5-26$$

In a more compact form  $N_{\text{sys}}$  is given as

$$N_{\text{sys}} = \sum_{n=1}^c \left\{ n \cdot \sum_{i=1}^n P_{ni} \right\} \quad 5-27$$

where  $n$  is the number of PE's and  $c$  is the system capacity.

The throughput ( $T$ ) is found,

$$T = \rho_{\text{sys}} \cdot (\Sigma UV / n) \quad \text{Instruction/unit time} \quad 5-28$$

and the throughput for each SPE is also found by,

$$T_{\text{SPE}} = (\rho_k^{\text{PE}}) \cdot (UV [k]) \quad 5-29$$

$$k = 1, 2, \dots, n$$

The mean queue length is given as

$$N_q = E[L_q] = 0[P_{11} + P_{12} + \dots + P_{1n}] + 1[P_{21} + P_{22} + \dots + P_{2n}] + \dots + (c-1)[P_{c1} + P_{c2} + \dots + P_{cn}]$$

$$N_q = \sum_{i=2}^c (i-1) \sum_{j=1}^n P_{ij} \quad 5-30$$

The average instruction response time,  $T_w$  is equal to  $N_{\text{sys}}/\lambda_a$  a

$$\text{where } N_{\text{sys}} = N_q + N_s$$

and  $\lambda_a$  = the actual arrival rate

$$= \lambda \left[ 1 - \sum_{j=1}^n P_{cj} \right] \quad 5-31$$

where  $\sum_{j=1}^n P_{cj}$  is the probability that the system is not full.

The average number in service,  $N_s$ , is equal to  $n \cdot E[\rho_{\text{sys}}]$  and

$$E[\rho_{\text{SPE}}] = \rho_k^{\text{PE}} = \left( \sum_{k=1}^n \sum_{i=1}^c P_{ik} \right) / n \quad 5-32$$

$$N_s = n \cdot \left[ \sum_{k=1}^n \sum_{i=1}^c P_{ik} \right] / n = \sum_{k=1}^n \sum_{i=1}^c P_{ik} \quad 5-33$$

From Eq 5-30 and Eq 5-33,  $N$  is found,

$$N_{\text{sys}} = \left( \sum_{i=2}^c (i-1) \sum_{j=1}^n P_{ij} \right) + \sum_{k=1}^n \sum_{i=1}^c P_{ik}$$

$$\begin{aligned} \text{let } k=j \\ N_{\text{sys}} &= \sum_{i=2}^c (i-1) \sum_{j=1}^n P_{ij} + \sum_{j=1}^n \sum_{i=1}^c P_{ij} \\ &= \sum_{\ell=1}^n P_{1\ell} + \sum_{i=2}^c \sum_{j=1}^n ((i-1) P_{ij} + P_{ij}) \\ &= \sum_{\ell=1}^n P_{1\ell} + \sum_{i=2}^c \sum_{j=1}^n i \cdot P_{ij} \end{aligned}$$

$$N_{\text{sys}} = \sum_{i=1}^c \sum_{j=1}^n i \cdot P_{ij} \quad 5-34$$

Equation 5-34 confirms our result obtained in Eq 5-27.

Consequently, the average response time is found,

$$T_w = \left( \sum_{i=1}^c \sum_{j=1}^n i \cdot P_{ij} \right) / \lambda \left[ 1 - \sum_{j=1}^n P_{cj} \right] \quad 5-35$$

and the average time in the queue,  $T_q$  is computed as,

$$T_q = N_q / \lambda_a = \left[ \sum_{i=2}^c (i-1) \sum_{j=1}^n P_{ij} \right] / \lambda \left[ 1 - \sum_{j=1}^n P_{cj} \right] \quad 5-36$$

### 5.3.1.3 Solution of the analytic technique

The previously derived state equations require a few matrix multiplications and inversions. For example, equations 5-17 through 5-20 must be solved recursively, starting with the last equation 5-20 and working up toward 5-17. When the vector  $[P_{1i}]$  is found it will provide the results in terms of  $P_0$ . Again, working down toward equation

5-20, the vectors  $[P_{2i}]$  through  $[P_{ci}]$  are found, and all are in terms of  $P_o$ . In Appendix A we illustrate the above discussion with the solution of a numerical example. APL is used to solve the system for higher orders. This language provides great flexibility when working with matrices. The interactive nature of this language has encouraged us even more. The algorithm used follows exactly the procedure above. The program listing is given in the Appendix B.

### 5.3.2 The Simulation Model

The simulation model analysis is a good supplement to the analytic model. The statistics collected in a simulation run should somewhat agree (or follow the same direction) with those obtained in the analytic case. Simulation models in general are used to observe more specific system behavior to variation of certain system parameters. The language GPSS is used for the model simulation. With simulation, however, we can relax some assumptions that are made in the analytic case. For instance, the composite service time of PE and of the control unit (used in the analytic case) can be separated without much complexity. The later attribute, will provide us the capability of measuring the CU statistics independently of the PE's. As it was pointed out earlier, the price for this great flexibility is paid for by the computation time necessary to run the simulation program.



Two different model combinations will be studied with the simulation. First, we will run the same program that corresponds to the analytic case. Then, the CU and PE service times are treated separately, for the SPE's, and the necessary modification for the SPE's are made. That is, each SPE will have its own micro-register in order to store the current microinstruction. In the second run two sets of statistics are generated, one for the CU and another for the SPE's.

The corresponding analytic model for the second case is very complex and finding the analytic equations is extremely tedious. We will not derive the mathematical equations, but instead will set up the model. However, by applying the same technique implemented earlier, the state transition diagram is produced, and is shown in Figure 22 for  $n=3$  and system capacity of  $c$ . As can be observed the level of complexity increases even more for a slightly higher  $n$ . In order to provide a feeling for this complexity, the sub-state transition diagram is presented in Figure 23 for the case  $n=5$ ,

where  $n < k < c$  and number of states in Figure 23  $= \sum_{i=0}^N \binom{N}{i}$   
 this is true for every  $n < k < c$ .

The total number of states  $= 1 + (1+N) + (1+2N) + \dots + (1+(N-1)N)$

$$+ \sum_{k=N}^c \sum_{i=0}^N \binom{N}{i}$$

$$c > n \quad \text{and} \quad \binom{N}{i} = \frac{N!}{i! (N-i)!}$$

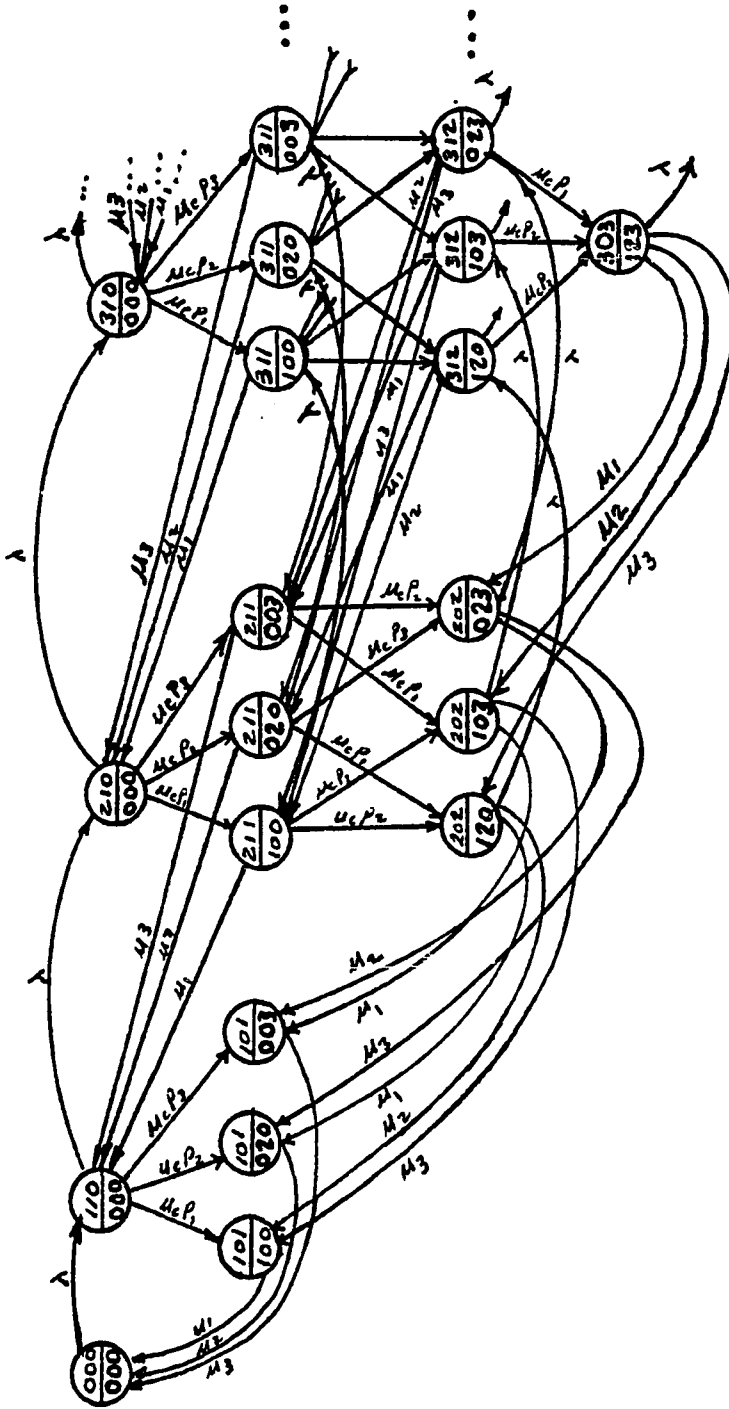


Figure 22: The State Transition Diagram for the Second Case

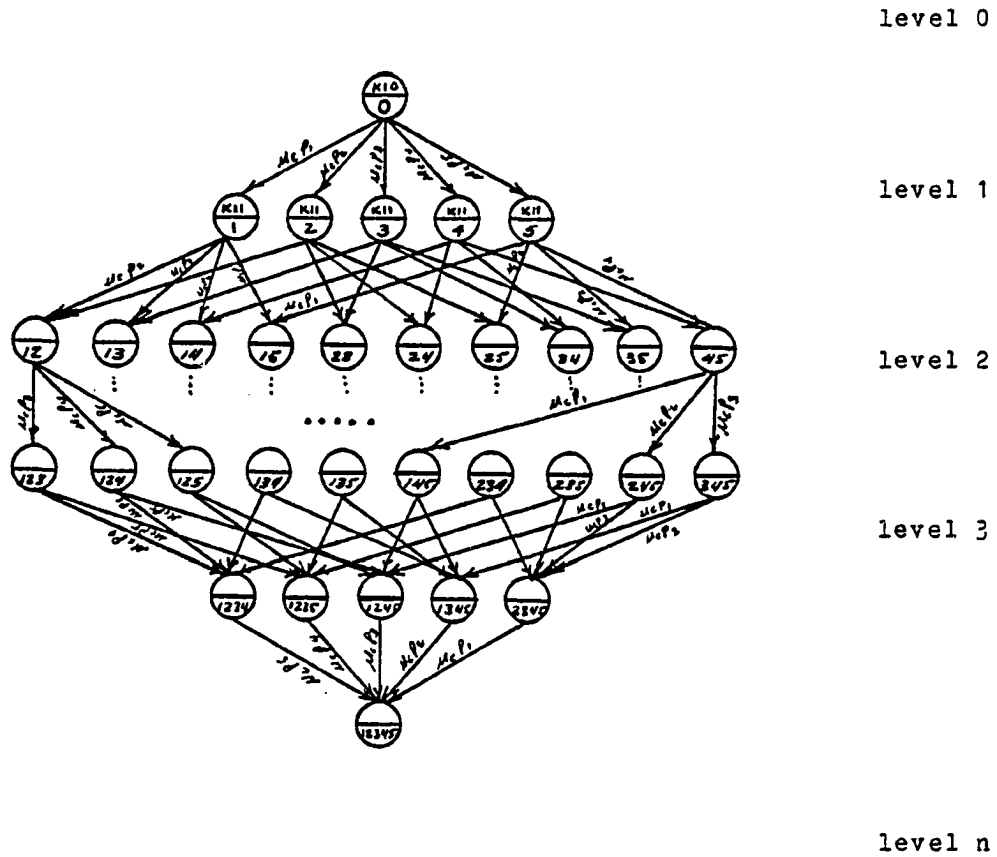


Figure 23: Sub-State Transition Diagram for n=5

The set of notations in Figure 22 is read as follows:

abc,ijk

- a: the number of instructions in the system
- b: 0 free } CU  
1 busy }
- c: number in service, (i.e. number of activ SPE ), c is <n
- i: 0 free } SPE1  
1 busy }
- j: 0 free } SPE2  
2 busy }
- k: 0 free } SPE3  
3 busy }

The assumptions made for this model are as follows:

1. The CU cannot remain idle while an instruction is in the queue, i.e., the  $n, 0, m$  (for  $m < n$ ) states are not defined.
2. Only one instruction at a time is serviced by the CU. When all SPE's are busy, and uc service completion occurs, the instruction will remain in that state until a  $\mu_i$  ( $i=1, \dots, n$ ) occurs.

In simulation, the interarrival and interdeparture times, instead of the arrival and service rate, are used. That is, in the analytic case  $\lambda$  and  $\mu$  are used for the average arrival and departure rate, where in simulation  $1/\lambda$  and  $1/\mu$  are used. These later modifications are due to the language requirements. The simulation flow chart for the controlled multiserver model is presented in Figure 24. The simulation program is given in Appendix C.

#### 5.4 ANALYSIS OF RESULTS

Two different analyses are performed for the controlled multiserver model. The first analysis includes both analytic and simulation models, whereas the second analysis covers only the simulation model. In the first analysis the assumptions made for the analytic case will also hold for the simulation case, essentially, the CU and one SPE can be

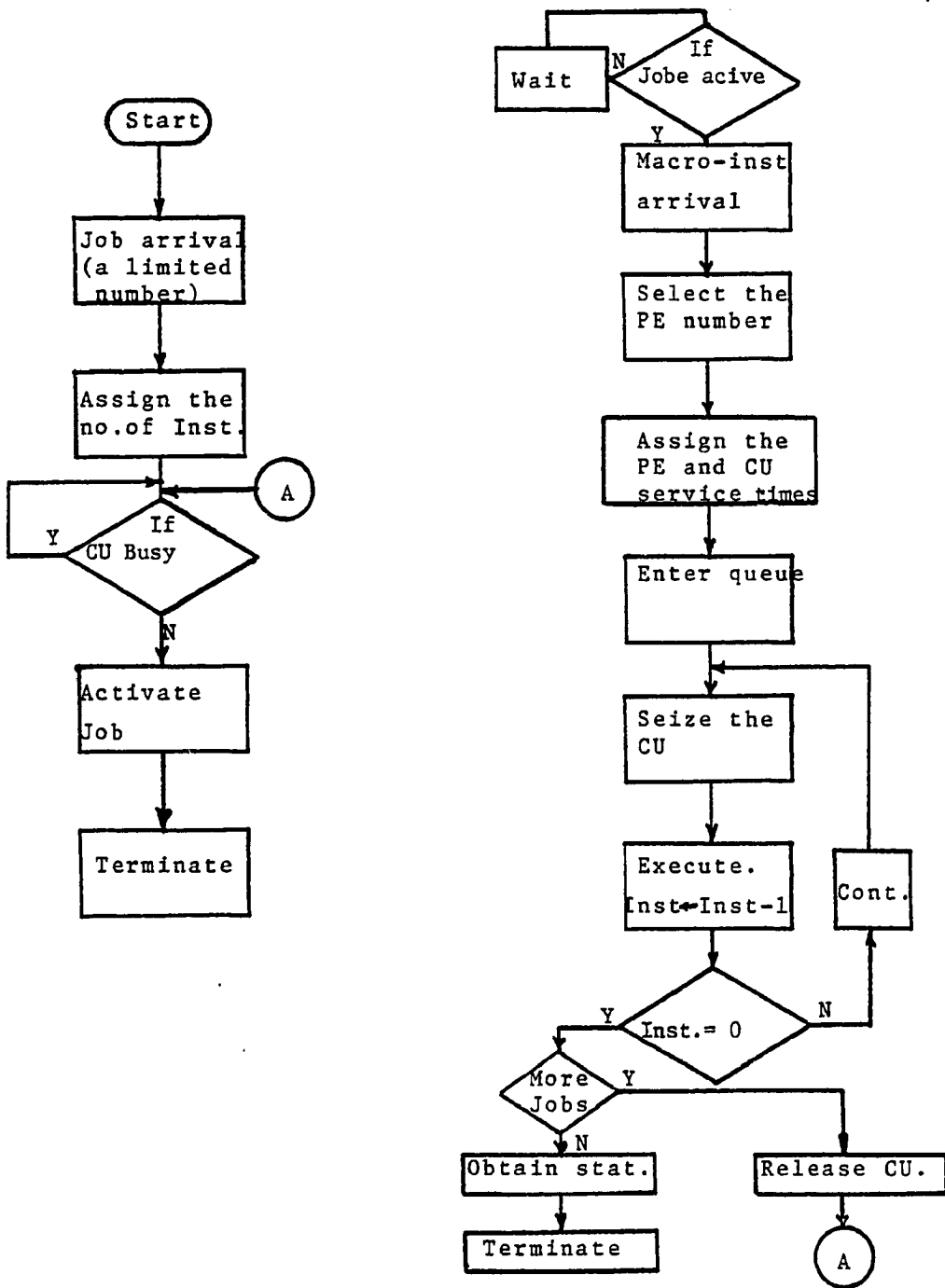


Figure 24: The Simulation Flowchart for the Controlled Multiserver Model

busy at any given time. The active SPE is selected according to the probability selection vector PV, as indicated earlier. The analytic model is valid for  $n > 3$ , where  $n$  is equal to the number of SPE's in the system.

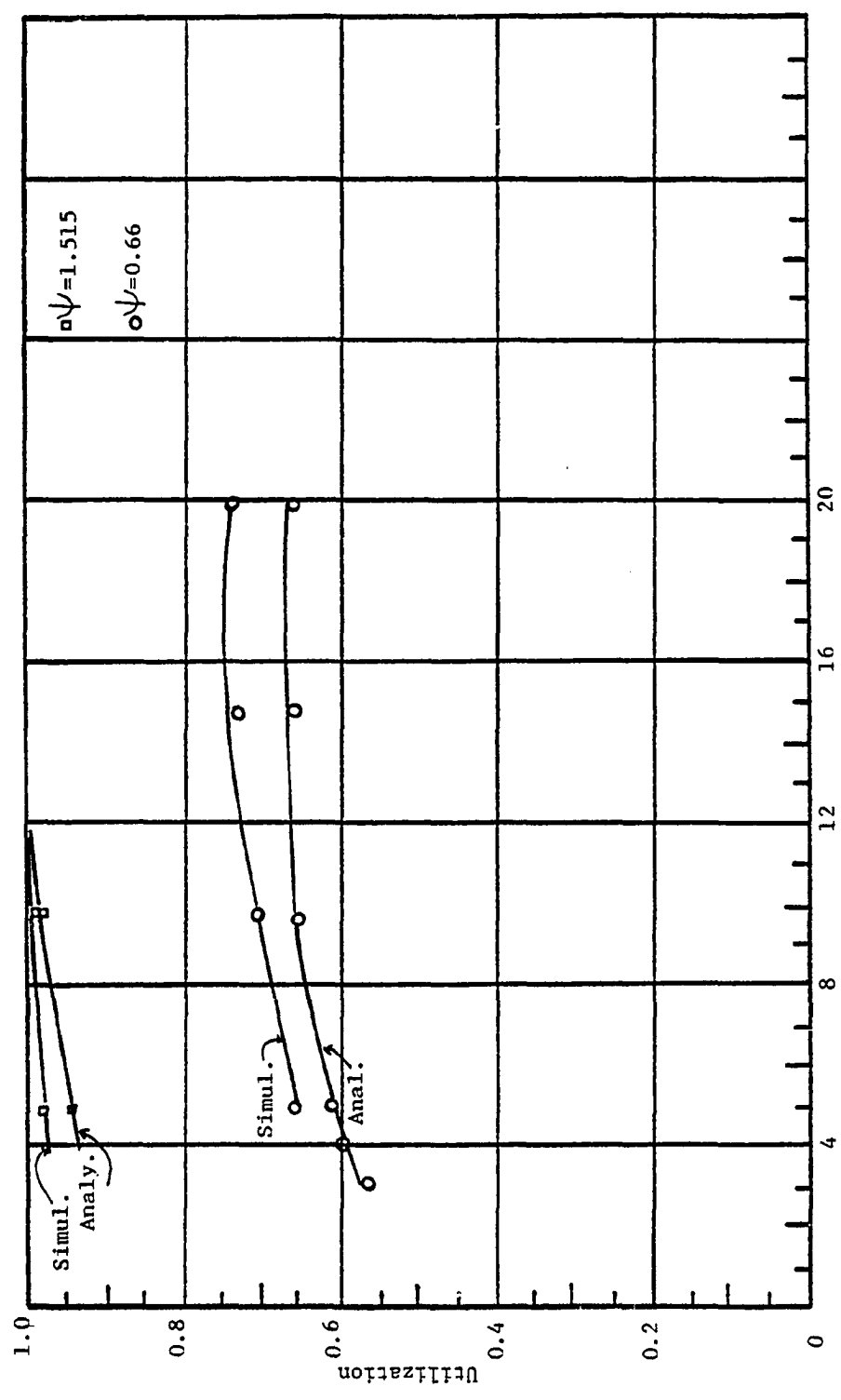
Figure 25 shows the average utilization (for the system and for the PE's) versus system capacity for different combinations of  $\psi$  (traffic intensity). Note, however, that the number of SPE's in this analysis will not be critical for only the CU and one SPE are active at any given instant. The average service time is found by  $1/(u_c + u_{SPE})$ . For higher  $\psi$  the utilization will also be higher; this is evident since the system remains idle fewer times for higher  $\psi$ . Figure 26 illustrates the throughput as a function of the system capacity. The utilization obtained via the simulation method is 11% higher than the one obtained by the analytic method. As  $\psi$  approaches unity, the gap between the analytic and simulation result decreases, as shown in Figure 25. Tables 1 and 2 show the utilization of individual SPE's for the case

$$PV = [0 \ 0 \ 0.04 \ 0.05 \ 0.06 \ 0.1 \ 0.15 \ 0.15 \ 0.2 \ 0.25]$$

Different selection probabilities can be implemented and studied if desired. Figure 27 represents the utilization of the SPE and the system utilization for the equal probability selection vector, i.e.,

$$PV = [.1 \ .1 \ .1 \ .1 \ .1 \ .1 \ .1 \ .1 \ .1 \ .1]$$

For higher number of SPE's, the utilization, hence the throughput of each SPE will decrease, as expected.



c: System Capacity

Figure 25: The Utilization Versus the System Capacity

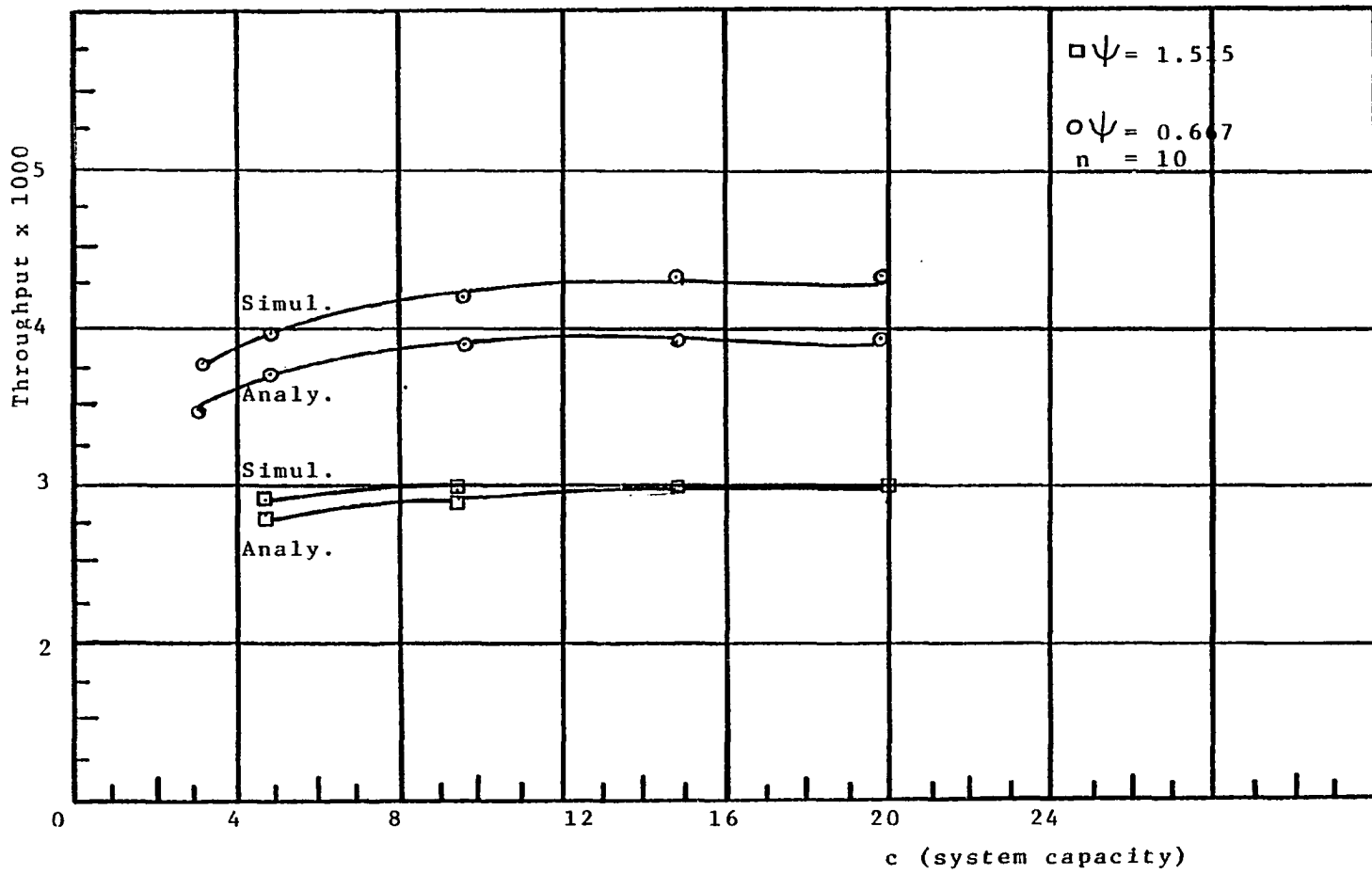


Figure 26: The Throughput Versus the System Capacity



TABLE 1

The Utilization of Individual SPE's ( $\psi=0.616$ )

PV = [ 0 0 0.04 0.05 0.06 0.10 0.15 0.15 0.2 0.25 ]

	<u>Simulation</u>				<u>Analytic</u>			
	c=5	c=10	c=15	c=20	c=5	c=10	c=15	c=20
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	5.7	4.6	1.2	2.9	3.82	3.98	3.99	4.0
4	3.5	4.3	2.9	1.7	4.7	4.97	4.99	5.0
5	3.1	5.0	9.7	4.1	5.72	5.96	5.99	5.99
6	10.1	7.3	7.0	8.1	9.54	9.946	9.99	9.99
7	15.1	16.1	16.8	13.3	14.3	14.9	14.99	15.0
8	13.4	20.8	12.9	17.7	14.3	14.9	14.99	15.0
9	22.7	12.3	17.9	23.3	19.07	19.98	19.99	20.0
10	26.1	28.7	30.5	28.0	24.0	24.87	24.98	24.999
$\Sigma$	99.7	99.1	98.9	99.1	95.4	99.46	99.93	99.99

The second case is analyzed by simulation only. As is shown in section 6.3.2, for such a case the analytic technique becomes very complex. In this analysis, we provide a buffer unit for each SPE, so that it will store its current microinstruction, thus freeing the control unit for other SPE. The current microinstruction will be loaded to the next SPE provided it is idle. The system can even become more efficient (as well as more complex) by providing each SPE with its own queue. This last modification is beyond the scope of this dissertation, and will not be

TABLE 2

The Utilization of Individual SPE's ( $\psi=1.5152$ )

$$PV = [0 \ 0 \ 0.04 \ 0.05 \ 0.06 \ 0.10 \ 0.15 \ 0.15 \ 0.2 \ 0.25]$$

	<u>Simulation</u>				<u>Analytic</u>			
	c=5	c=10	c=15	c=20	c=5	c=10	c=15	c=20
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	3.1	1.4	0.9	0.9	2.54	2.65	2.70	2.66
4	3.1	2.0	1.4	1.4	3.17	3.314	3.33	3.33
5	2.0	3.7	2.7	2.7	3.81	3.977	3.997	4.0
6	9.8	8.5	10.4	10.4	6.35	6.63	6.662	6.78
7	11.2	8.5	8.6	8.6	9.52	9.94	9.992	9.99
8	10.3	10.1	9.7	9.7	9.52	9.94	9.992	9.99
9	12.2	18.7	20.2	20.2	12.69	13.26	13.32	13.43
10	16.0	18.4	20.0	20.0	15.86	16.57	16.65	16.66
$\Sigma$	67.7	71.3	73.9	73.9	63.46	66.28	66.62	66.66

TABLE 3

The Total Throughput of the SPE's

$$PV = [0 \ 0 \ 0.04 \ 0.05 \ 0.06 \ 0.1 \ 0.15 \ 0.15 \ 0.2 \ 0.25]$$

	<u>Simulation</u>				<u>Analytic</u>			
	c=5	c=10	c=15	c=20	c=5	c=10	c=15	c=20
$\Sigma$	0.00299	0.00297	0.00297	0.00297	0.00286	0.00298	0.00300	0.00300
$\psi = 1.5152$								
$\Sigma$	0.00406	0.00430	0.00440	0.00440	0.00380	0.00398	0.00399	0.00400
$\psi = 0.616$								

discussed here.

An improvement is achieved in the utilization of each SPE, as illustrated by Figure 28. As the number of SPE's

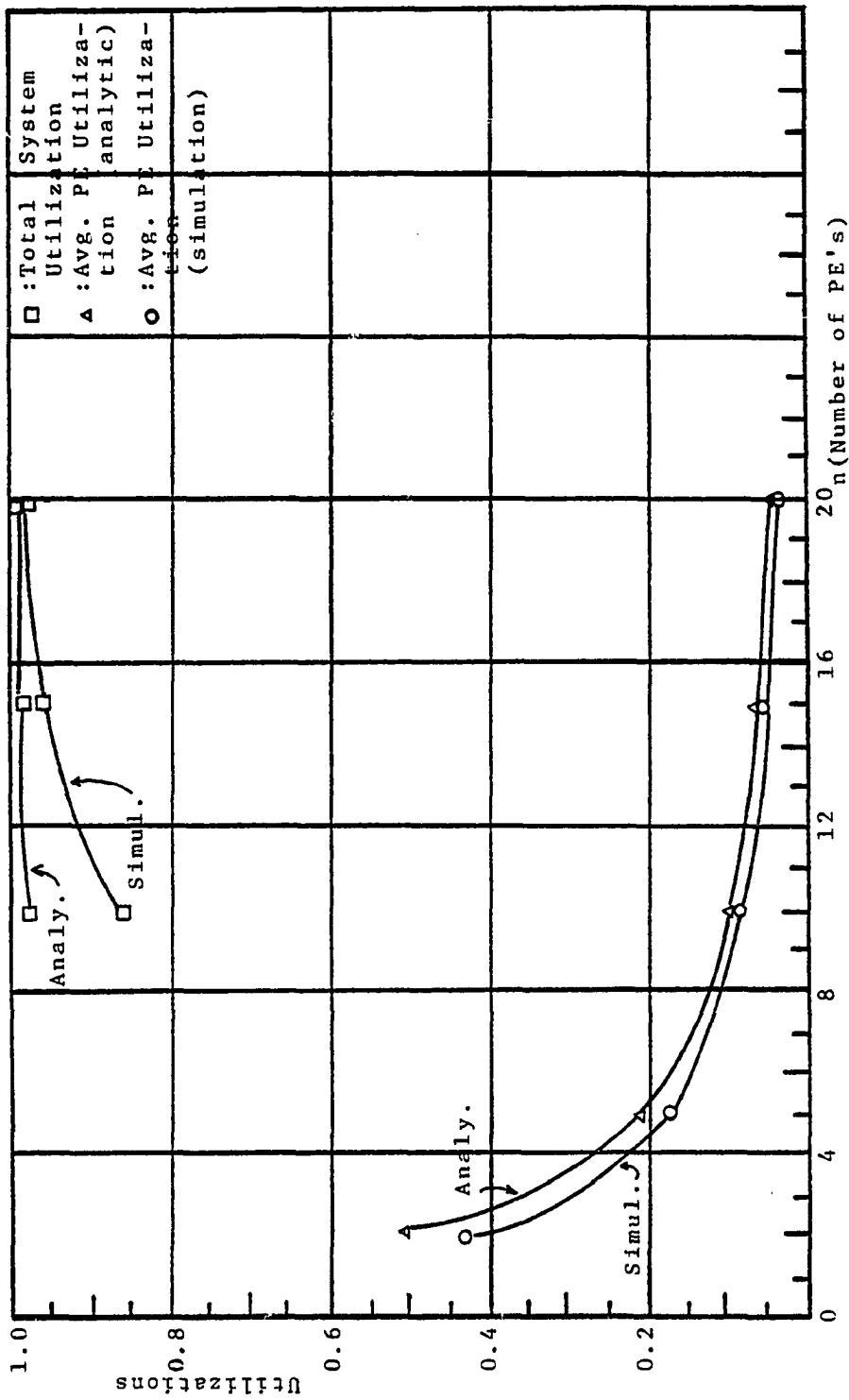


Figure 27: The Utilization as a Function of n

(n) increases, the SPE utilization remains somewhat unchanged, or varies very slightly. Comparing the graphs of Figure 28 with those of Figure 27, the improvement is evident. For example, when  $n=10$ , the improvement is between 300% to 400% higher than that of the first case. The total system throughput is also given by the graphs of Figure 29. The average queue length remains practically uniform for each system capacity. Figure 30 illustrates the average queue length for  $c=10$  and 30, as a function of  $n$ .

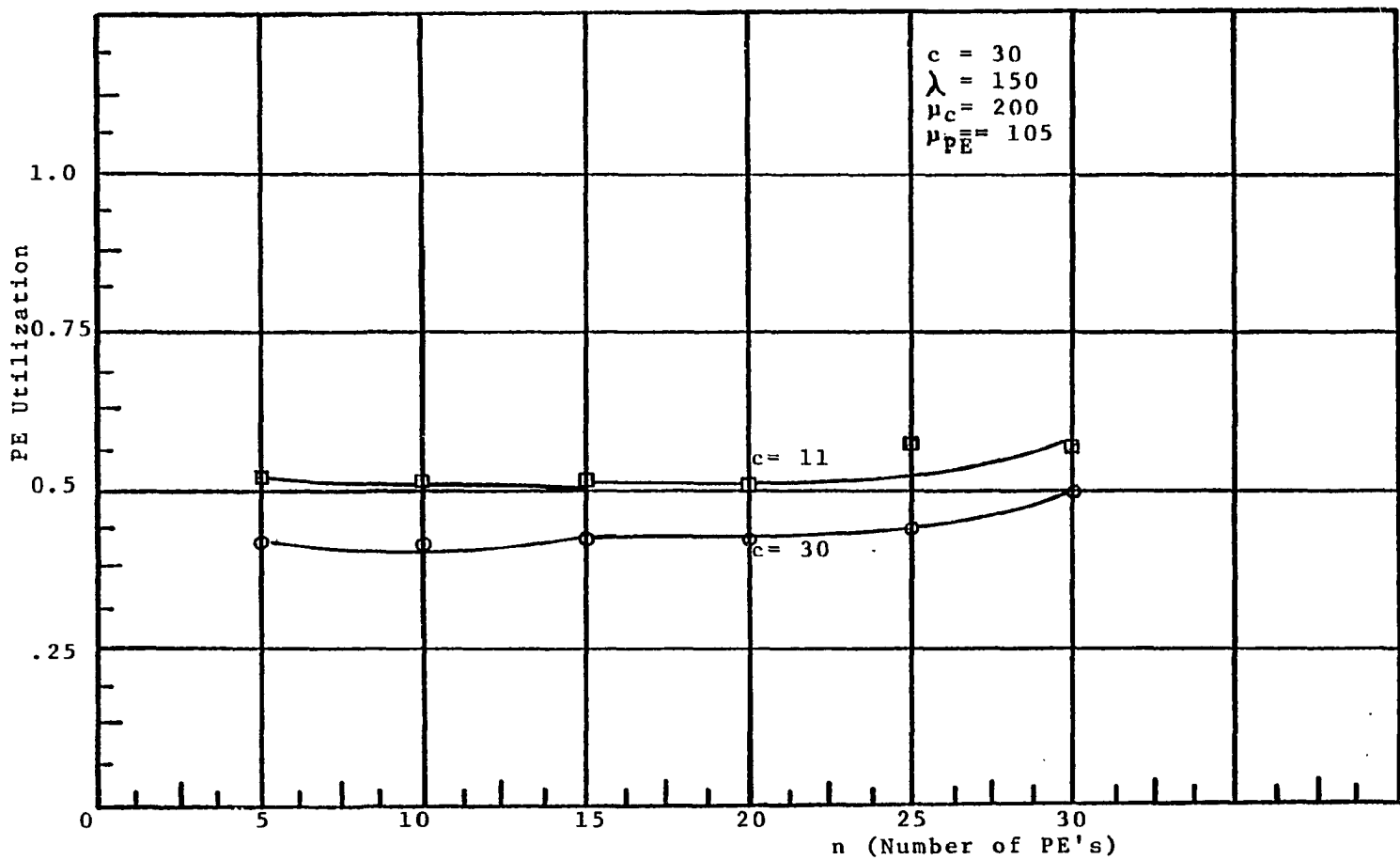


Figure 28: The SPE Utilization as a Function of n

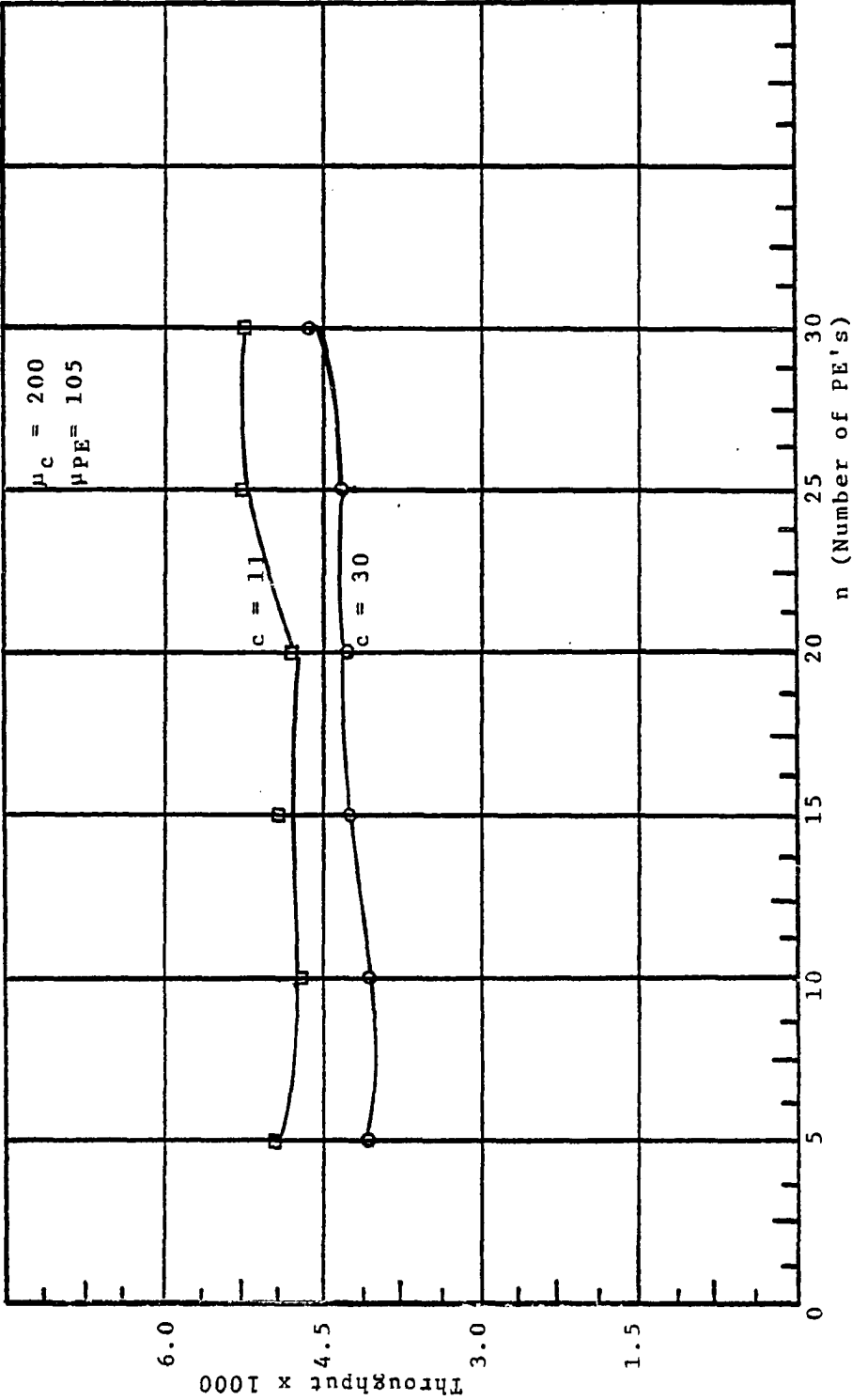


Figure 29: System Throughput as a Function of n

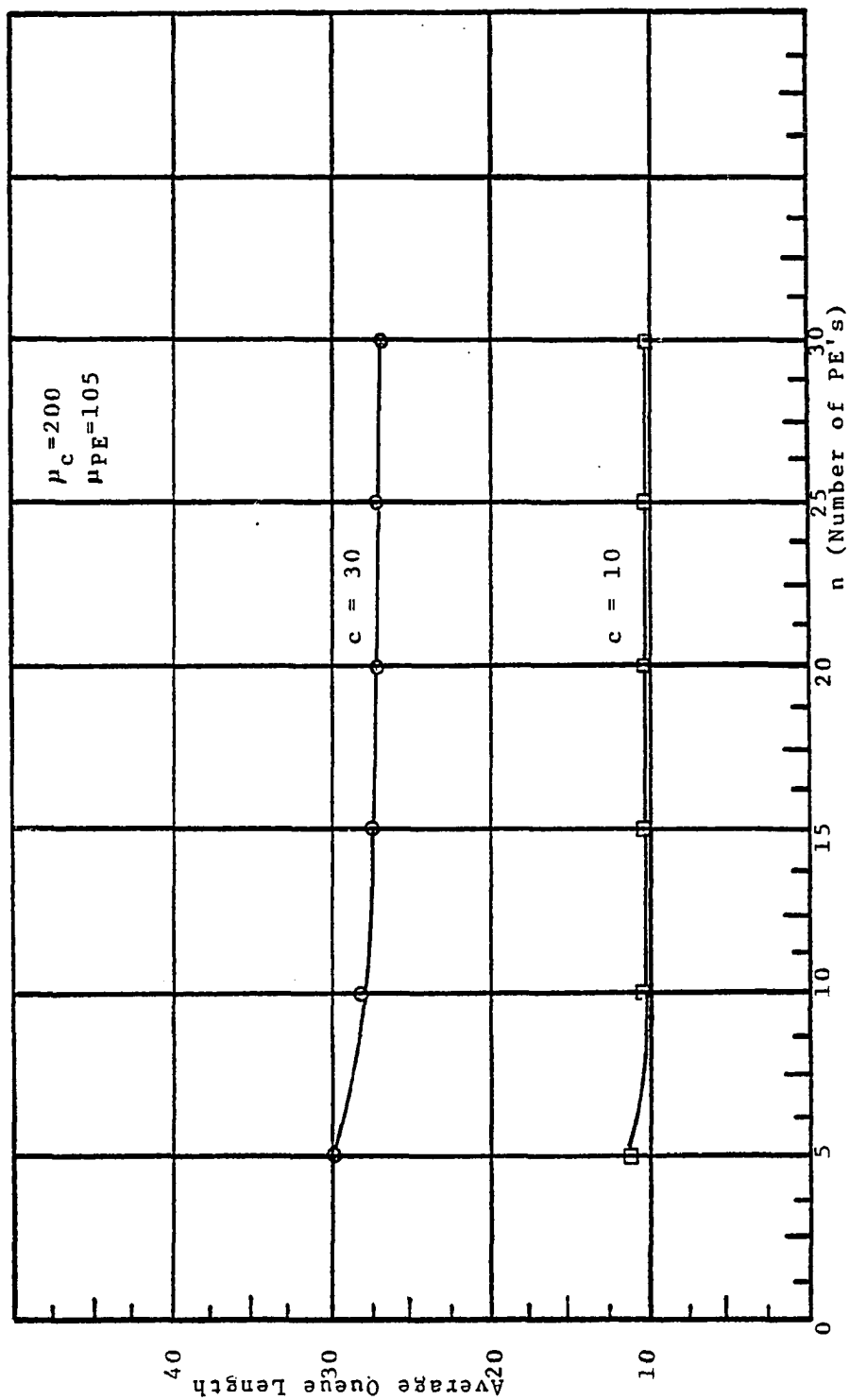


Figure 30: The Average Queue Length as a Function of n

## Chapter VI

### A PROGRAMMABLE ARRAY MODEL

#### 6.1 INTRODUCTION

In array processing systems, a number of identical processors are used. The manner in which these processing elements. ( PE's ) are connected differ from one system to another. Some important array models are presented in chapter II. As an example of an array system, we propose the array model discussed within this chapter. The need for array processing architecture arises in environments where speed and throughput are of great importance to the extent that cost will not be very critical. Array processors are highly specialized systems. An example of an array system that uses bit-slice elements is the Purdue multiprocessor (PM 4) system, [BRIG79 ],and [ BRIG82].



## 6.2 SYSTEM ORGANIZATION

The system can be configured to consist of two principle blocks:

1. The control block section.
2. The processing block section.

The control block consists of a complete bit-slice machine, i.e., a control unit and a processing element. A number of identical processing elements constitute the processing block. Figure 31 illustrates the organization of these two blocks.

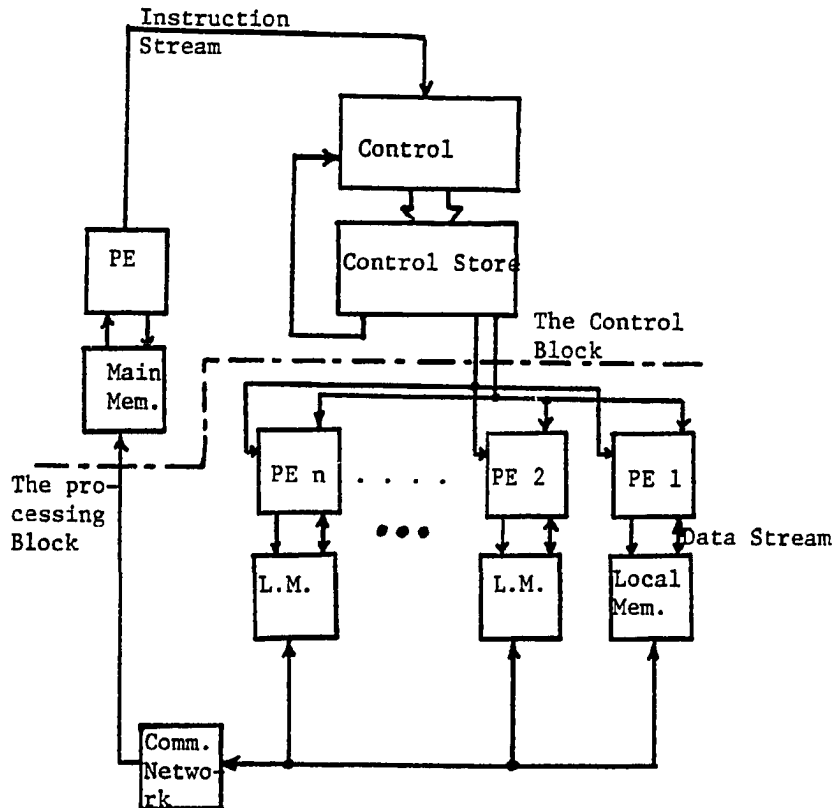


Figure 31: A Block Diagram of the System

The control section contains the control store which serves as the main supplier of microinstructions to the processing elements PE-1 through PE-n, and to the PE of the control section. The intercommunication between the two sections is controlled by the microcode. We will explain this organization in more detail in the latter sections.

### 6.3 HARDWARE ARCHITECTURE

As it was pointed out in the last section, the NMSU-MBSE system consists of two sections, namely, the control unit and the processing elements. We now go through some more detail to explain how the hardware modules are configured and how the interconnections between the different modules are performed.

#### 6.3.1 Processing Elements (PE's)

The modified ALU slice developed in chapter III (Figure 13), will be used in this array organization. Since the control unit will be used to serve a number of PE's, and furthermore, since each PE has its own data stream to work on, then each PE will require its own status, shift, and carry control unit. Thus it is necessary to use a separate AM2904 for each PE.

The modular concept could also be applied here, since more than one board for each PE can be used (e.g., a 16-bit

PE). Thus, some means of communication and shift justification must be made. The multiplexers 1,2, and 3 in Figure 13 are implemented for this purpose. As a result, the overall system is left unchanged with regard to the modularity. A variable length PE can easily be implemented. This latter feature provides an adaptable array system that can be used in applications where the word length is quite long, such as in image analysis and pattern recognition application. A simplified PE block diagram will be used from here on, and is given in Figure 32. Whenever the diagram of Figure 32 is used, it should automatically imply Figure 13.

### 6.3.2 The Control Section

The control unit controls the activities of the whole system. It has the necessary microcode to emulate a specific machine. By the same token, different microcodes could be implemented to emulate different microprocessor systems or even new ones.

The microprogram store width will be slightly different from that of chapter III. Each microinstruction, in addition, should have a field specified for the intercommunication purposes. Furthermore, some microbits should be assigned for the selection of the various PE's. For an  $n$  PE system, we require  $n+1$  bits for this selection. Since two different sets of microinstructions are

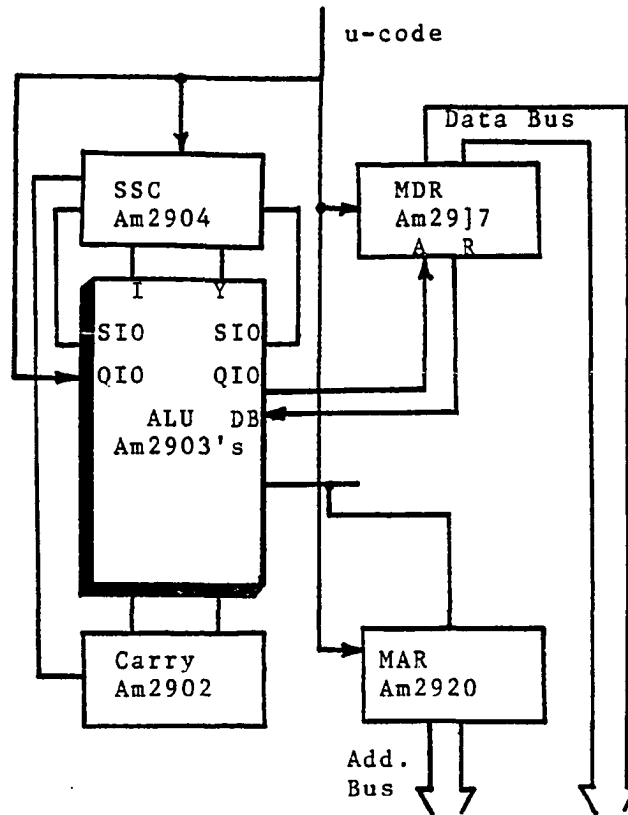


Figure 32: The Simplified PE Block Diagram

implemented, i.e., one for the CU and one for the PE's, some means of distinguishing between them is required. The instructions for the CU are of the control type (that is, jump, subroutine call, ...), whereas the instructions used in the PE's section are for data manipulations (i.e., add, sub, multiply, ...). The micro instructions are shown to consist of three principle fields:

THE INTERCOMMUNICA- TION FEILD	THE SEQUENCER FEILD	THE PROCESSING ELEMENT FEILD
-----------------------------------	------------------------	---------------------------------

Figure 33 illustrates how these fields are arranged.

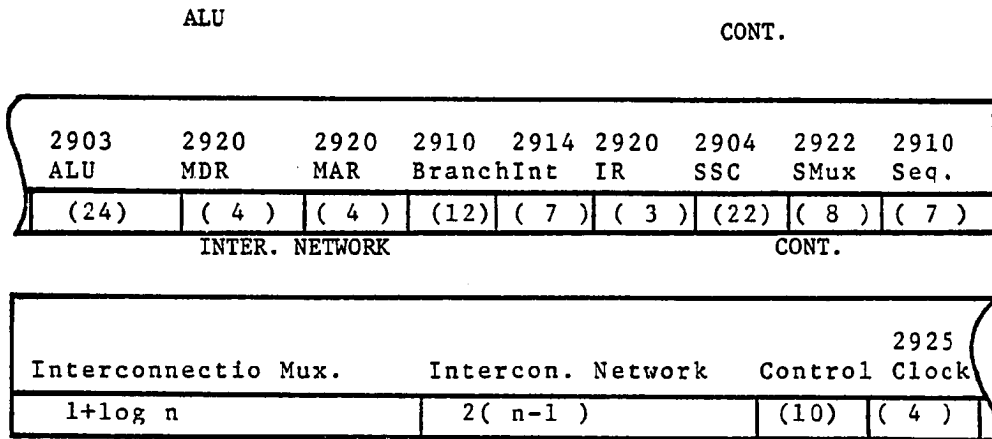


Figure 33: The Microinstruction Fields

A detailed system organization is presented in Figure 34. There are two means of communications in the system: the intra-module communication, between the processing elements, and the intercommunication of the controller with the processing elements. An interconnection network is used for the intra-module communication. Figure 35 shows this interconnection network for the case where the number of PE's and the number of memory units are equal to four. This network is essential for the vector operation case discussed in section 6.5. Each PE can have access to the memory of the neighboring PE. The interconnection network is controlled by the microcode. Each subunit in the network takes two control bits, one for each direction. The total number of microcode bits designated for the interconnection

network is equal to  $(n-1) \times 2$ , where  $n$  is the number of PE's (also in this case the number of memory units equal to  $n$ , i.e., each PE has its own local memory).

A multiplexer is used to interconnect the PE's to the controller. Some microcode bits are designated to control this mux. For an  $n$  PE system the following will be true:

1. Number of microcode bits for the Mux select lines equals

$$\lceil \log_2 n \rceil + 1$$

where:  $\lceil \log_2 n \rceil$  bits are used for the select and the 1 bit is used for the control line.

2. For an  $n$  PE and an  $M$  bits data bus ( $M$ -bit machine), then the number of multiplexers =  $M$  each of  $n$  to 1 type.

Figure 36 illustrates a system of 4 PE's and 8-bits data bus.

As a final note, we compare the two interconnection networks of Figure 37. In each case the number of PE's and memory units are equal to 4. In part a, each PE can have a direct access to any of the memory units. This means of communication is a costly one. The number of gates required ( $G$ ) is found as follows:

$$G_a = 2n w \quad i$$

where  $w$  is the width of the data bus. The  $G$  for part b (the one implemented above) is found as

$$G_b = 2(n-1)w \quad ii$$

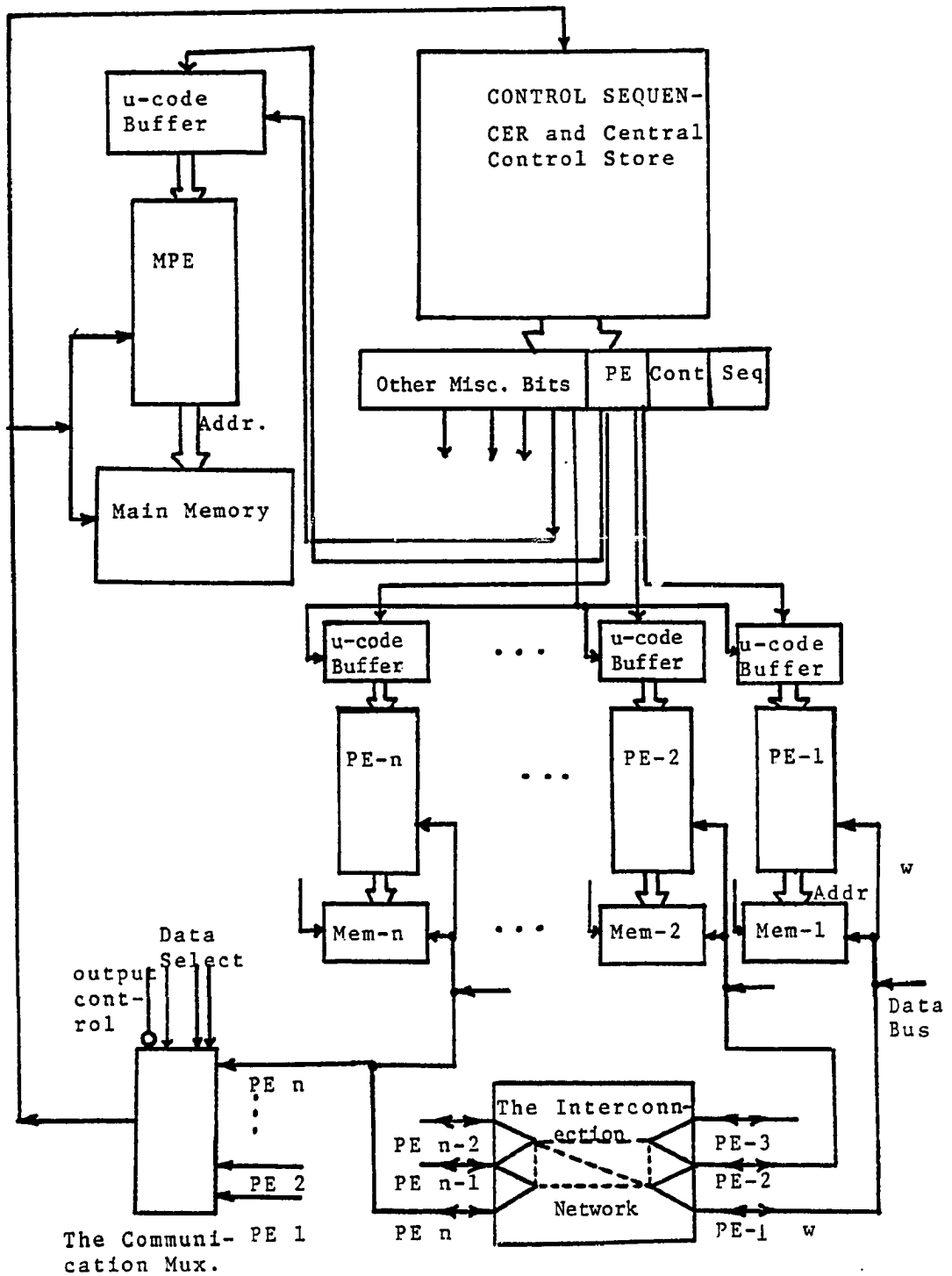


Figure 34: The Complete System Organization

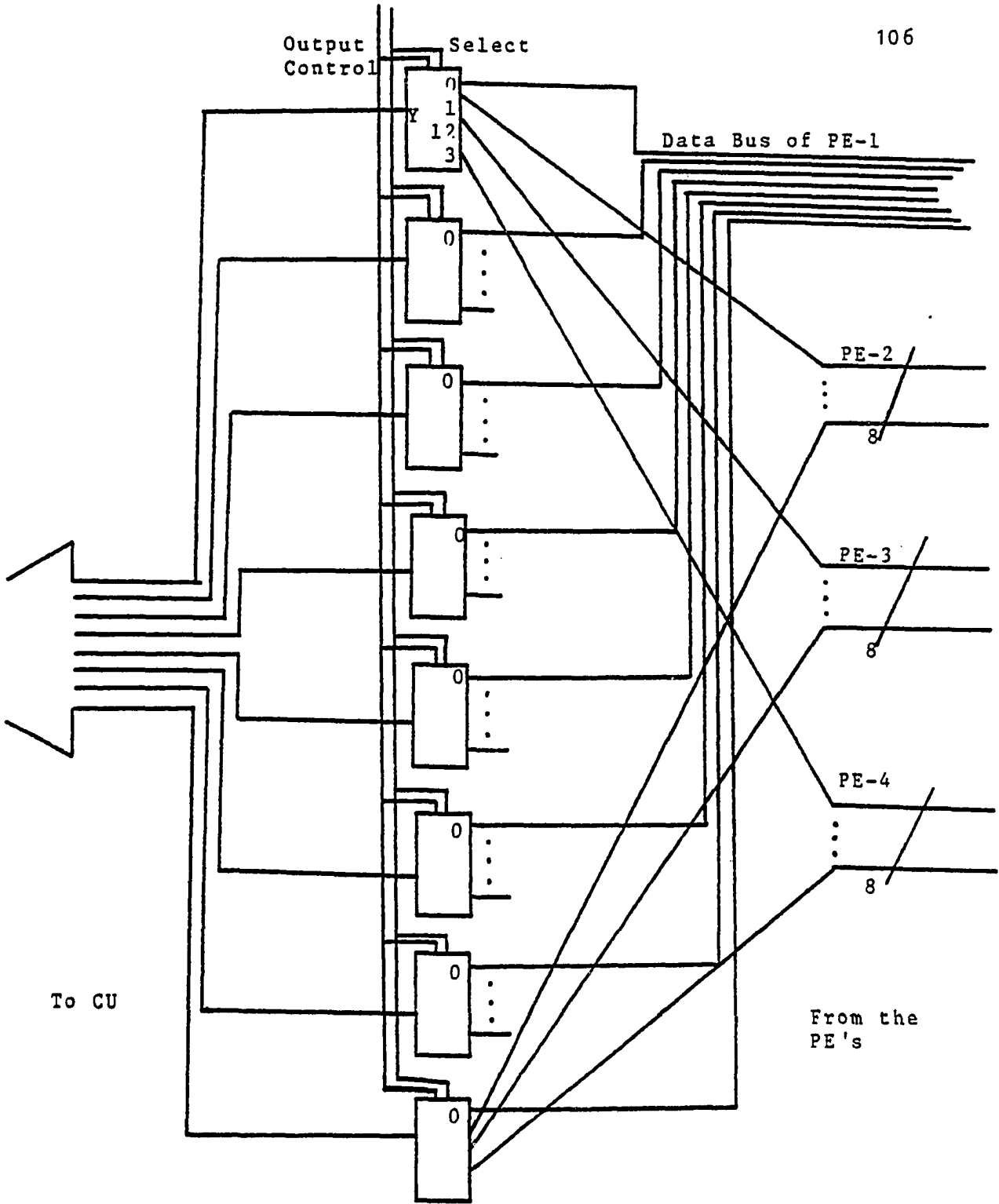


Figure 35: The Interconnection Network



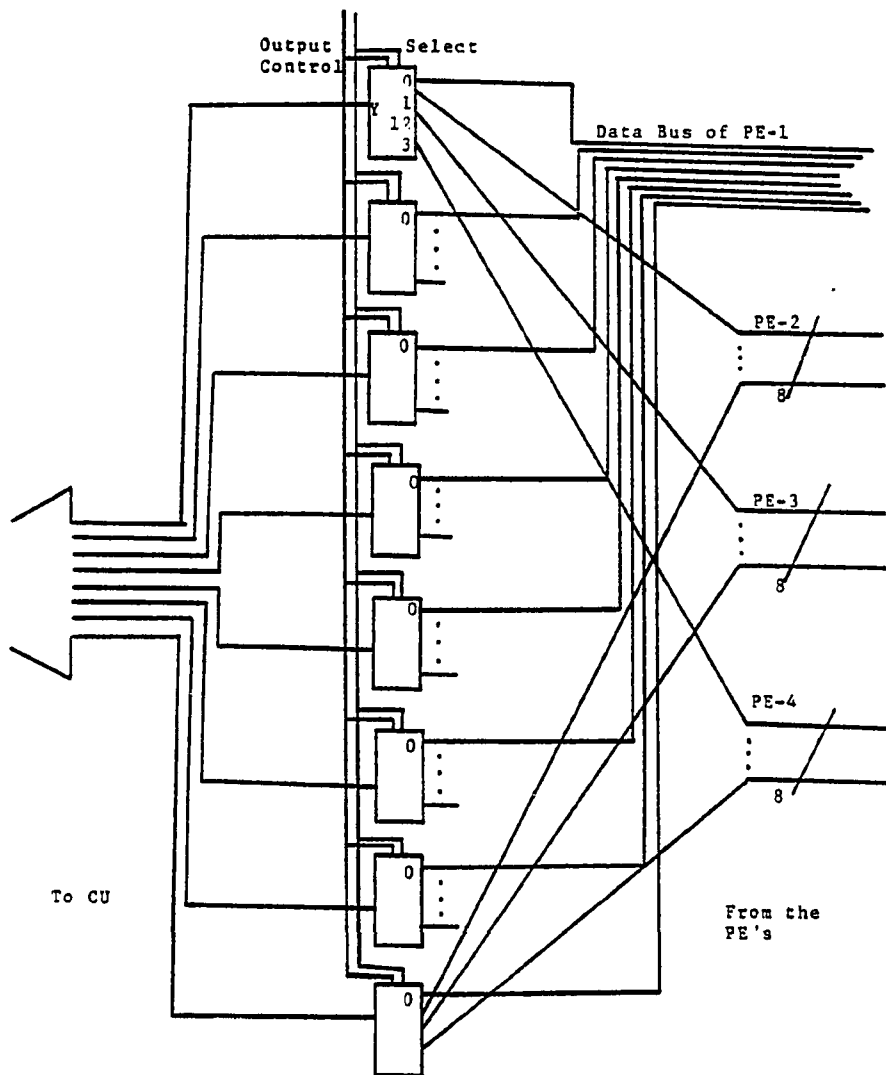


Figure 36: The Arrangement of the Data Mux for a 4-PE Array System

It is clear that the second network is far less expensive than the first one. For a data bus width of 8-bits,  $G_a = 256$  gates, whereas  $G_b = 48$  gates. Furthermore, for bit-slice applications the second is more convenient to use for it takes less microcode bits to control. More specifically, in the first case it takes  $G_a/w = 2n^2 = 32$  bits, and in the second it takes  $G_b/w = 2(n-1) = 6$  bits. However, the main disadvantage with the second is the delay associated with it when implemented in large networks.

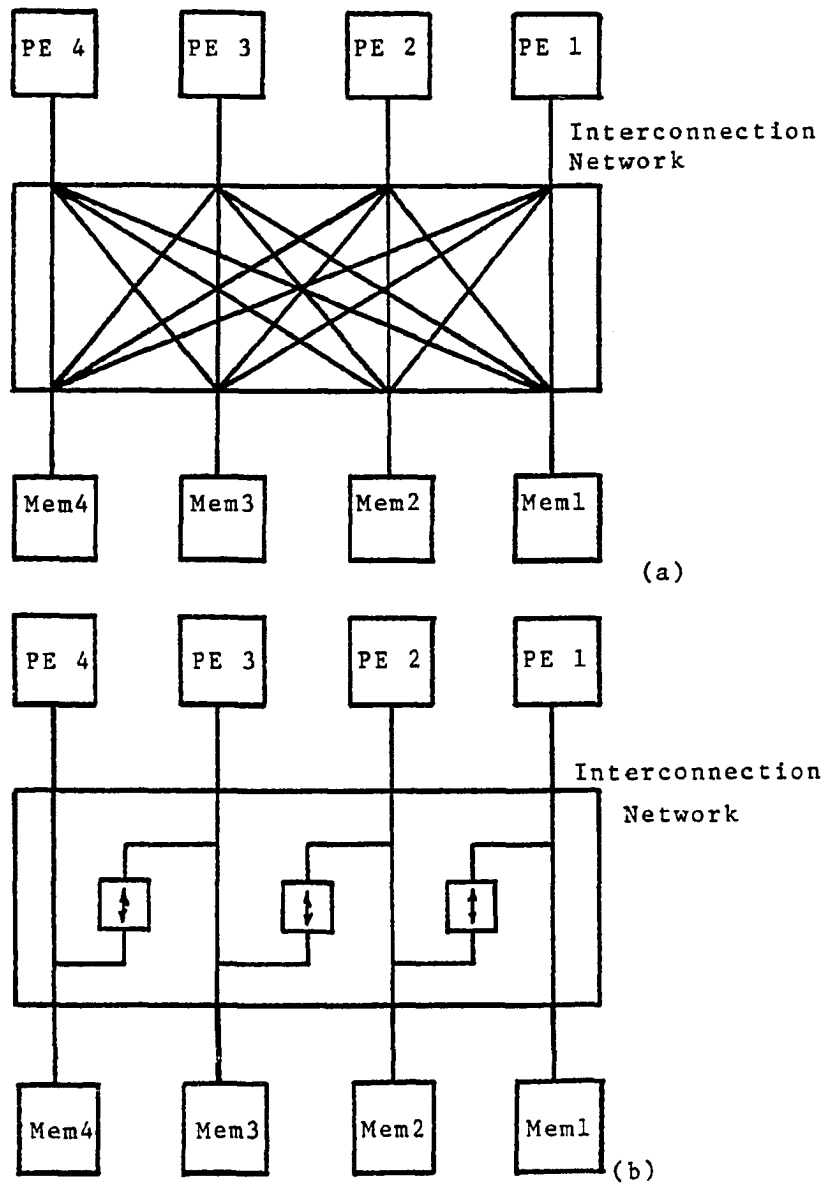


Figure 37: The Two Interconnection Networks

#### 6.4 ANALYSIS OF THE ARRAY MACHINE

The array machine can be viewed analytically, as in Figure 38. The incoming jobs pass through the queue and wait until the controller is free. After the required number of PE's are available, then they are allocated and the control unit is employed to that particular job.

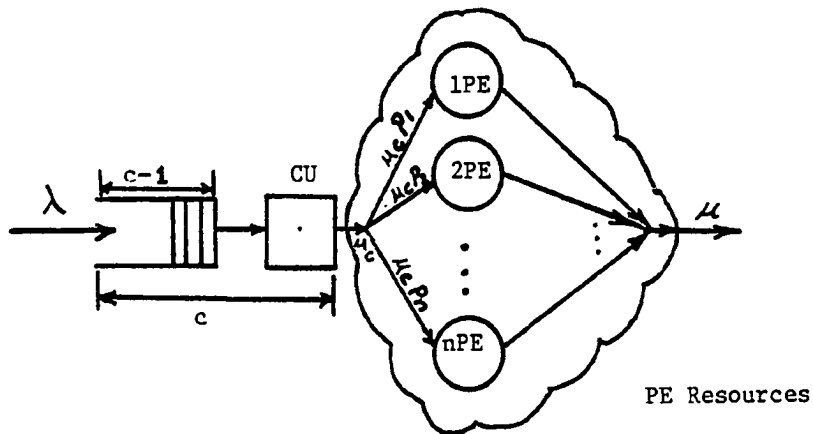


Figure 38: The Queueing System

Each job selects a number of PE's equal to  $X$ ; where  $X$  is a random number ranging between 2 and  $n$ , i.e.,

$$2 < X \leq n$$

#### 6.4.1 The Analytic Model

The model of Figure 38 is first analyzed analytically. In order to analyze this system, we need to derive the state probabilities. The state transition diagram for this system is somewhat similar to the one of chapter V except that all of the PE's have the same service rate since they are executing the same microinstruction. Figure 39 shows a reduced state transition diagram for  $n=3$  and system capacity  $c$ . Hwang and Lee, [HWAN79] and [HWAN81], have analyzed an array system with a multiple control unit for the PM4 system [BRIG79].

It is very important to notice the dissimilarity between this model and that of the modular organization presented in chapter V. In this model the probability vector  $[PV]$  specifies the probability of selecting only a particular number of PE's, i.e., allocating 2,3,...,n PE's to the job, whereas in the other model,  $[PV]$  specifies the probability of selecting only one type of PE, i.e. ADD, SUB, MUL-PE,... etc, by the instruction. We will not present the details of obtaining the state probabilities since the method is very similar to the one done in chapter V. Instead, it suffices to give the general final form:

$$P_{ij}(\lambda + \mu) = P_{0j} \lambda p_j + \mu p_j \sum_{i=1}^n P_{2i} \quad 6-0$$

for  $j=1, \dots, n$

$$P_{kj}(\lambda + \mu) = P_{k-1,j} \lambda + \mu p_j \sum_{i=1}^n P_{k+1,i} \quad 6-1$$

for  $k=2, \dots, c-1$  and  $j=1, \dots, n$

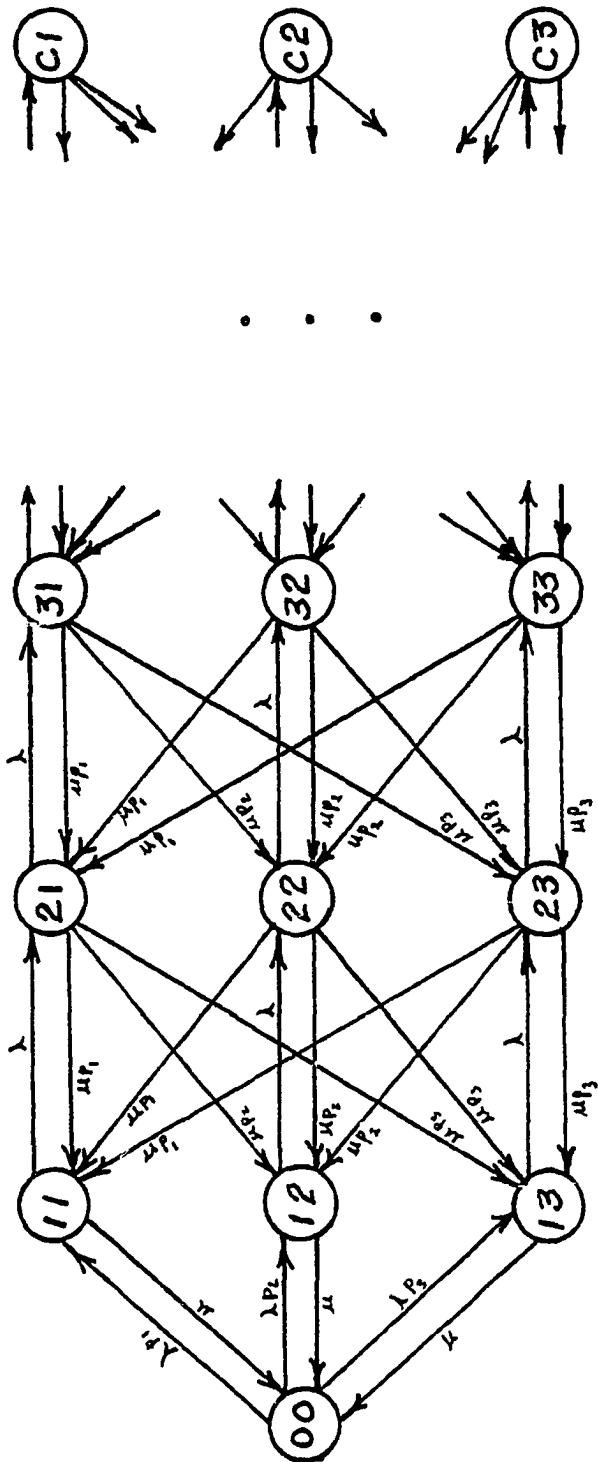


Figure 39: The State Diagram for the System

and finally:

$$P_{cj} = (\lambda / \mu) P_{c-1,j} \quad 6-2$$

for  $j=1, \dots, n$

As before, it is convenient to use matrix forms for the calculation

$$\begin{bmatrix} P_{11} \\ P_{12} \\ \cdot \\ \cdot \\ P_{1n} \end{bmatrix} = [B]^{-1} \left\{ [A] \begin{bmatrix} P_{21} \\ P_{22} \\ \cdot \\ \cdot \\ P_{2n} \end{bmatrix} + \lambda P_0 [I] \begin{bmatrix} P_1 \\ P_2 \\ \cdot \\ \cdot \\ P_n \end{bmatrix} \right\} \quad 6-3$$

$$\begin{bmatrix} P_{21} \\ P_{22} \\ \cdot \\ \cdot \\ P_{2n} \end{bmatrix} = [B]^{-1} \left\{ [A] \begin{bmatrix} P_{31} \\ P_{32} \\ \cdot \\ \cdot \\ P_{3n} \end{bmatrix} + [L] \begin{bmatrix} P_{11} \\ P_{12} \\ \cdot \\ \cdot \\ P_{1n} \end{bmatrix} \right\} \quad 6-4$$

$$\begin{bmatrix} P_{c-1,1} \\ P_{c-1,2} \\ \cdot \\ \cdot \\ P_{c-1,n} \end{bmatrix} = [B]^{-1} \left\{ [A] \begin{bmatrix} P_{c,1} \\ P_{c,2} \\ \cdot \\ \cdot \\ P_{cn} \end{bmatrix} + [L] \begin{bmatrix} P_{c-2,1} \\ P_{c-2,2} \\ \cdot \\ \cdot \\ P_{c-2,n} \end{bmatrix} \right\} \quad 6-5$$

$$\begin{bmatrix} P_{c1} \\ P_{c2} \\ \cdot \\ \cdot \\ P_{cn} \end{bmatrix} = \lambda \begin{bmatrix} \mu & 0 & \dots & 0 \\ 0 & \mu & \dots & 0 \\ \cdot & \cdot & \dots & \cdot \\ 0 & 0 & \dots & \mu \end{bmatrix}^{-1} \begin{bmatrix} P_{c-1,1} \\ P_{c-1,2} \\ \cdot \\ \cdot \\ P_{c-1,n} \end{bmatrix} \quad 6-6$$

where the matrices A , B , and L are defined as:

$$A = \begin{bmatrix} \mu p_1 & \mu p_1 & \cdot & \cdot & \cdot & \mu p_1 \\ \mu p_2 & \mu p_2 & \cdot & \cdot & \cdot & \mu p_2 \\ \cdot & & \cdot & & & \cdot \\ \cdot & & & \cdot & & \cdot \\ \cdot & & & & \cdot & \cdot \\ \mu p_n & \mu p_n & \cdot & \cdot & \cdot & \mu p_n \end{bmatrix} \quad \text{an nxn matrix}$$

$$B = \begin{bmatrix} \lambda + \mu & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \lambda + \mu & \cdot & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & \cdot \\ \cdot & & & \cdot & & \cdot \\ \cdot & & & & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot & \cdot & \lambda + \mu \end{bmatrix} \quad \text{an nxn matrix}$$

$$L = \lambda [I] = \begin{bmatrix} \lambda & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \lambda & \cdot & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & \cdot \\ \cdot & & & \cdot & & \cdot \\ \cdot & & & & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot & \cdot & \lambda \end{bmatrix} \quad \text{an nxn matrix}$$

All the state probabilities (  $\neq i$ 's) are in term of  $P_0$ :

$$[[P_{1i}'s], [P_{2i}'s], \dots, [P_{ci}'s]] \quad 6-7$$

Then  $P_0$  is solved for as before

$$P_0 + \sum_{j=1}^c \sum_{i=1}^n P_{ji} = 1 \quad 6-8$$

$$P_0 [ 1 + \sum_{j=1}^c \sum_{i=1}^n P_{ji}' ] = 1 \quad 6-9$$



$$P_o = [ 1 + \sum_{j=1}^c \sum_{i=1}^n P_{ji}' ]^{-1} \tag{6-10}$$

where  $P_{ij} = P_o P'_{ij}$ . Therefore,  $P_o$  is now known and can be used to find the  $P'_{ij}$ .

Then eq. 6-7 becomes:

$$[[ P'_{1,i's}], [ P'_{2,i's}], \dots, [ P'_{c,i's}]] \tag{6-11}$$

from Eq. 6-11 we can write a  $P'$  matrix which consists of  $n$  rows and  $c$  columns. In fact, the  $P'$  matrix is the state probability matrix.

	Number of PE's allocated	Number of jobs in the system	1	2	3	h	c-1	c		
$P =$	1	P'_{11}	P'_{21}	.	.	P'_{h1}	.	P'_{c-1,1}	P'_{c1}	} 6-12
	2	P'_{12}	P'_{22}			P'_{h2}		P'_{c-1,2}	P'_{c2}	
	3	.							.	
	k	P'_{1k}	P'_{2k}			P'_{hk}		P'_{c-1,k}	P'_{ck}	
	.	.							.	
	n	P'_{1n}	P'_{2n}	.	.	P'_{hn}	.	P'_{c-1,n}	P'_{cn}	

where  $1 \leq h \leq c$  and  $1 \leq k \leq n$

where  $P_{h,k}$  expresses the probability of the system being in state  $h,k$  i.e.  $h$  jobs are in the system and  $k$  processors are allocated to the one being served. In order to find the probability of PE allocation, the  $P'$  matrix is used quite effectively. For example:  $\sum_{i=1}^n P_{ik}$  equals the probability of  $k$  processing elements being allocated in the system.

The state transition matrix is shown to be somewhat complex. The dimension of the state transition matrix ( $Q$ ) is  $((nxc)+1) \times ((nxc)+1)$ . In equation 6-13, it is shown

that the state transition matrix (crossing out the first row and column) consists of  $c$  square submatrices each with a dimension of  $n \times n$ . Even for a small number of PE's, the number of states will be high and the number of possible transitions will grow very rapidly. Specifically the number of states is  $(n+c)+1$  and the number of transitions equal to

$$(3c-2)(n)^2 + 2(n+c)+1 \quad 6-12.a$$

Ideally, there should be

$$(cn)^2 + 2(cn)+1 \quad 6-12.b$$

transitions, but due to the fact that the system can go forward to one state only, then the reduced number of transitions (in A) is true. The number of zero (impossible) transitions is found by subtracting Eq-12.a from Eq-12.b, and is found to be

$$cn(cn-3n)+2n^2 \quad 6-12.c$$

	$0 \quad 1-\lambda t e^{\lambda t}$	$P_1 \lambda e^{-t \lambda} \dots P_n \lambda e^{-t \lambda}$	0	0	0	0	0	0	0	0	0	0	0
$11$	$\mu e^{-t \mu}$												
	$\vdots$	$\frac{A}{-1}$	$\frac{B}{-1}$	0	0	0	0	0	0	0	0	0	0
	$\mu e^{-t \mu}$												
$1n$													
$21$													
$2n$	0	X	$\frac{A}{-2}$	$\frac{B}{-2}$	0	0	0	0	0	0	0	0	0
$31$													
$3n$	0	0	$\frac{A}{-3}$	$\frac{B}{-3}$	0	0	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$											$\vdots$
$\vdots$	$\vdots$	$\vdots$											$\vdots$
$\vdots$	$\vdots$	$\vdots$											$\vdots$
$c-1,1$	0	0	0	0	0	0	0	0	0	X	$\frac{A}{c-1}$	$\frac{B}{c-1}$	
$c-n,1$													
$c,1$													
$c,n$	0	0	0	0	0	0	0	0	0	0	X	Y	

$[Q] =$

$$\frac{A}{i} = \left. \begin{array}{l} i1 \\ i2 \\ \cdot \\ \cdot \\ in \end{array} \right\} \left( \begin{array}{cccc} 1 - (\mu e^{-\mu t} + \frac{(\lambda t)}{i!} e^{-\lambda t}) & 0 & \cdot & \cdot & 0 \\ 0 & 1 - (\mu e^{-\mu t} + \frac{(\lambda t)}{i!} e^{-\lambda t}) & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot & 1 - (\mu e^{-\mu t} + \frac{(\lambda t)}{i!} e^{-\lambda t}) \end{array} \right)$$

$i = 1, 2, \dots, c$

$$X = \left. \begin{array}{l} i1 \\ i2 \\ \cdot \\ \cdot \\ in \end{array} \right\} \left( \begin{array}{cccc} p_1 \mu e^{-\mu t} & p_2 \mu e^{-\mu t} & \cdot & \cdot & p_n \mu e^{-\mu t} \\ p_1 \mu e^{-\mu t} & p_2 \mu e^{-\mu t} & \cdot & \cdot & p_n \mu e^{-\mu t} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ p_1 \mu e^{-\mu t} & p_2 \mu e^{-\mu t} & \cdot & \cdot & p_n \mu e^{-\mu t} \end{array} \right)$$

$i = 2, 3, \dots, c$

$$\frac{B}{i} = \left. \begin{array}{l} i1 \\ i2 \\ \cdot \\ \cdot \\ in \end{array} \right\} \left( \begin{array}{cccc} \frac{(\tau \lambda)^i}{i!} e^{-\tau \lambda} & 0 & \cdot & \cdot & 0 \\ 0 & \frac{(\tau \lambda)^i}{i!} e^{-\tau \lambda} & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & \frac{(\tau \lambda)^i}{i!} e^{-\tau \lambda} \end{array} \right)$$

$$Y = \left. \begin{array}{l} c1 \\ c2 \\ \cdot \\ \cdot \\ cn \end{array} \right\} \left( \begin{array}{cccc} 1 - \mu e^{-\mu t} & 0 & \cdot & \cdot & 0 \\ 0 & 1 - \mu e^{-\mu t} & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot & 1 - \mu e^{-\mu t} \end{array} \right)$$

Note: other than the first row and first column, all other submatrices are of dimensions  $n \times n$  and we have  $c^2$  of these

submatrices. Also observe that the principle diagonal and the upper diagonal are nonzero, all other entries are zero.

The utilization of the PE's is defined as:

$$\psi = \frac{\text{Expected no of PE's in the busy state}}{\text{Total no of PE's in the system}}$$

$$\text{where } \psi = (1P_{11} + 2P_{12} + \dots + nP_{1n}) + (1P_{21} + 2P_{22} + \dots + nP_{2n}) + \dots + (1P_{c1} + 2P_{c2} + \dots + nP_{cn})$$

$$\text{or } \psi = \sum_{i=1}^c \sum_{j=1}^n j \cdot P_{ij} \quad 6-14a$$

$$\rho_{PE} = (1/n) \sum_{i=1}^c \sum_{j=1}^n j \cdot P_{ij} \quad 6-14b$$

The system utilization, as generated in chapter V, is given by

$$\rho_{sys} = (1 - P_0) \quad 6-15$$

therefore, system throughput is

$$T_{sys} = (\rho_{sys}) \times \mu \quad 6-16$$

The average number of jobs waiting in the queue is given

by

$$N_q = E[L_q] = 0 \cdot P_{1j} + 1 \cdot P_{2j} + 2 \cdot P_{3j} + \dots + (c-1) \cdot P_{cj}$$

$$j=1, \dots, n$$

$$\text{i.e., } N_q = \sum_{j=1}^n \sum_{i=1}^c (i-1) \cdot P_{ij} \quad 6-17$$

Note that if there are  $i$  jobs in the system, then, there are  $(i-1)$  jobs waiting in the queue. The average number of jobs in the system (in queue plus in service) is

$$N_{sys} = \sum_{j=1}^n \sum_{i=1}^c i \cdot P_{ij} \quad 6-18$$

Thus by subtracting Eq-17 from Eq-18, we obtain the average number of jobs in service.

In order to calculate the average job response time, Little's formula is applied.

$$\text{average job response time} = T_w = \frac{\text{number in system}}{\text{actual arrival rate}}$$

$$T_w = (N_{\text{sys}} / \lambda_a) = (N_q + N_s) / \lambda_a = (N_q + 1) / \lambda_a \quad 6-19$$

where  $\lambda_a$  is the actual arrival rate, and is found by the following,

$$\begin{aligned} \lambda_a &= \text{the ideal arrival rate} \times \text{the probability the system is not full} \\ \lambda_a &= \lambda (1 - [P_{c1} + P_{c2} + \dots + P_{cn}]) \\ \lambda_a &= \lambda (1 - \sum_{j=1}^n P_{cj}) \end{aligned} \quad 6-20$$

where  $\sum_{j=1}^n P_{cj}$  is the probability that the system has a full queue and a busy group of  $j$  servers, finally,  $T_w$  is found as

$$T_w = (1 + \sum_{j=1}^n \sum_{i=1}^c (i-1) \cdot P_{ij}) / (\lambda (1 - \sum_{j=1}^n P_{cj})) \quad 6-21$$

From Eq 6-19 we see how to obtain the average time spent in the queue and in the system. The above system of equations are solved using APL, the program listing is shown in Appendix D.

#### 6.4.2 The Simulation Model

The simulation model for the array model is somewhat similar to that of the controlled multiserver model. The major distinction in this model is the allocation of the number of PE for each incoming job. For an  $n$ -PE system,

each job is allowed to seize between two and  $n$  PE's. Note that at any given instant, the single control unit employed can only serve one groupe of PE's, and the rest of the PE's will remain idle.

Simulation is performed on two levels, macro and micro. The macro level is used to validate the analytic model. The macro model is associated with the jobs in the model, whereas the micro model is associated with the macroinstructions execution. The macro analysis flowchart is given in Figure 40.

On the other hand, the micro model consists of two segments. One segment is concerned with the job arrival, and the other segment is associated with the macroinstruction in that particular job. The two segments work interactively. The overall flowchart for the simulation model is shown in Figure 41.

A Poisson arrival and exponential service time distribution are assumed. The number of jobs in the system is not that critical in the analysis of the micro model. However, the number of macroinstructions in a job is of concern for it will have a direct effect on the performance measure. As mentioned earlier, the controller can only serve one job at a time, and as long as it is busy serving that job, it will do that until completion, nonpreemptive. The results of the analysis are discussed in section 6.6. The simulation programs are given in Appendix E.

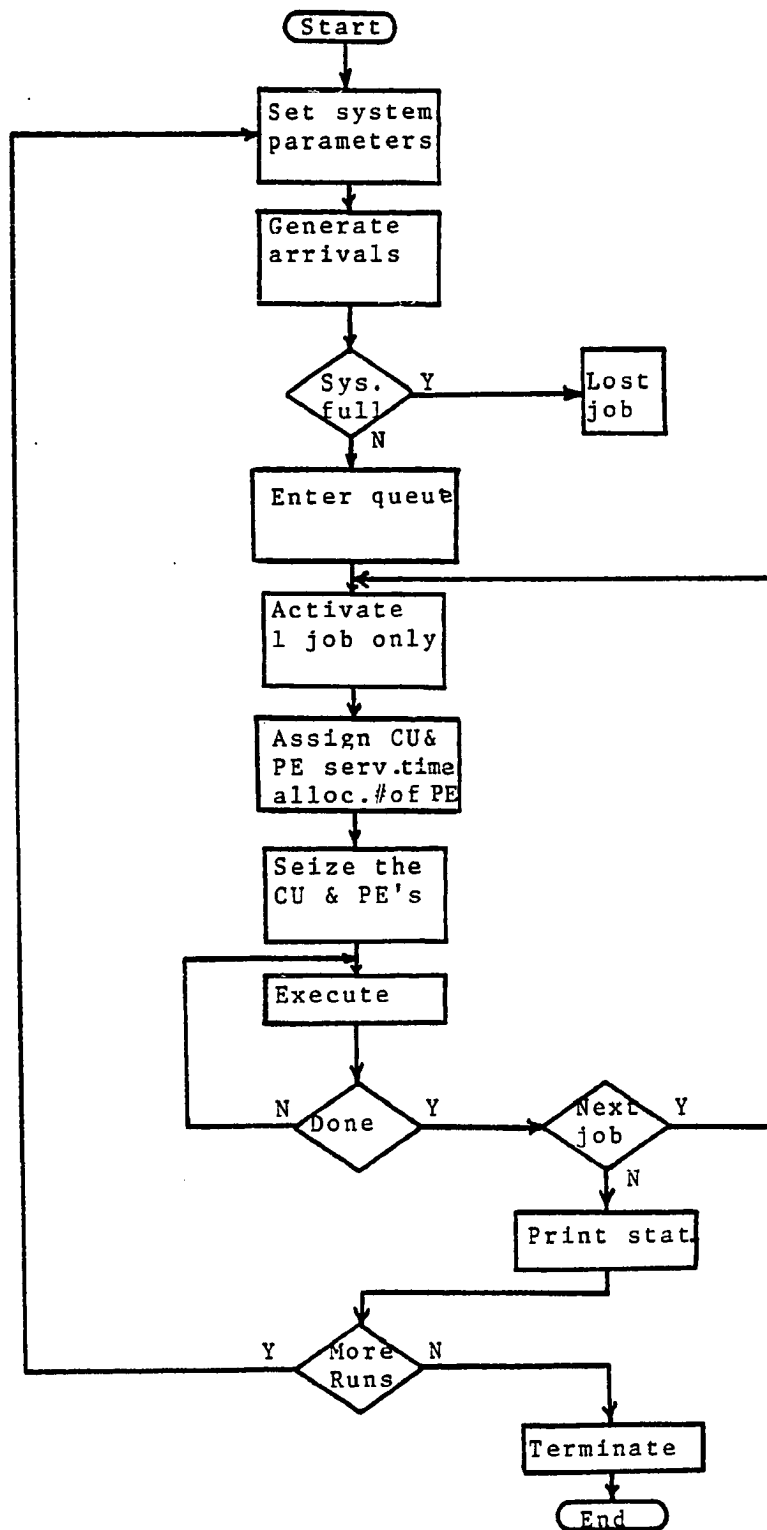


Figure 40: The Simulation Flowchart for the Macro Model



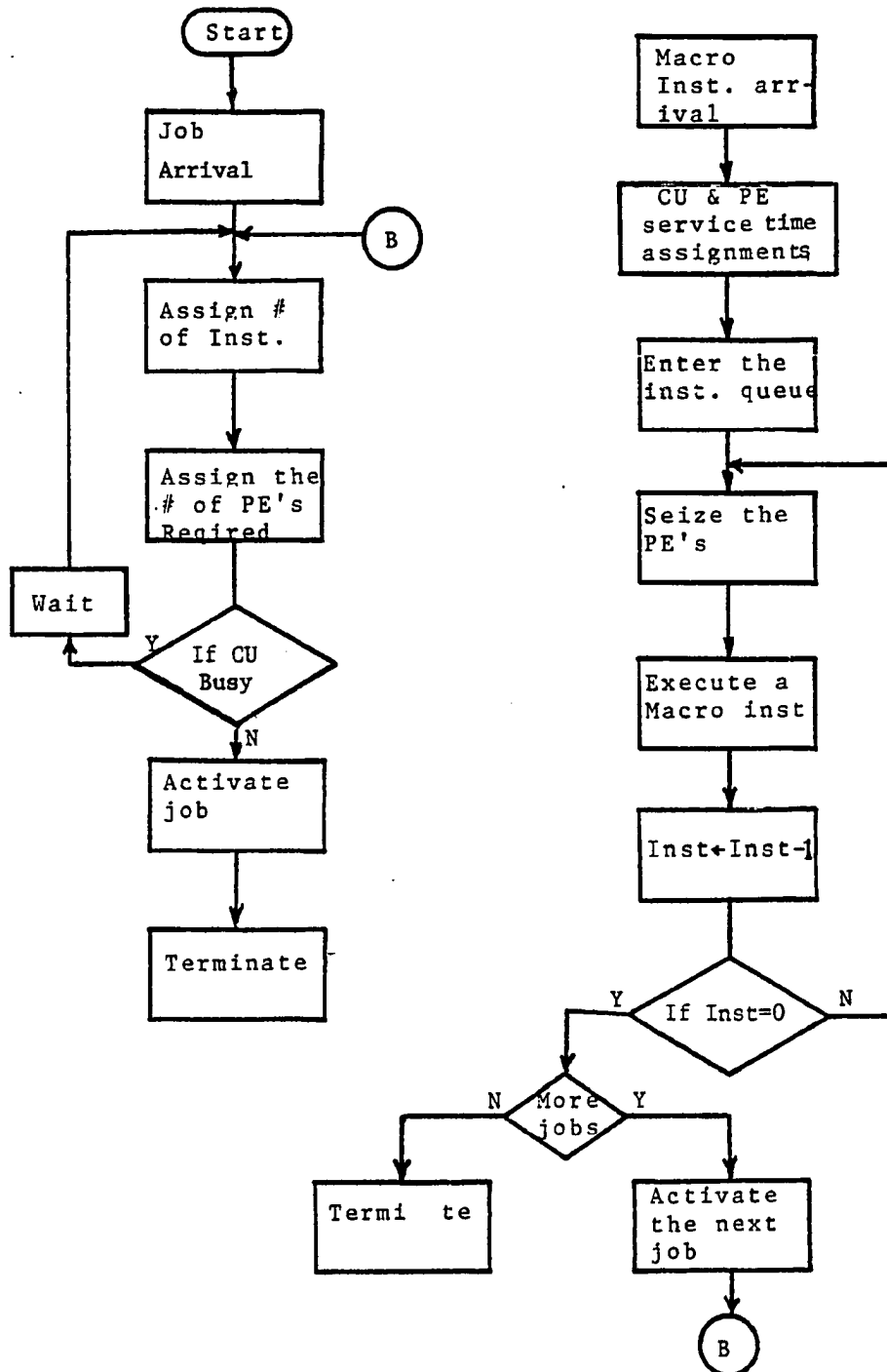


Figure 41: The Composite Flowchart for the Array Model

## 6.5 POSSIBLE APPLICATION EXAMPLES

Writing algorithms to maximize the utilization of the array system is one of the most challenging tasks. In the given organization, each PE can have access to all of the PE's in the system. Clearly, this kind of interconnection suffers from a delay problem when dealing with large numbers of PE's.

The system described above can be used in the following organization. Consider outside jobs that are arriving to the system queue as transactions. A number of PE's will be allocated to each job under execution. This is an open network type. The jobs are assumed to perform vector operation only. Each job is considered not to utilize the full power of the PE's. In some cases, and for a particular job, a subset of the whole PE set might be used and the remaining PE's are left idle, thus reducing the total PE utilization. Not using the full power of the system is clearly an undesirable drawback. This fact by itself constitutes the major disadvantage of this kind of organization. Two application examples that use this kind of systems will be illustrated in detail.

### Example 1:

Consider an array system that is used to read data from several locations such as in a weather station or from several identical sources (satellite and radar tracking stations). We are interested in finding the average data

read in each case.

The program for doing this is the same for all of the PE's (single instruction stream), but each PE has its own data path (multiple data streams). Let us make the following assumptions: the number of PE's in the system are known and are even. The different data are stored in the local memory of each PE. After each set of data is read, the vector addition is performed and the final result is stored in the main memory for further calculations. The intermediate results, however, are stored in the lower indexed PE of each PE pair, i.e.  $[PE_i] \leftarrow [PE_i] + PE_{[i+1]}$  for  $i=1,2,\dots,n-1$ . The flowchart of Figure 42 illustrates the procedure in detail. The notation  $\lceil \log_2 M \rceil$  signifies the ceiling of  $\log_2 M$  (the next higher digit that is greater than or equal to  $\log_2 M$ ), and  $M$  is the number of elements in the vector. Furthermore, assume that the number of processing elements in the system are equal to the number of elements in the vector.

It is noted that in each subaddition, the number of components in the  $[I]$  vector will reduce by a factor of one half. The utilization of the PE's is 100% in the first step, 50% in the second step, etc. Mathematically, the utilization can be expressed as:

$$P_j = (1/2^{j-1}) \times 100 \quad \text{for } j=1,2,\dots,\log_2 M$$

where  $j$  indicates the step number. The average total utilization is given by

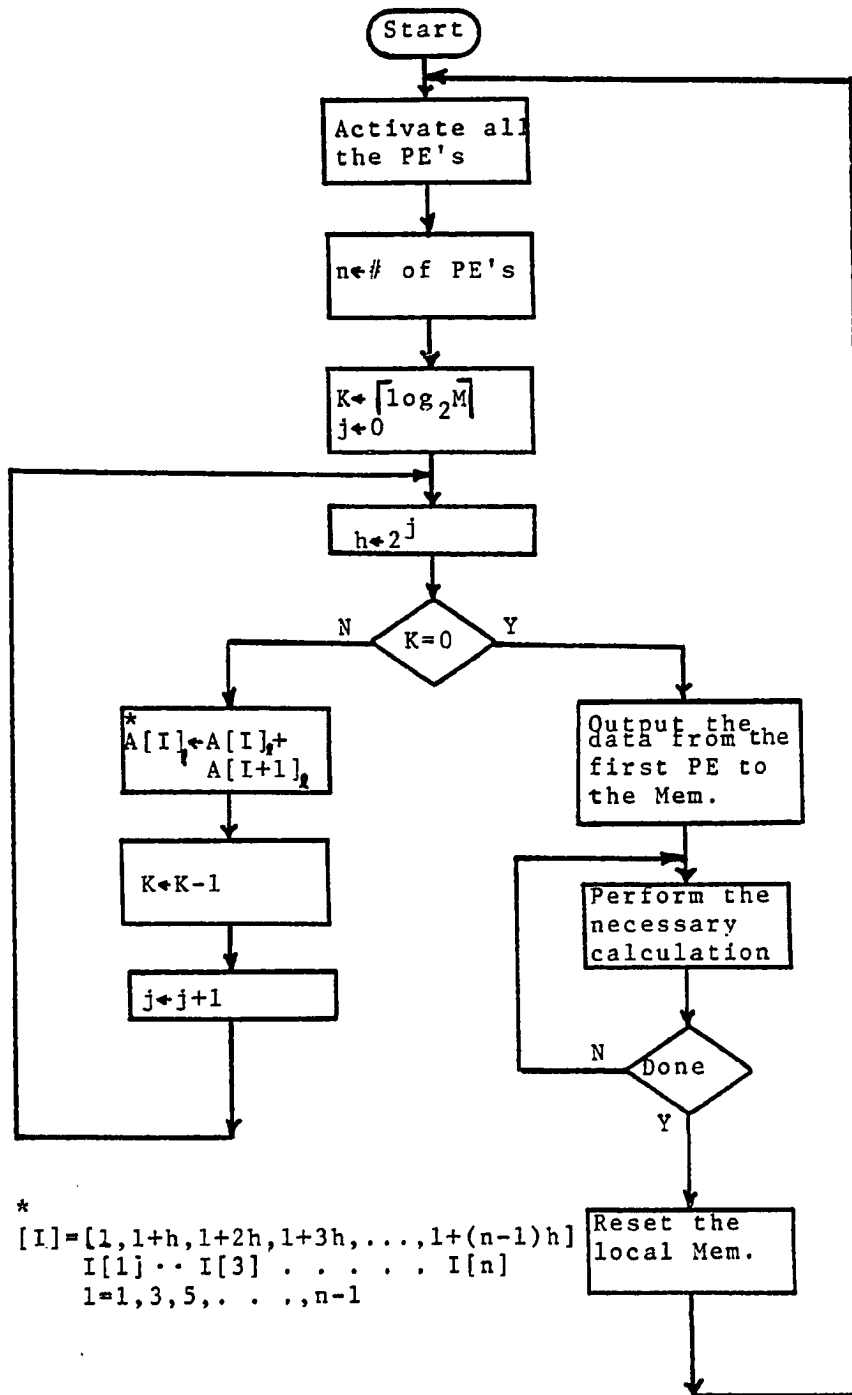


Figure 42: The Flowchart for the First Example

$$\begin{aligned} & \left( \sum_{j=1}^{\log_2 M} \rho_j \right) / \log_2 M \\ &= (2 \times 100 / \log_2 M) [1 - (1/2)^{((\log_2 M)+1)}] = (2 \times 100/k) [1 - (1/2)^{K+1}] \end{aligned}$$

where  $K = \log_2 M$ . Comparing the above algorithm with the uniprocessor case we notice that the array system takes  $\log M$  instruction cycles, whereas the uniprocessor case takes  $(M-1)$  instruction cycles. Consequently, one should observe greater significance for higher  $M$ . As an example, for a 16-element vector, the array system will take 4 cycles compared to the 15 cycles in the uniprocessor case. The graph of Figure 43 provides a plot for the above two cases. Moreover, using higher  $M$  will require a higher number of PE's, hence more complex interconnection network.

#### Example 2:

In this case we discuss another possible configuration in which each PE is regarded strictly as an input channel, thus providing a multiport input system. This application is desirable in data acquisition environments. The same program code is used to direct the activities of all the PE's. Each data path will have its own input channel (i.e., its own PE). In this case the number of PE's is assumed to be fixed. This is done at the design stage since the number of data input lines are assumed to be known. In some real time applications this configuration is a typical one. Furthermore, assume that the rate of data input is always

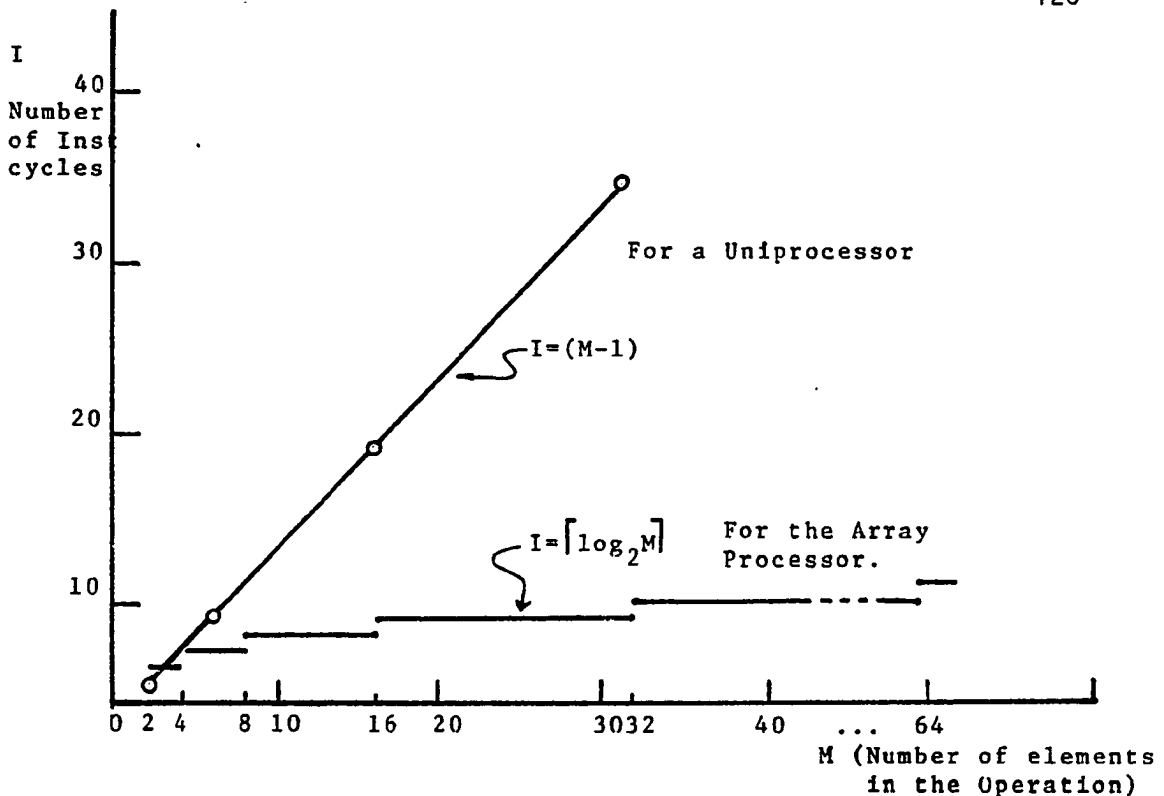


Figure 43: The Uniprocessor Versus the Array System Cycle Requirements

slower than the rate at which the processors execute the program. This last assumption will assure the reading of correct data. The size of the main memory should be large enough in order to accommodate data for the desired period of time. The readings of data and time take place in the processing elements section. The program is executed over and over again, each time reading a different set of data.

The microprogramming ability of the system provides the individual control of the PE's in the processing section and the PE of the control section. The flowchart shown in

Figure 44 precisely illustrates the procedure discussed above. The main and local memories content are shown in Figure 45 and should be read in conjunction with Figure 44.

As a final note on the array system, the following drawbacks and limitations are discussed. The serious problem with the array system in general is that the failure of any of the PE's will jeopardize the operation of the whole system. As a result there should be a supervisor processor that will detect the faulty PE. In case a faulty PE is detected, then, there are two alternatives, either to bring the whole system to a halt ( a clear disadvantage ), or to replace the faulty PE with a standby PE. The idea of having a standby processor is not very favorable for it will complicate the interconnection even further. To be very reliable, there should be more than one PE for each PE location. The array systems are highly specialized. The system, very often, is only suitable for the application it is designed for.

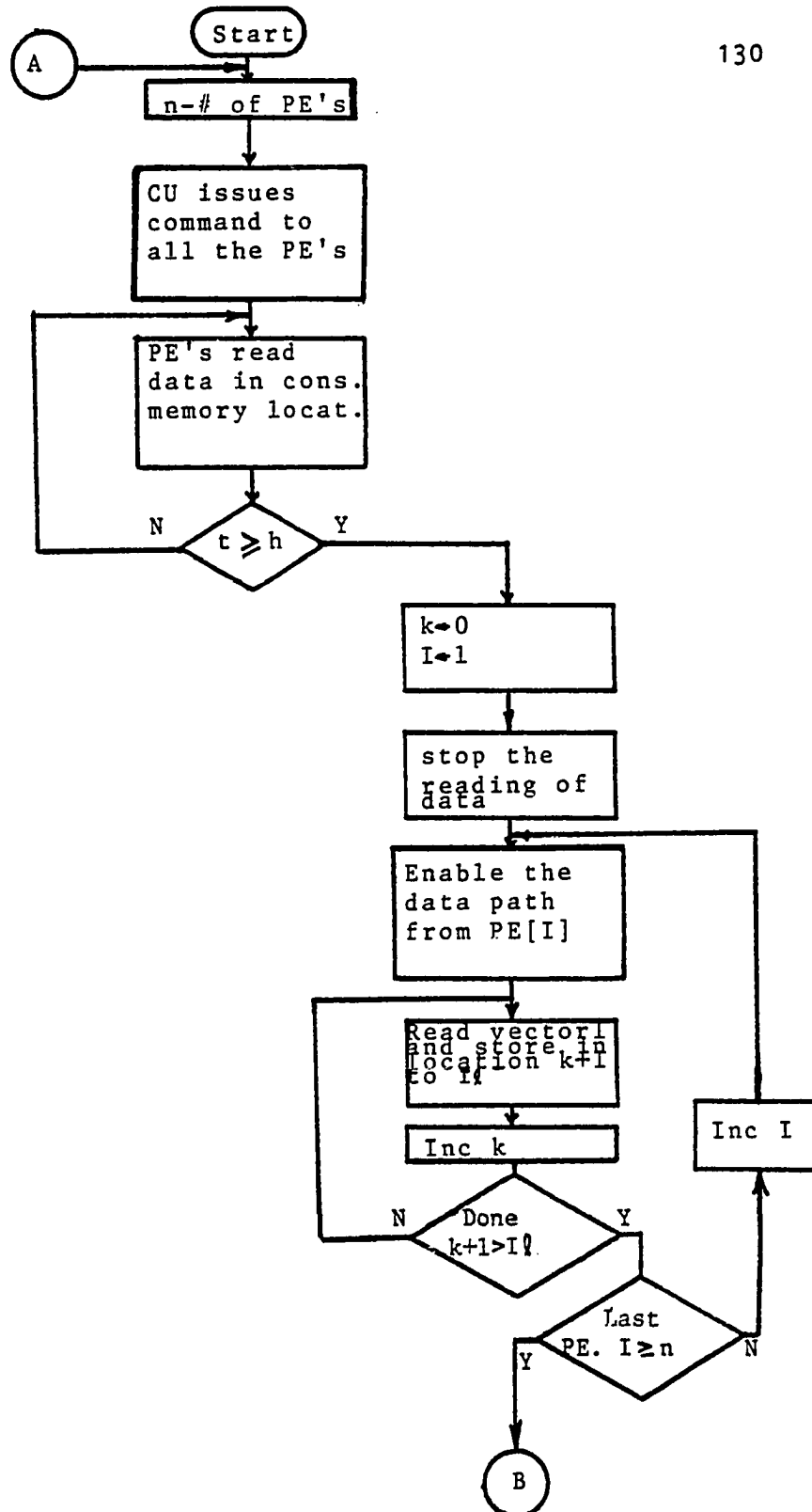
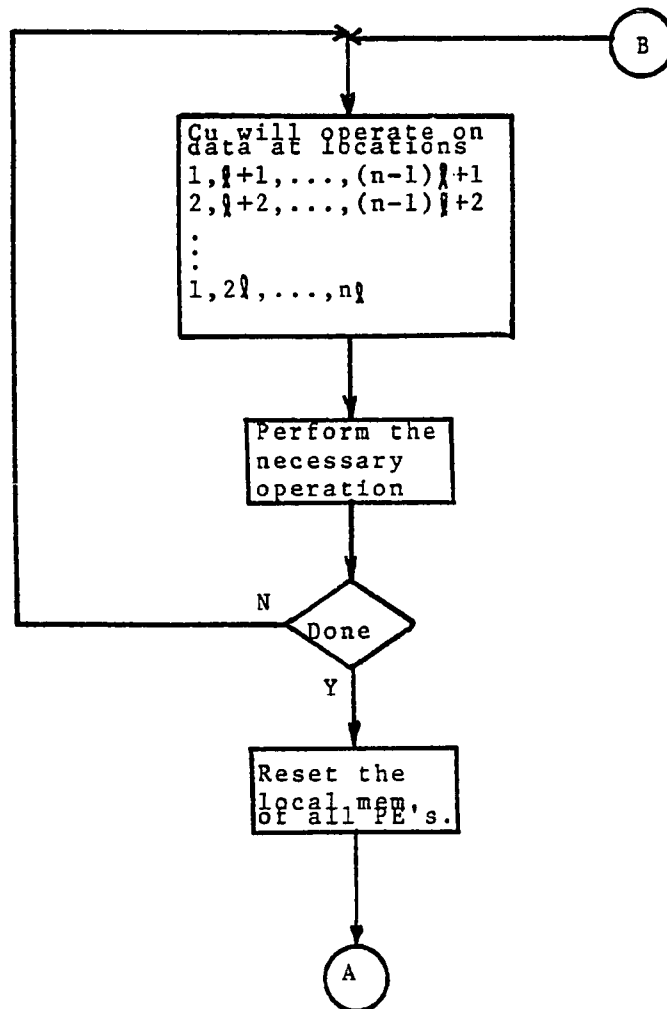


Figure 44: The Flowchart for the Second Example





$n$ : number of PE's  
 $h$ : the time interval for reading the data  
 $l$ : number of memory locations required to store data  
 $l = h/\text{IAT}$  of data  
 $0 \leq k \leq l$  for looping  
 $1 \leq i \leq n$  purposes  
 IAT: interarrival time

Figure 44: (Continued)

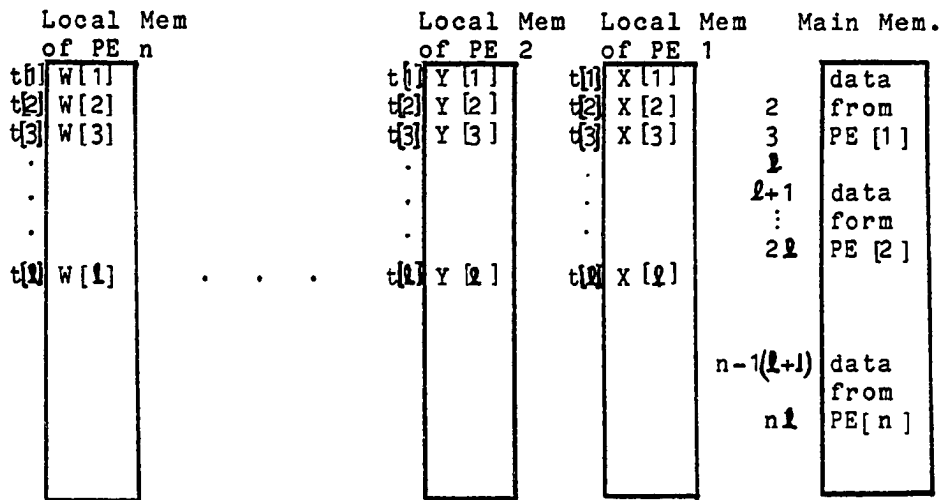


Figure 45: The Main and Local Memories Contents

## 6.6 ANALYSIS OF RESULTS

Two different analyses are performed, macro- and micro-analysis. In the macro analysis both simulation and analytic models are studied. The job as a whole is considered as a transaction unit in the system. The system capacity,  $c$ , the number of PE's, and the arrival rate are the parameters that are varied. The resource utilizations and the average queue length as a function of system capacity and the number of PE's are the main objectives in this analysis.

### 6.6.1 Macro-Analysis of the Array System

In order for a simulation study to be reliable, it is necessary to bring system to a steady-state condition. That is, it is necessary to decide the confidence interval. The confidence interval is estimated by some experimental studies. The simulation model reached the steady-state condition in about 3000 time units, or when about 60 jobs are passed through the system. The analytic model is valid for  $n > 4$ . The simulation results confirm the validity of our queueing model as shown in the different graphs. In Figure 46 the system and PE utilization are plotted as a function of the arrival rate. It is noted that the system utilization is always greater than the PE utilization, since the system utilization includes the CU utilization. The average queue length is shown in Figure 47. The flat region is where the service rate equals approximately the arrival rate. When the number of PE's is varied from 5 to 20, the graphs in Figure 48 are obtained. For a higher system capacity, the utilization of both the system and the PE's are higher. It is worth observing that the average queue length is not dependent on the number of PE's. This is true because the extra PE's introduced are also utilized by the same instruction, as confirmed in Figure 49. Moreover, the system capacity is also varied from 5 to 30. The system and PE utilization, and the average queue length are computed and plotted in Figures 50 and 51, respectively. Finally the

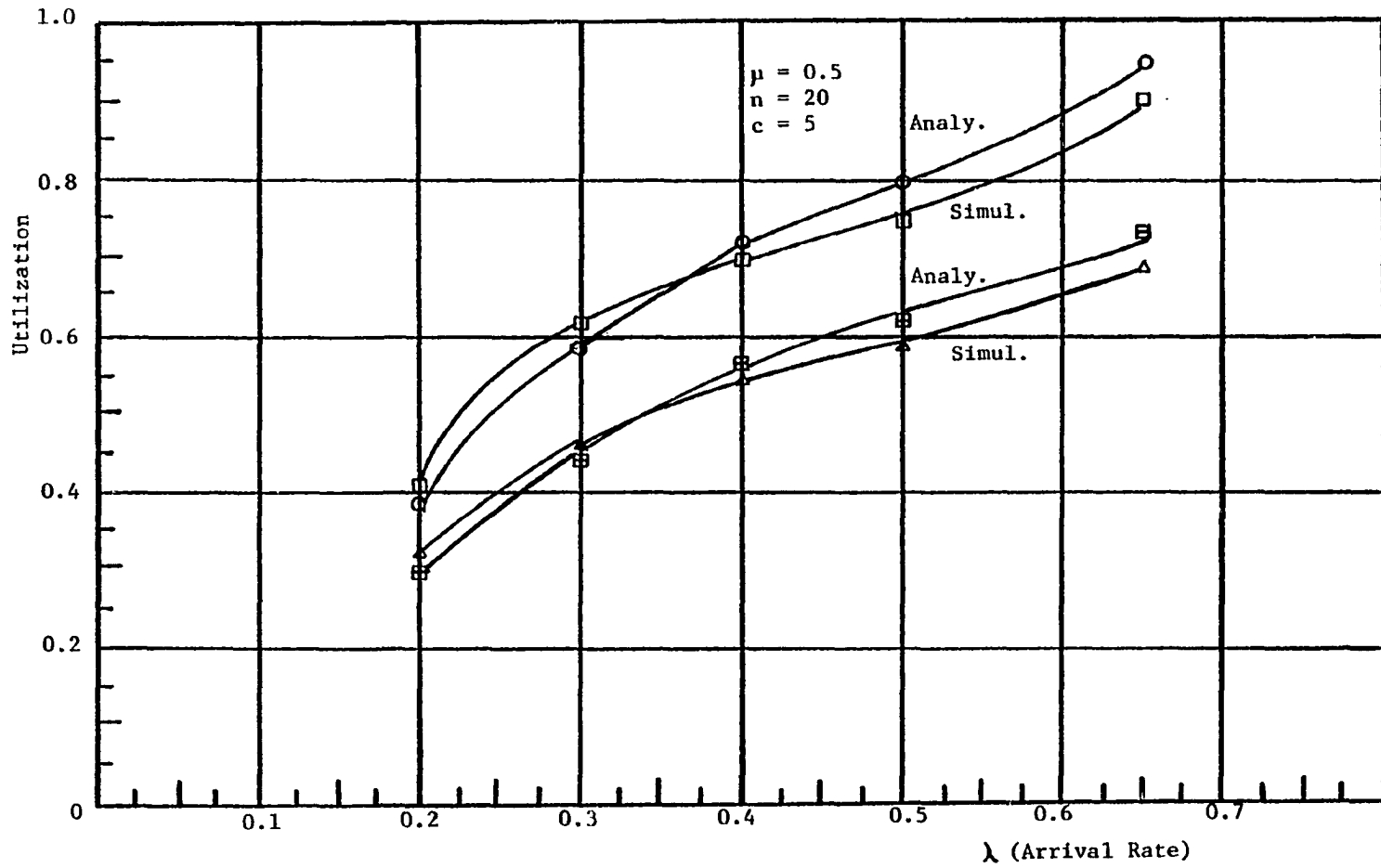


Figure 46: The PE and System Utilization as a Function of  $\lambda$

total system utilization and the PE utilization are plotted for different arrival rates. The higher the arrival rate, the higher the utilization will be, as shown in Figure 52.

The PV (probability selection vector) used is consistent in both the analytic and simulation study. However, the model (especially the analytic) could be simplified a great deal by using a PV of the form:

$PV = [0 \ 0 \ . \ . \ . \ 0 \ 1]$ , that is, the whole PE subset will be allocated to the job under execution. The queue length, consequently, will be independent of the variance in the number of PE's.

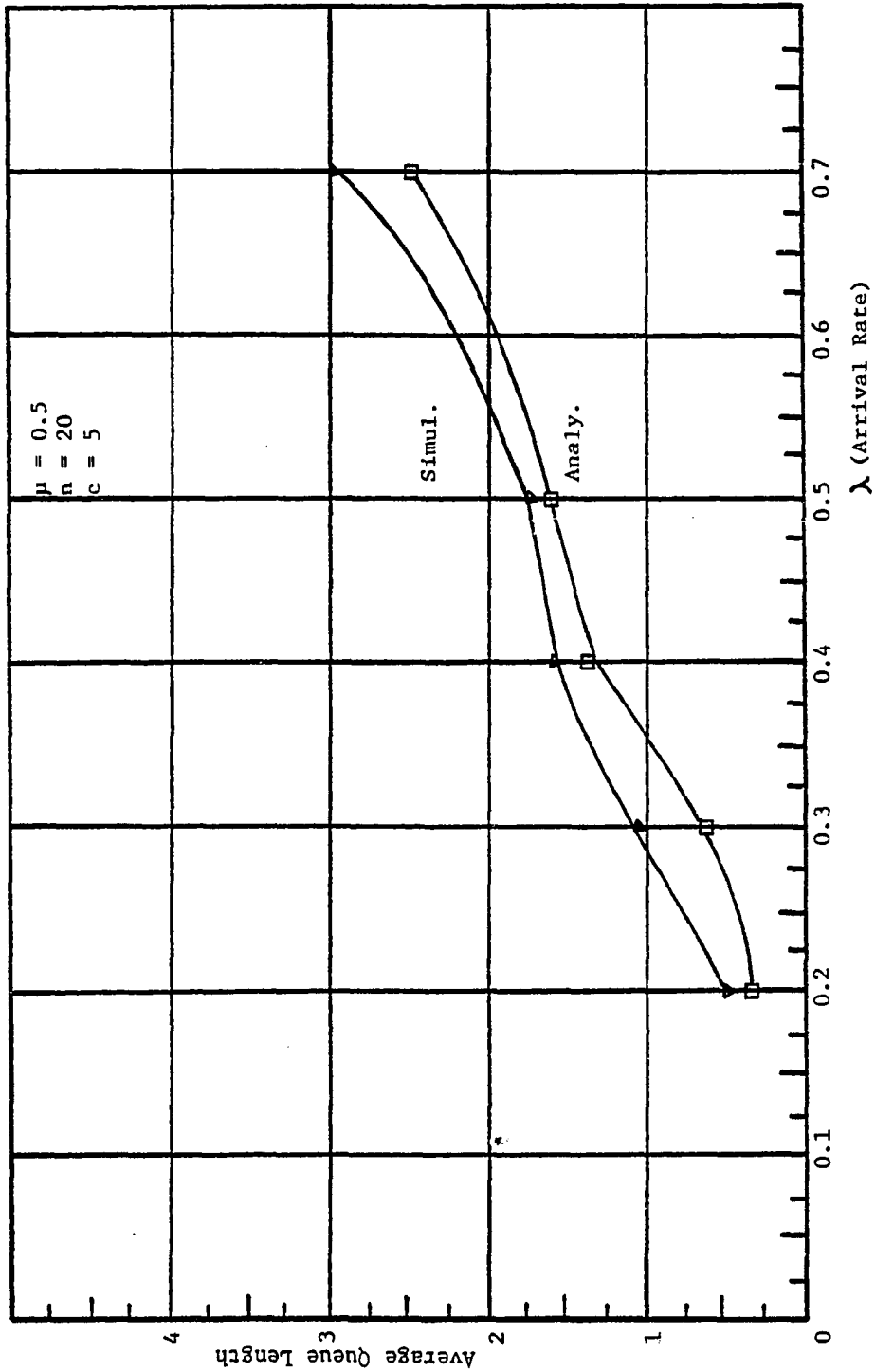


Figure 47: The Average Queue Length as a Function of  $\lambda$

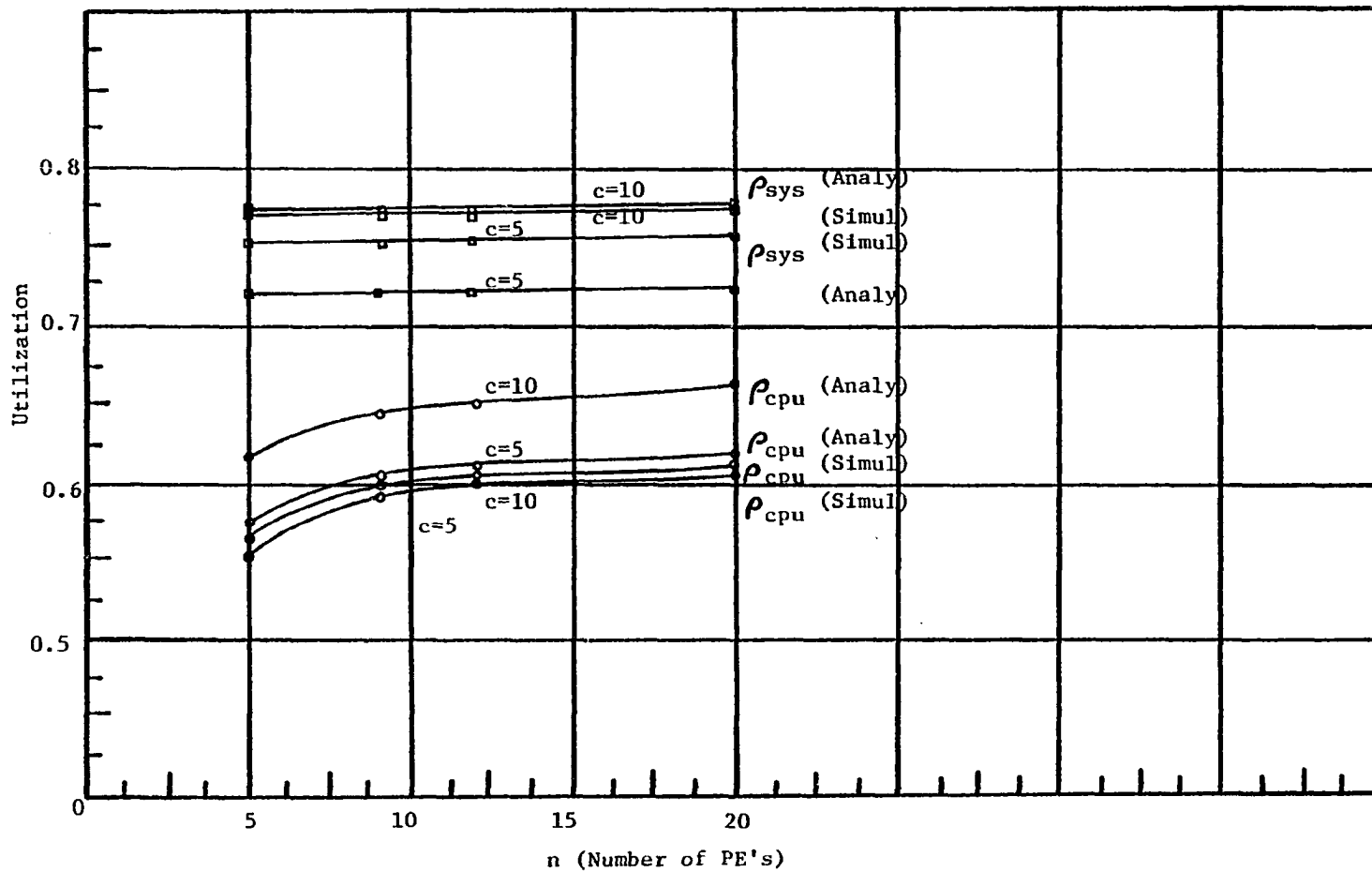


Figure 48: The Utilization as a Function of n

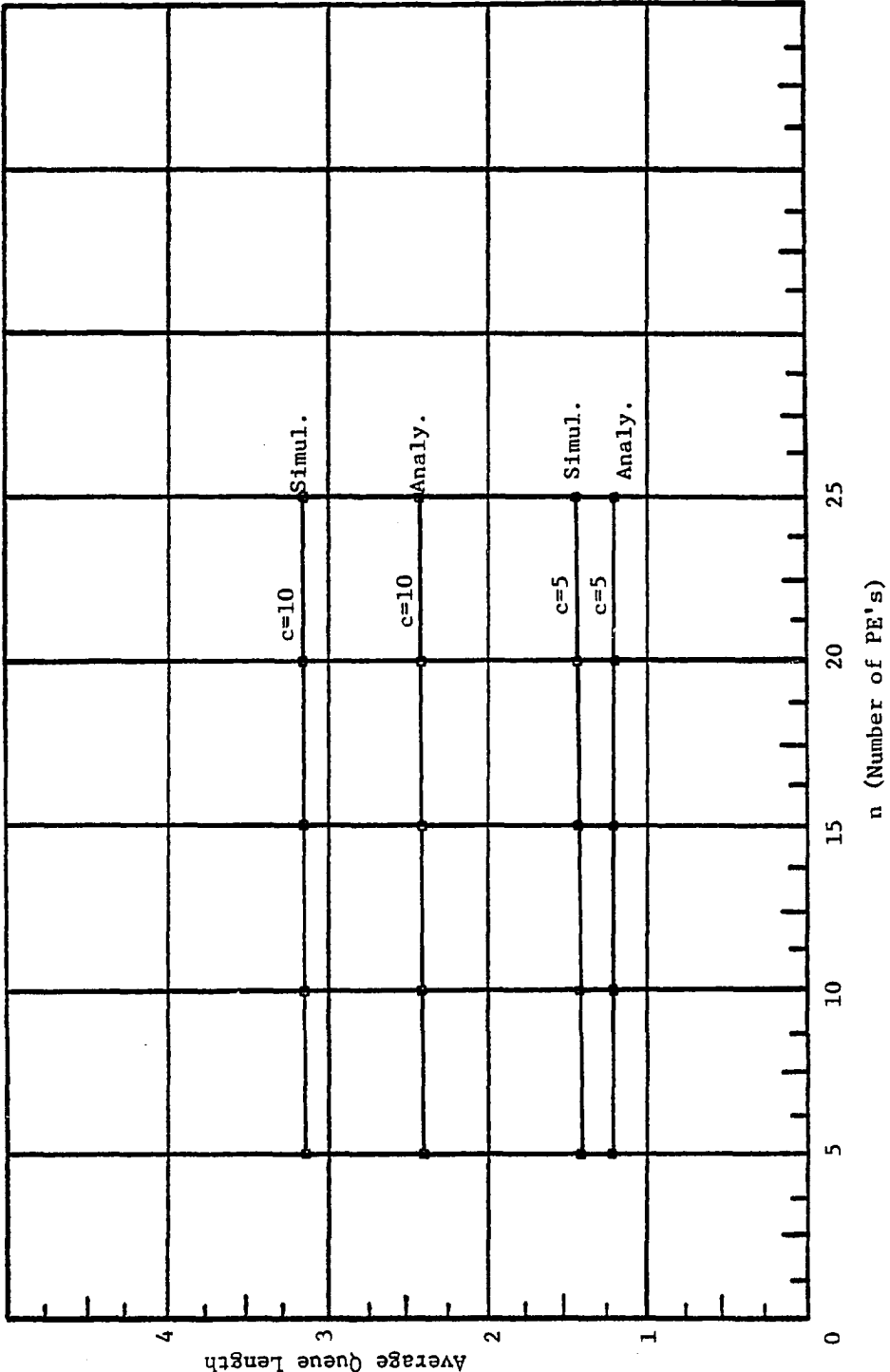


Figure 49: The Average Queue Length as a Function of n



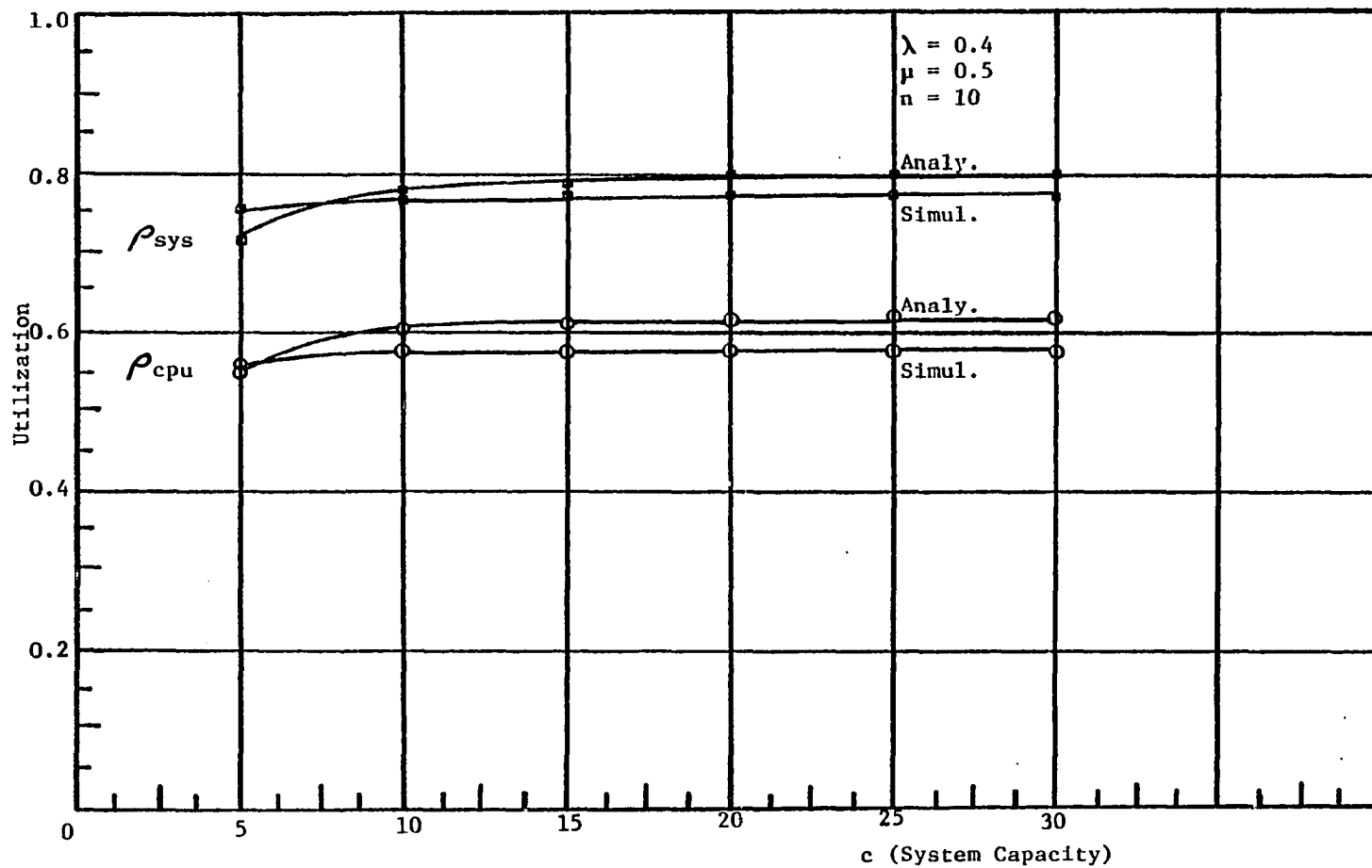


Figure 50: The Utilizations as a Function of  $c$

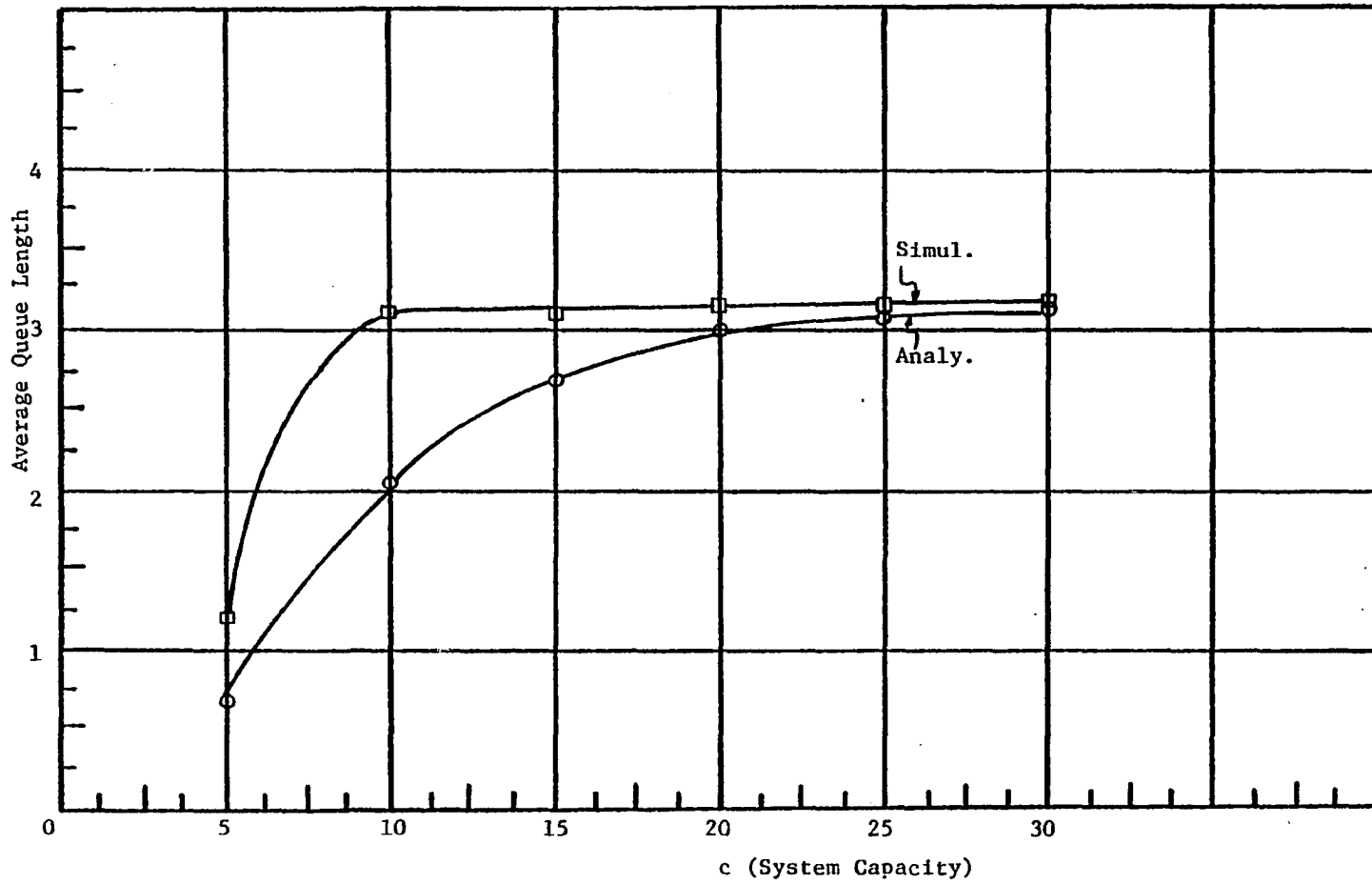


Figure 51: The Average Queue Length as a Function of c

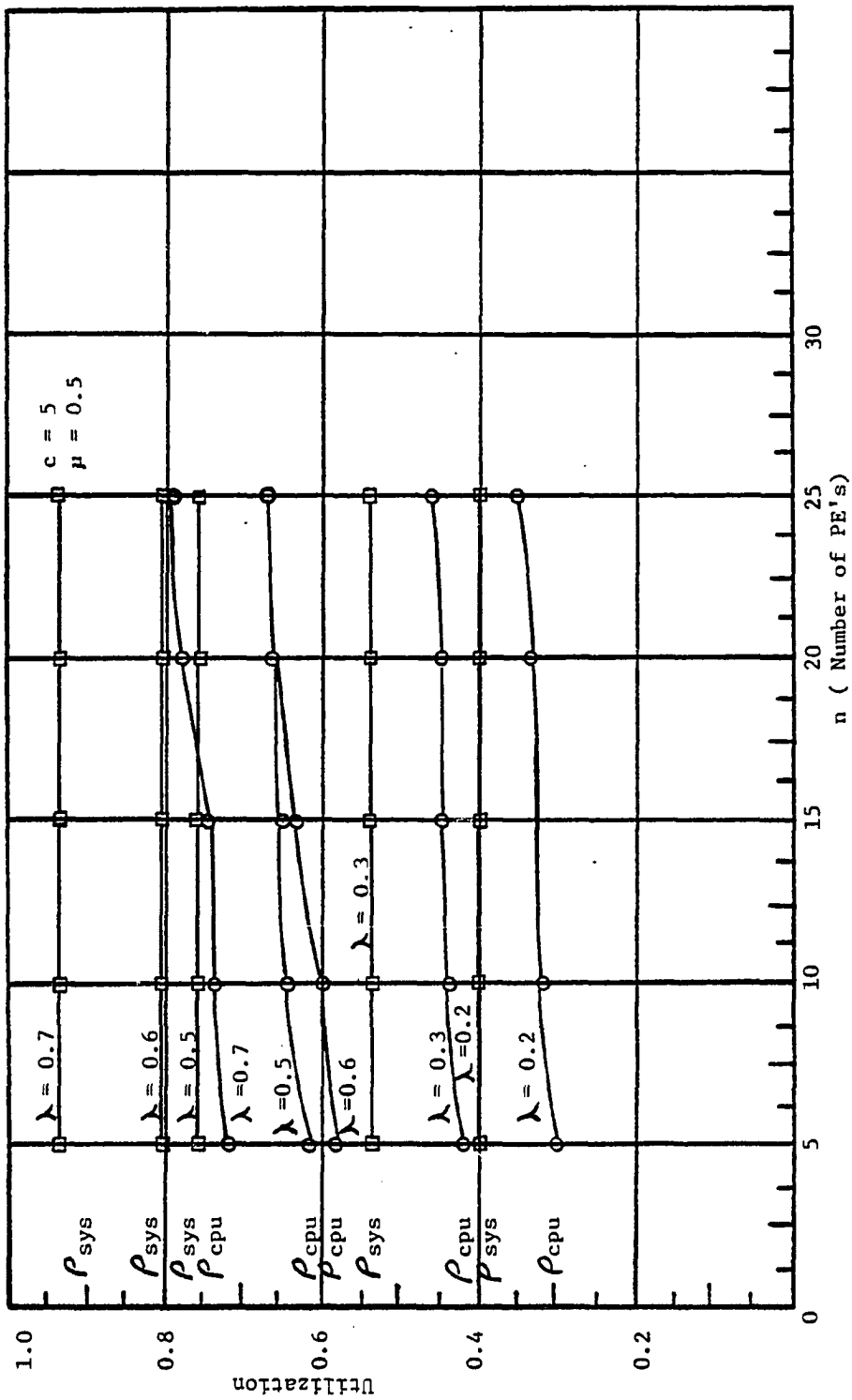
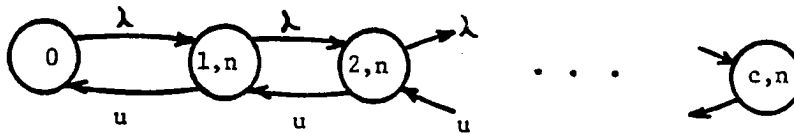


Figure 52: The Utilization as a Function of n and  $\lambda$

The state transition diagram of Figure 39 will reduce to a one which has only the last row, i.e.,



Therefore, it can easily be treated as a one-dimensional algebraic system. In Figure 53 the total system utilization is given as a function of system capacity for different numbers of PE. The utilization will greatly be affected by the service and arrival rates.

#### 6.6.2 Micro-Analysis of the Array Model

For the micro-analysis, only simulation experiments are performed. A fixed number of jobs are assumed to be in the system. Each job contains a fixed number of instructions. When a job is executed it uses the same number of PE's through out the execution period. This last restriction will ease the analysis somewhat. The time for macro-instruction fetch from the main memory to the IR is designated as the interarrival time. The service time of the control unit and the microinstruction access time is defined as the total controller's execution time. The PE's will have identical service time distribution since all are assumed to perform the same microinstruction.

In Figure 54 the system and the PE's utilization are plotted as a Function of  $\lambda$  for different values of  $c$ . The maximum PE utilization is reached when  $\mu$  approaches  $\lambda$ . The queue lengths are shown in Figure 55. For  $\lambda$  20% higher than  $\mu$ , the average queue length reaches the system capacity and remains unchanged thereafter.

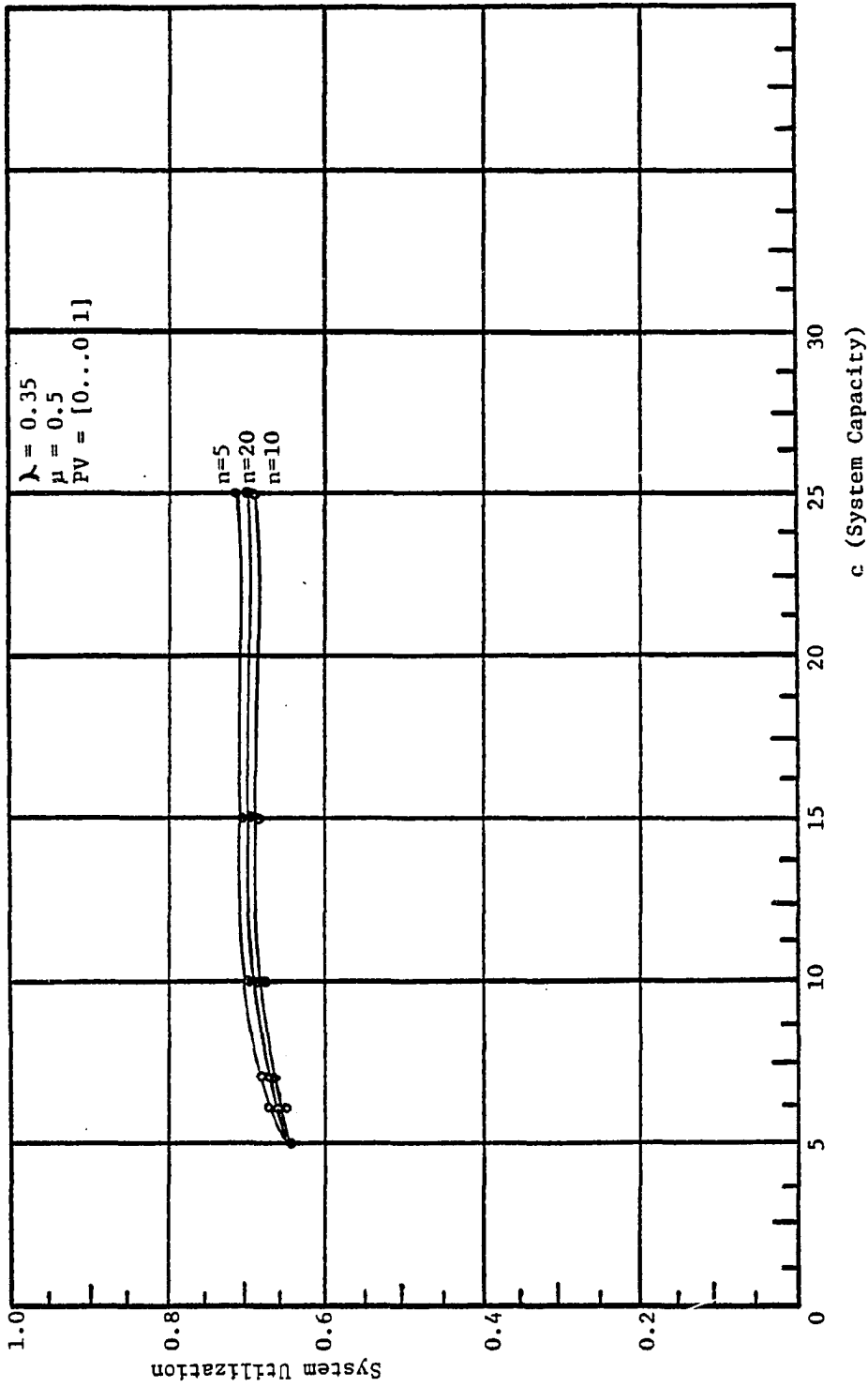


Figure 53: The Utilization as a Function of c

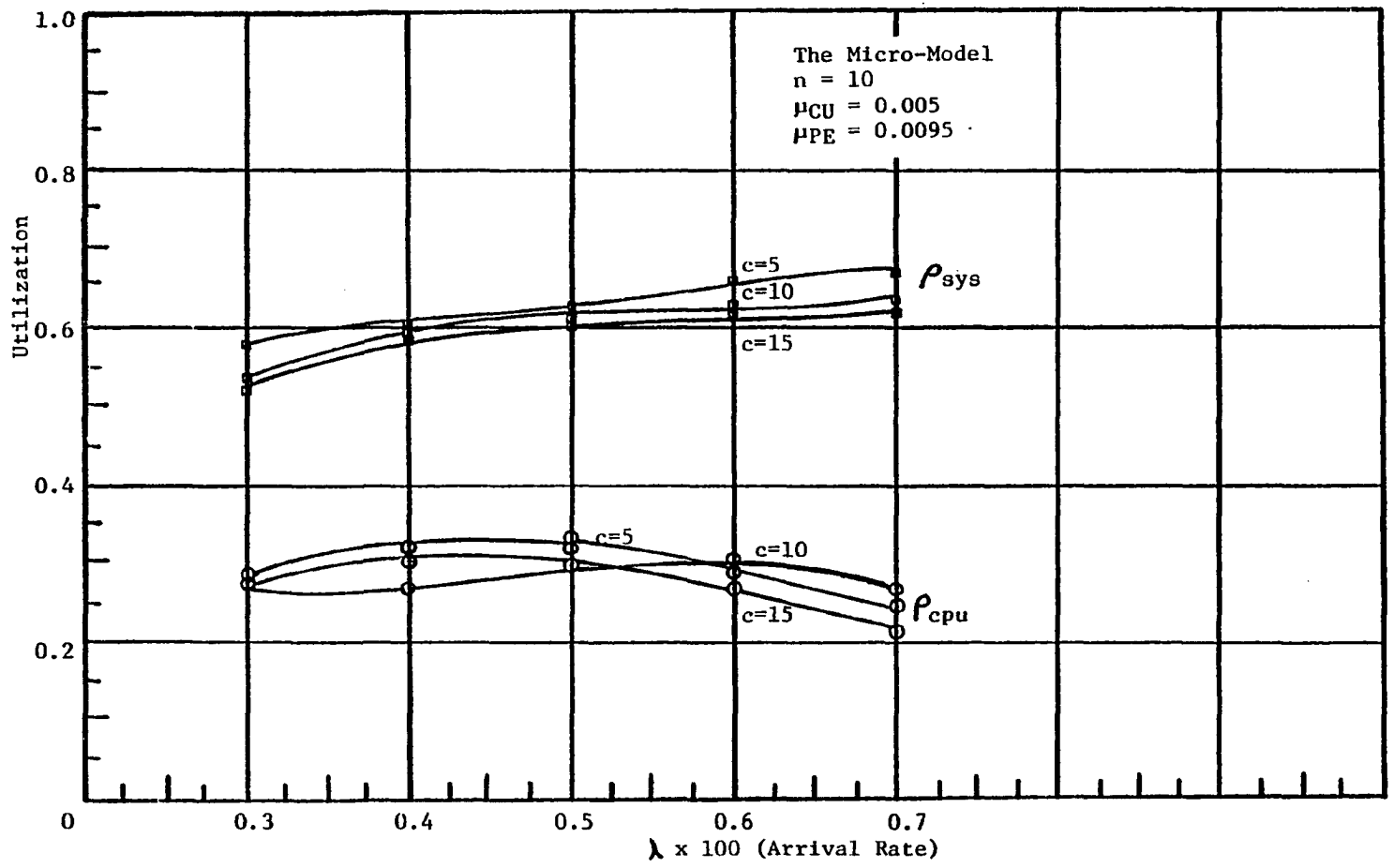


Figure 54: The Utilization as a Function of the Arrival Rate

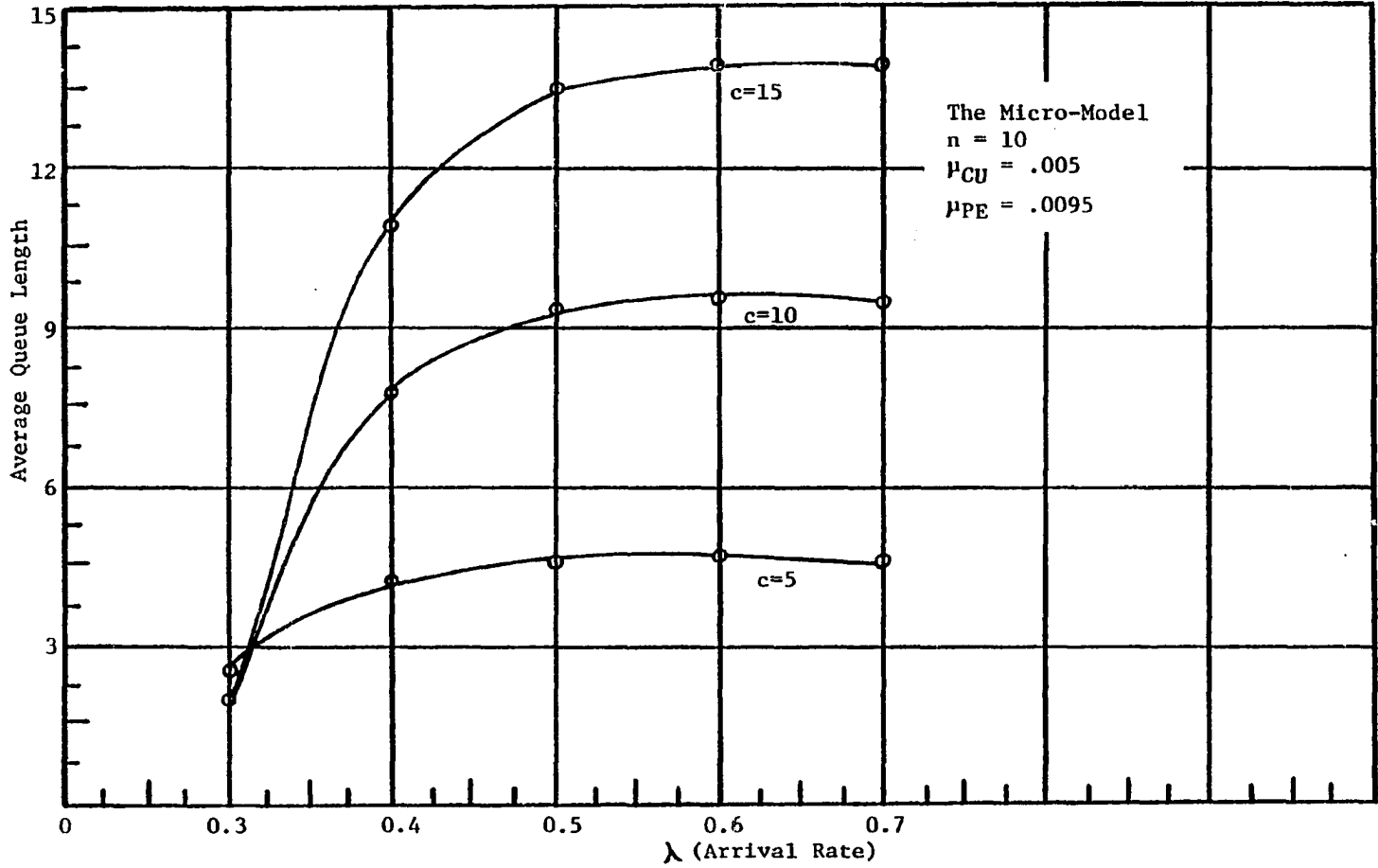


Figure 55: The Average Queue Length as a Function of  $\lambda$



## 6.7 REMARKS

The direct execution computer of chapter II can be referred to here in order to compare against our array organization. The similarity between these two systems organizations lies in the separation of data and control. In Chu's direct execution machine [CHU81] the control processor executes tokens which are part of the control flow, whereas the data processor executes tokens which are part of the data flow. In our design the lexical processor is absent. Therefore, all the program instructions to be executed reside in the main memory of the system (that of the MPE). However, the instruction set should have two distinguished groups of instructions. One set of macroinstructions is used for the SPE and another set for the MPE. The real burden is with the controller and the microcode. Each op code (for SPE and MPE macroinstructions) will have its corresponding microcode. The data-type instructions that are specified for the SPE units should enable the SPE and disable the MPE, whereas the control type instructions and data to be performed by the MPE should enable the MPE to receive the microcode and disable all the SPE's. One bit of microcode is designated for each PE (SPE and MPE).

## Chapter VII

### DATA FLOW MODEL

#### 7.1 DATA FLOW CONCEPTS

Data flow computers are based on the readiness of the operands involved in the computation. Unlike the conventional systems (e.g., von Neumann machines) data flow systems do not execute the program in a sequential fashion, but rather, an instruction is executed or prepared for execution when all of its operands are ready. For example, an Add x,y instruction will initially wait in the main memory for the operands x and y to be either defined or be supplied as a result of execution of other instructions. When the operands become available, then the instruction is sent to a free execution unit (PE).

Sequencing through the instruction of a given program is an attribute of the von Neumann machine. The data flow principle utilizes the flow graph method in order to invoke the parallelism inherent in a program. Thus only those programs that are suitable for parallel application can be implemented on a data flow machine. The theoretical groundwork for data flow computation can be traced back to 1966 [WATS79]. At MIT a team led by J. B. Dennis pioneered the

research for the realization of data flow machines [DENN74]. A number of systems that use the data flow principle have been developed and studied [AGER82].

## 7.2 DATA FLOW PROGRAM EXAMPLE

In order to illustrate the data flow program execution, we use the following simple example:

```
INPUT a,b,c,d
Begin
  x := ((a+b) - c )/( d * a )
End.
OUTPUT x
```

In order to execute this program, it should first be written in data flow graph form and entered into the memory. The memory entries consist of two main parts: instruction store entries and initial token entries. Let us derive the data flow graph for the above program. Developing the data flow graph is straightforward although it can be tedious in some instances. The graph is started with the initial tokens, i.e., the input variables, in this case a,b,c, and d. For each operation a box is defined. Adjacent to each box an address is indicated. To each box there is a number of links going in and out. In this example, two links in and one link out, as shown below in Figure 56.

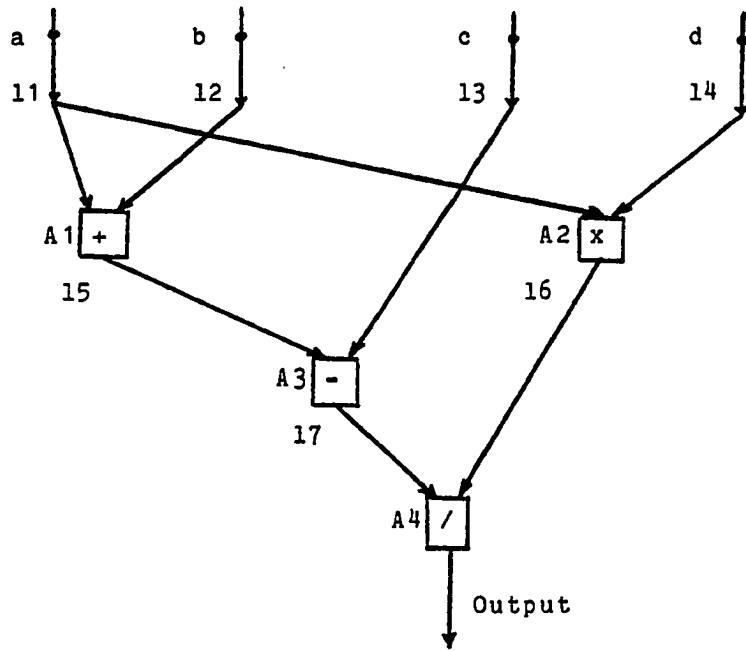


Figure 56: The Data Flow Graph

Instruction store entries:

Addr.	Operation	Next Instr. Addr A' LH/RH	Next Instr. Addr B' LH/RH
A1	ADD	A3 LH	-----
A2	MUL	A4 RH	-----
A3	SUB	A4 LH	-----
A4	DIV	Output	-----

Initial tokens entries:

Value	Lable	Next Instr. Addr LH/RH	Next Instr. Addr LH/RH
a	--	A1 LH	A2 LH
b	--	A1 RH	-----
c	--	A3 RH	-----
d	--	A2 RH	-----

Since the op code used requires two operands, therefore, at each address we should have two inputs. Some of the inputs are initial values of the tokens, while others are the results of execution of other instructions. In this simple example, we note that address A1 has both of its operands ready, i.e., A1 LH and A1 RH, where LH and RH stand for left-hand side and right-hand side, respectively. The instruction at address A1 will then be ready for execution. The same applies for the instruction at address A2. Executing the instruction at A1 will make the instruction at address A3 ready for execution since A3 has the RH operand ready, and A1 supplies the LH operand. By inspecting the data flow graph of Figure 56, we observe three levels of executions:

level 1: Operation at A1 and A2 are under execution

level 2: Operation at A3 is under execution

level 3: Operation at A4 is under execution.

Hence, if the system has  $n$  processing elements, only two of the  $n$  PE's will be used concurrently at any one point. For this particular simple program only two processing elements (or less) are required. However, there exist applications where hundreds or even more processing elements are needed concurrently. One such example is in digital signal analysis and vector computation.

### 7.3 THE BASIC HARDWARE UNITS FOR A DATA FLOW SYSTEM

A basic data flow system consists of several modules. Each module operates asynchronously and independently of the other modules in the system. Each module processes the incoming data to its port and then sends it to the next module in line. Figure 57 shows the overall interconnection of a typical data flow computer [DENN74].

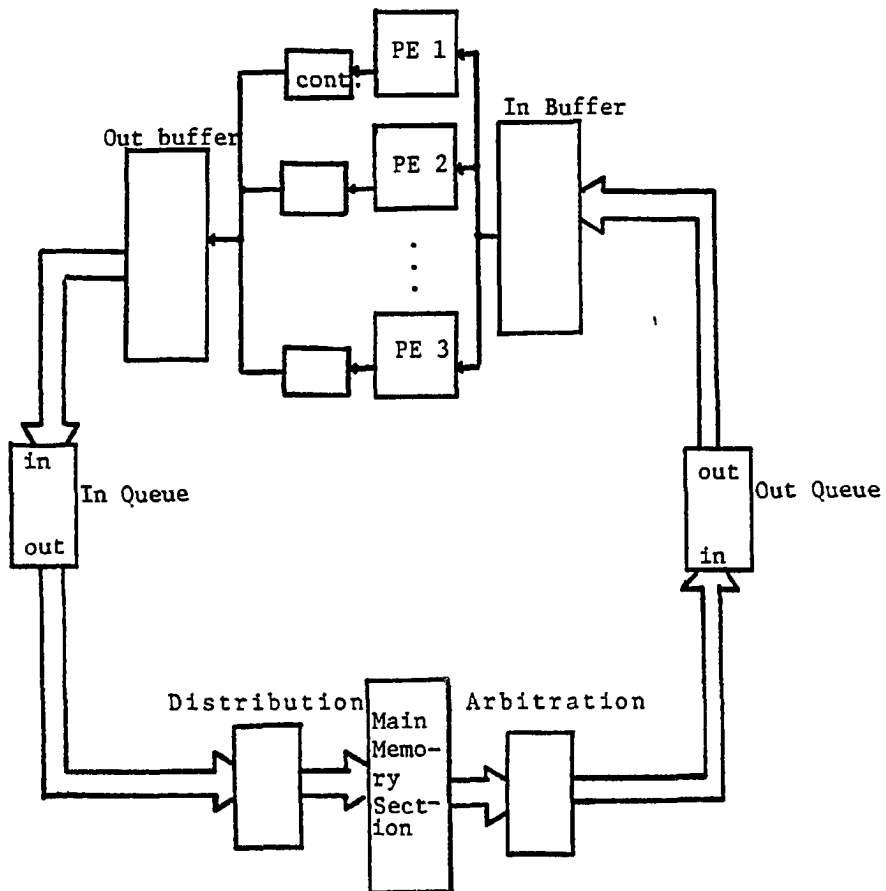


Figure 57: The Basic Blocks of a Data Flow System

In this section selected portions of the proposed data flow machine architecture will be examined in greater depth.

### 7.3.1 The PEQUE to PE Connection

The PEQUE is the queue that holds the ready instructions. In Figure 58 the connection from the PE to this queue is shown. In order to avoid the selection of more than one PE, a priority encoder AM2913 is implemented. For each additional eight PE's a new Am2913 is introduced. In Figure 58 we illustrate, for simplicity, a 3-PE system. The PE's generate a processor available signal to feed into the Am2913. Depending on the output of the encoder, only one set of (ai,bi) gates will be selected, which in turn enables the data path to the selected PE. The outputs of the (ai) gates are fed to an OR logic. The OR gate (C) generates a low to high signal whenever any PE is available, and provides the PD signal to the queue.

### 7.3.2 The Processing Element (PE)

Since the operation packet consists of the operation as well as the destination addresses, some bits in the byte are used for the control purposes. If the most significant three bits are used, eight possible signals can be generated. Table 4 defines these signals. The control signal 000 is used for the selection of the op code. When

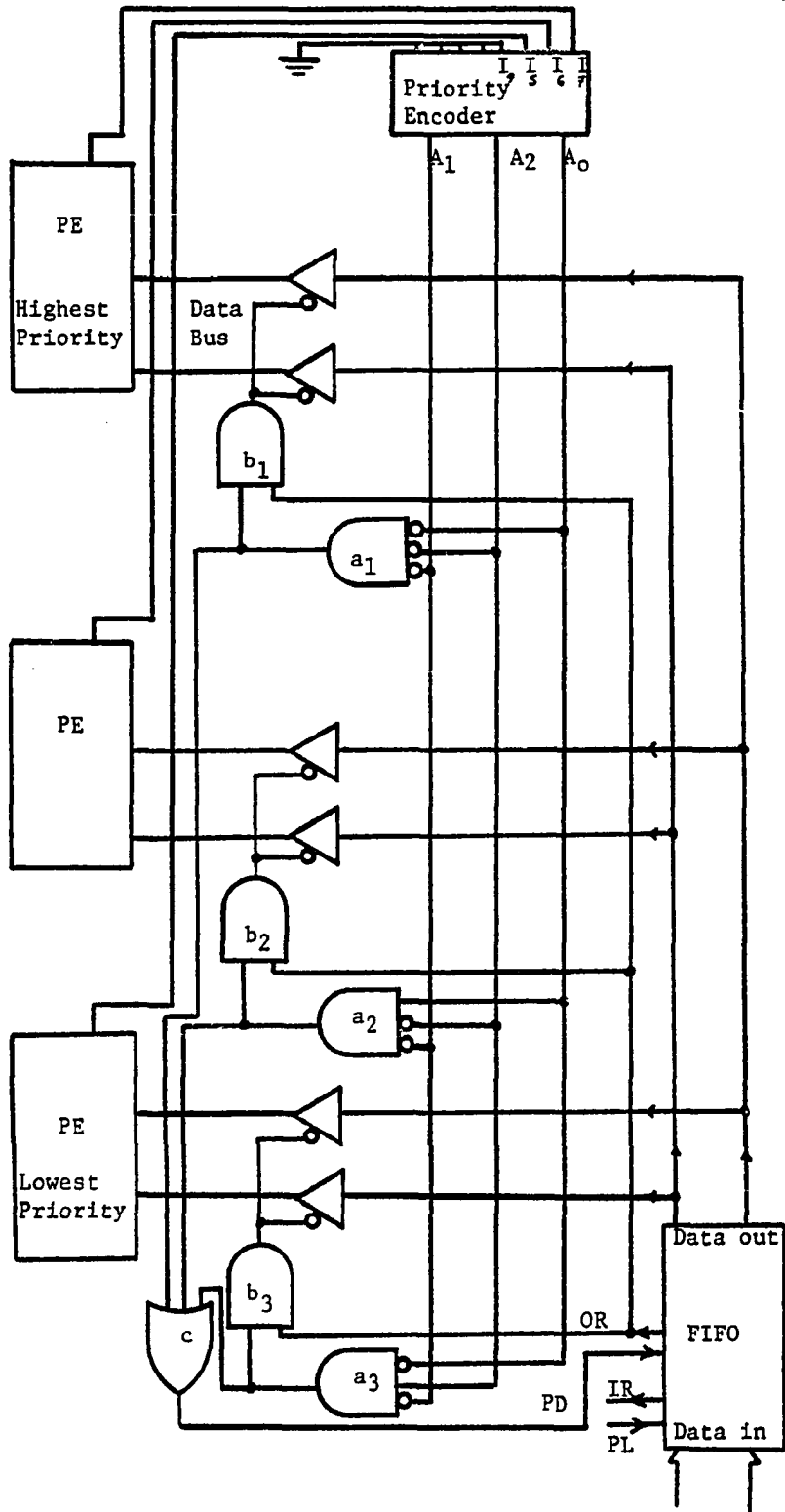


Figure 58: The PEQUE to PE connection



the byte (or the word) has its most significant 3 bits equal to 000, then the byte carries the op code portion of the packet.

TABLE 4

Definitions of the Signals	
Signal	Function
000	op code
001	opnd 1
010	opnd 2
011	dest 1
100	dest 2
.	.
111	last byte

Byte serial transmission is used. The control signal 111 indicates the end of the byte transmission. Figure 59 shows the distribution of the different signals in the PE circuit; note that each PE is a complete bit-slice microprocessor.

### 7.3.3 The Queueing Circuit

The queueing circuit is very straightforward. The logic should provide the following signals:

- PL: parallel load (in)
- IR: input ready (out)
- PD: parallel dump (in)

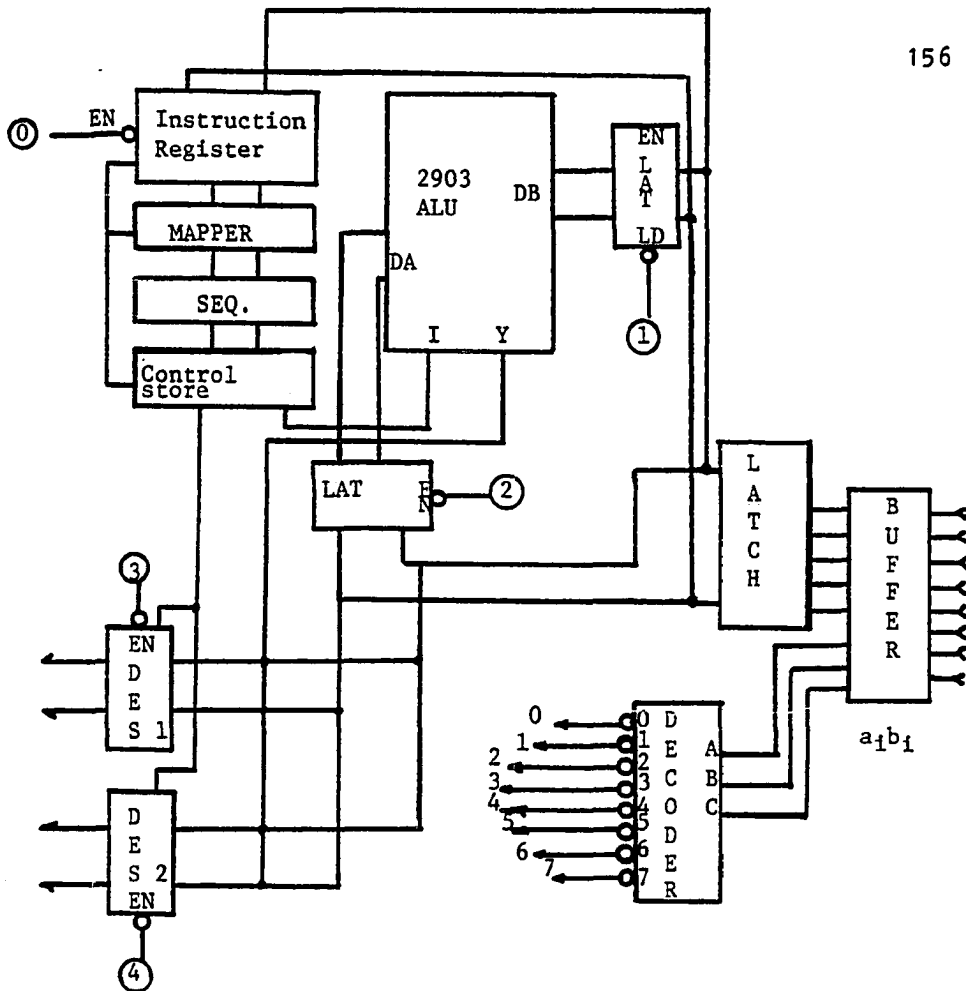


Figure 59: The PE Circuit Diagram

OR: output ready (out)

The input ready signal provides an enable input signal for the memory buffer and the memory control unit provides a parallel load (PL) signal to the queue. The other two signals (PD and OR) are for the communication between the PE and the queue. Whenever a PE is available, a PD signal is generated that is input to the queue. The OR and PE available signals will route the data to the specific PE.

The Am2812 FIFO chip can be used here. It is a 32-word x 8-bit FIFO and is expandable in both word and bit directions. The queueing circuit between the PE and memory is constructed in a similar fashion.

#### 7.3.4 The Memory Section

The memory section should contain the necessary logic for the selection of the ready instruction and the distribution of results from the previously executed instructions, thus forming the most complex part of the system. A complete bit slice microprocessor is used for this control. In fact, the memory section is considered the central control unit for the whole system for the instructions are initiated and maintained here. We will discuss the main activities that take place in the memory section.

The main memory blocks can basically be represented by the blocks of Figure 60. It is assumed that the memory contains instructions of a given program. Moreover, the program is assumed to contain some degree of parallelism.

The instructions and the tokens reserve two memory sections of the form:

A	op code	1st opnd	2nd opnd	Next Inst. address 1	Next Inst. address 2	opnd cntr
B	Value	Inst. address 1	Inst. address 2			

The following steps take place:

1. First start by pointing to address of the 1st opnd in A and latch it to a register.



tokens are only defined in the definition phase.

4. After performing step 2 and (or) step 6, whenever the operand counter is equal to 0, the instruction should be passed to the PE queue (PEQUE) by enabling the buffer.

The path from the PE to the memory is done in the following two steps;

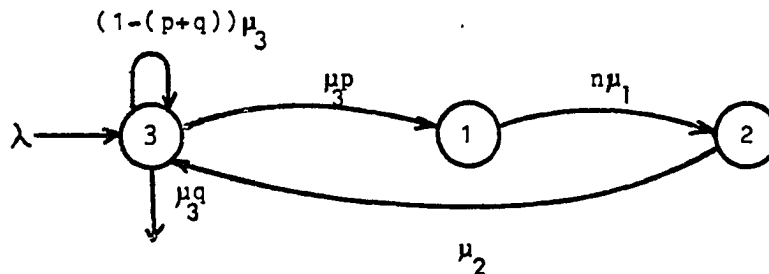
5. The result is passed to the distribution network queue. When a resultant packet arrives (which consists of two parts: result and address), the result is placed in the instruction cell whose operand address (1st or 2nd) agrees with the destination address in the result packet.
6. The control unit in the memory should deal with the incoming packet and place the value in the proper address. Repeat step 4 if necessary.

#### 7.4 THE DATA FLOW MODEL

In this section we will apply two analysis techniques to the data flow machine. We start by configuring the queueing elements involved in the model construction. The illustration shown in Figure 61 basically represents the desired model. To simplify the discussion for the moment, the model is divided into three different subsections, namely 1,2, and 3 as indicated by Figure 61. By inspecting the model of Figure 61, one can realize the complexity

involved in analyzing the system analytically. For this kind of queueing networks, however, one usually pursues simulation rather than analytic techniques. Furthermore, by treating the whole model as a number of queues, one then can apply standard queueing techniques to each section independently of the other sections, as pointed out in chapter IV.

The network is represented in a nodal form by the following:



and each node can be treated independently. The nodes used here are of mixed types. Node 1 is an (M/M/n) system, node 2 is an (M/M/1) system, and finally node 3 is an (M/M/1) system with feedback. Of all the three independent sections, we will only derive the steady state (balance equations) for section 3. The results for sections one and two can be obtained without much difficulty [KLEI75a], [WHIT75].



- $u_3$  : The system service rate (macroinstructions/unit time)
- $p$  : The probability that the instructions are ready and will be fed to the queue of stage 1.
- $q$  : The probability that the instruction will be output.

As a consequence,  $(1-(p+q))$  is the probability of remaining in the system. The above probabilities can easily be estimated from a real data flow program. The state transition diagram is represented by Figure 62.

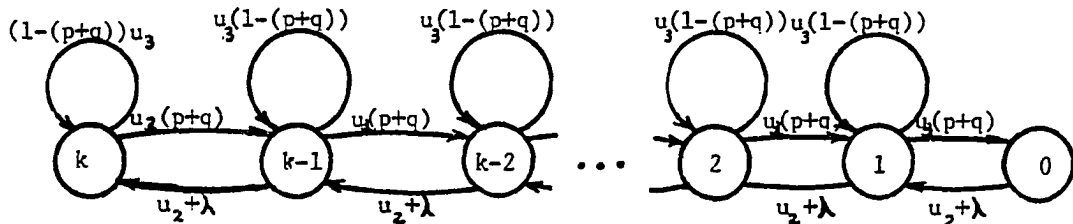


Figure 62: The State Transition Diagram

Let  $1-(p+q) = z$  ;  $(p+q) = x$  ;  $u_2 + \lambda = \psi$  ; and  $\beta = \gamma/u_3 x$

$$P_0 \psi = P_1 \mu_3 x \quad \longrightarrow \quad P_1 = P_0 (\psi / \mu_3 x) = P_0$$

$$P_1 (\psi + \mu_3 x) = P_0 \psi + P_2 \mu_3 x$$

$$P_2 = \frac{P_0 \psi ((\psi + \mu_3 x) / (\mu_3 x - 1))}{\mu_3 x}$$



$$\therefore P_2 = P_0 (\psi / \mu_3 x)^2 = P_0 \beta^2$$

$$\text{and } P_3 = P_0 \beta^3$$

In general  $P_n = P_0 \beta^n = P_0 \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)^n$  7-1

In order to compute  $P_0$  (the probability the system being empty) the normalizing relation is applied and

$$\sum_{n=0}^K P_0 \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)^n = 1$$

$$1 = P_0 \left[ \frac{1 - \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)^{K+1}}{1 - \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)} \right]$$

$$\therefore P_0 = \left[ \frac{1 - \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)}{1 - \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)^{K+1}} \right] \quad 7-2$$

Now, substituting the final expression for  $P_0$  in 7-1 we obtain the the following general state probability

$$P_n = \frac{1 - \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)}{1 - \left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)^{K+1}} \left[ \frac{\left( \frac{\mu_2 + \lambda}{\mu_3 (p+q)} \right)^n}{\mu_3 (p+q)} \right] \quad 7-3$$

The parameter  $u$  in Eq 7-3 signifies the fact that the state probability does indeed depend on the service rate of stage 2. Stage 1 is an example of  $(M/M/m):(FCFS/k/\infty)$  and the

solution in general is given by

$$P_k = P_0 \frac{\lambda^k}{k! \mu^k} \quad 0 \leq k \leq m-1 \quad 7-4a$$

$$P_k = P_0 (\lambda / \mu)^k \cdot ((m)^{m-k} / m!) \quad m \leq k \leq K \quad 7-5a$$

Next  $P_0$  is calculated

$$\begin{aligned} \sum_{k=0}^K P_k &= 1 \\ \sum_{k=0}^{m-1} P_k + \sum_{k=m}^K P_k &= 1 \\ \sum_{k=0}^{m-1} P_0 \frac{\lambda^k}{k! \mu^k} + \sum_{k=m}^K P_0 (\frac{\lambda}{\mu})^k \frac{(m)^{m-k}}{m!} &= 1 \\ \text{let } k-m &= z \\ \therefore \sum_{k=0}^{m-1} P_0 (\frac{\lambda}{\mu})^k \frac{1}{k!} + \sum_{z=0}^{K-m} P_0 (\frac{\lambda}{\mu})^{z+m} \frac{(m)^{m-(z+m)}}{m!} &= 1 \\ P_0 \left[ \sum_{k=0}^{m-1} (\frac{\lambda}{\mu})^k \frac{1}{k!} + \sum_{z=0}^{K-m} (\frac{\lambda}{\mu})^{z+m} \frac{(m)^{-z}}{m!} \right] &= 1 \\ \text{let } (\frac{\lambda}{\mu}) &= \rho \\ \therefore P_0 \left[ \sum_{k=0}^{m-1} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \sum_{z=0}^{K-m} \frac{\rho^z}{m^z} \right] &= 1 \\ P_0 \left\{ \sum_{k=0}^{m-1} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \left[ \frac{1 - (\frac{\rho}{m})^{K-m+1}}{1 - (\frac{\rho}{m})} \right] \right\} &= 1 \\ \therefore P_0 &= \left\{ \sum_{k=0}^{m-1} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \left[ \frac{1 - (\frac{\rho}{m})^{K-m+1}}{1 - (\frac{\rho}{m})} \right] \right\}^{-1} \end{aligned}$$

Therefore, equations 7-4a and 7-5a become,

$$P_0 = \left\{ \sum_{k=0}^{m-1} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \left[ \frac{1 - (\frac{\rho}{m})^{K-m+1}}{1 - (\frac{\rho}{m})} \right] \right\} \left( \frac{\lambda}{\mu} \right)^k \frac{1}{k!} \quad 7-4b$$

$0 \leq k \leq m-1$

$$P_k = \left\{ \sum_{k=0}^{m-1} \frac{\rho^k}{k!} + \frac{\rho^m}{m!} \left[ \frac{1 - (\frac{\rho}{m})^{K-m+1}}{1 - (\frac{\rho}{m})} \right] \right\} \left( \frac{\lambda}{\mu} \right)^k \frac{(m)^{m-k}}{m!} \quad 7-5b$$

where  $m \leq k \leq K$

Stage 2 is the classic (M/M/1):(FCFS/1/∞) case, the solution is given by

$$P_k = \begin{cases} \left[ \frac{1 - (\frac{\lambda}{\mu})}{1 - (\frac{\lambda}{\mu})^{L+1}} \right] \cdot \left( \frac{\lambda}{\mu} \right)^k & 0 \leq k \leq L \\ 0 & \text{otherwise} \end{cases} \quad 7-6$$

In subsystem 1, the queue holds all the instructions that are ready for execution. As soon as a PE becomes available, the instruction at the head of the queue will seize it. The overall average service time is considered as the average service time of all the PE's.

In subsystem 2, the queue will hold only the resultant package. The result of executing an instruction in the PE will be tagged to each of the destination addresses in the execution packet. For example, if the execution packet is of the form :

$$\left[ \text{op code, opnd1, opnd2, dest1, dest2, dest3} \right]$$

then the resultant package produced is of the form :

<Result , Dest1>  
 <Result , Dest2>  
 and <Result , Dest3>

These three packets will be in the queue in this order. This subsystem will examine each resultant packet and place the result in the memory at the address specified by the destination part of the packet. The queue is of the FCFS type and only one instruction is serviced at a time. The capacity of this queue is assumed to be 1. The capacity can be calculated by

$$1 = \text{Number of PE's} \times (\text{Average number of destination addresses in each instruction})$$

The third subsystem represents the model of the central memory of the system. The memory can be modeled as one big queue whose size is M (the size of the memory).

The utilization of the PE's equal to the expected number of the PE's in the busy state to the total number of PE's in the system.

$$\rho_{PE} = \frac{1}{n} E [ PE_B ] \quad 7-7$$

The number of instructions in the system can be defined according to the following:

$$N_{sys}(t) = N_{qs}(t) + N_E(t) + N_R(t)$$

Let  $P_{ihj}(t)$  designate the probability that the system is in state  $E_{ihj}$ ,

$$\text{where } 0 < i < (K+1+n) \cong K \text{ for } K \gg n$$

$$0 < j < N$$

$$0 < h < (K-n) \cong K \text{ for } K \gg n$$

From Eq 7-7, the  $E [ PE ]$  is given by:

$$\begin{aligned}
E [PE_B] = & (1.P_{oo1} + 2.P_{oo2} + \dots + n.P_{oon}) + \\
& (1.P_{o11} + 2.P_{o12} + \dots + n.P_{o1n}) + \dots + \\
& (1.P_{ok1} + 2.P_{ok2} + \dots + n.P_{okn}) + \dots + \\
& (1.P_{\ell o1} + 2.P_{\ell o2} + \dots + n.P_{\ell on}) + \dots + \\
& (1.P_{\ell k1} + 2.P_{\ell k2} + \dots + n.P_{\ell kn})
\end{aligned}$$

$$E [PE_B] = \sum_{i=0}^k \sum_{h=0}^k \sum_{j=1}^n j.P_{ihj}$$

Therefore, the utilization of the PE's is found

$$\rho_{PE} = \frac{1}{n} \left[ \sum_{i=0}^k \sum_{h=0}^k \sum_{j=1}^n j.P_{ihj} \right] \quad 7-8$$

Equation 7-8 can be reduced even further. We lump  $N_{qs}$  and  $N_R$  into one parameter ( $N_w$ ) to represent all the instructions in the system that are not under execution. This later modification will reduce the complexity of Eq 7-8 especially when a large memory is employed.

Therefore,  $N_w(t) = N_q(t) + N_R(t)$

In fact, the PE section sees  $N_{qs}$  and  $N_R$  indistinguishable. Also note that the transfer rate from the memory to the queue is not significant. Now, the system is considered to be at state  $E_{rj}$  at time  $t$ , when

$$N_{qs}(t) + N_R(t) = N_w(t) = r \quad \text{and} \quad N_E(t) = j$$

where  $0 < r < 2K$  and  $0 < j < n$ .  $P_{rj}(t)$  is the probability that the system is in state  $E_{rj}$ . When working in steady state conditions, the following are true,

$$\lim_{t \rightarrow \infty} N_{qs}(t) = N_{qs}$$

$$\lim_{t \rightarrow \infty} N_R(t) = N_R$$

$$\text{and} \quad \lim_{t \rightarrow \infty} N_w(t) = N_w$$

Therefore,  $N_w = N_{qs} + N_R$ , and equation 7-7 becomes,

$$\rho_{PE} = \frac{1}{n} E[PE_B] = \frac{1}{n} \left[ \sum_{r=0}^{2k} \sum_{j=1}^n j \cdot P_{rj} \right] \quad 7-9$$

Observe the reduction in the number of terms in equations 7-9 and 7-8, there are  $2K \times n$  terms in Eq 7-9, and  $k \times K \times n$  terms in Eq 7-8. That is, Eq 7-9 reduces the number of terms by a factor of  $(K/2)$  which is significant for large memories.

The average number of instructions waiting in the queues equal to :

$$N_q = [1 \cdot P_{11} + 1 \cdot P_{12} + \dots + 1 \cdot P_{1n}] + [2 \cdot P_{21} + 2 \cdot P_{22} + \dots + 2 \cdot P_{2n}] + \dots + [2k \cdot P_{2k,1} + 2k \cdot P_{2k,2} + \dots + 2k \cdot P_{2k,n}]$$

$$N_q = \sum_{r=1}^{2k} \sum_{j=1}^n r \cdot P_{rj} \quad 7-10$$

Due to the fact that the PE's can not be idle while an instruction is ready and waiting, then the only condition under which an instruction is truly waiting are  $i \cdot P_{i,n}$  for  $1 \leq i \leq 2K$ , and as a result Eq 7-10 becomes

$$N_q = \sum_{i=1}^{2k} i \cdot P_{i,n} \quad 7-11$$

The APL program for the analytic case is shown in appendix (F).

#### 7.4.2 The Simulation Analysis

Two simulation analysis will be performed,

**Case 1:** The general instruction execution type. The instructions are treated as a whole and the decisions are based on probabilities rather than the exact arrival of operands.

**Case 2:** The more detailed type. A typical data flow

execution is simulated. In this model each instruction cell (i.e., op code, operands, destination addresses) is completely specified.

#### 7.4.2.1 Case 1

Case 1 should support the analytic study of section 7.4.1. The simulation flow chart for this case is shown in Figure 63. Due to the GPSS limitation, a maximum of 200 transactions (instructions) can be present in the system at any given time. The case 1 simulation program is given in appendix G. The results of simulation and analytic solutions are presented in section 7.5 .

#### 7.4.2.2 Case 2

The second simulation study demonstrates the exact execution of a typical data flow program. It shows the power of GPSS language for these kinds of simulation. Each instruction cell is represented by a number of parameters. The program to be simulated is shown in Figure 64. Each box represents a specific operation. To be general, we will not specify the kind of operation and will represent the service rate of each PE with an exponential service rate. Parameter 4 is used to store the service time of each PE. If desired, the specific service time can be supplied. Figure 65 represents the flowchart of the second case. The memory

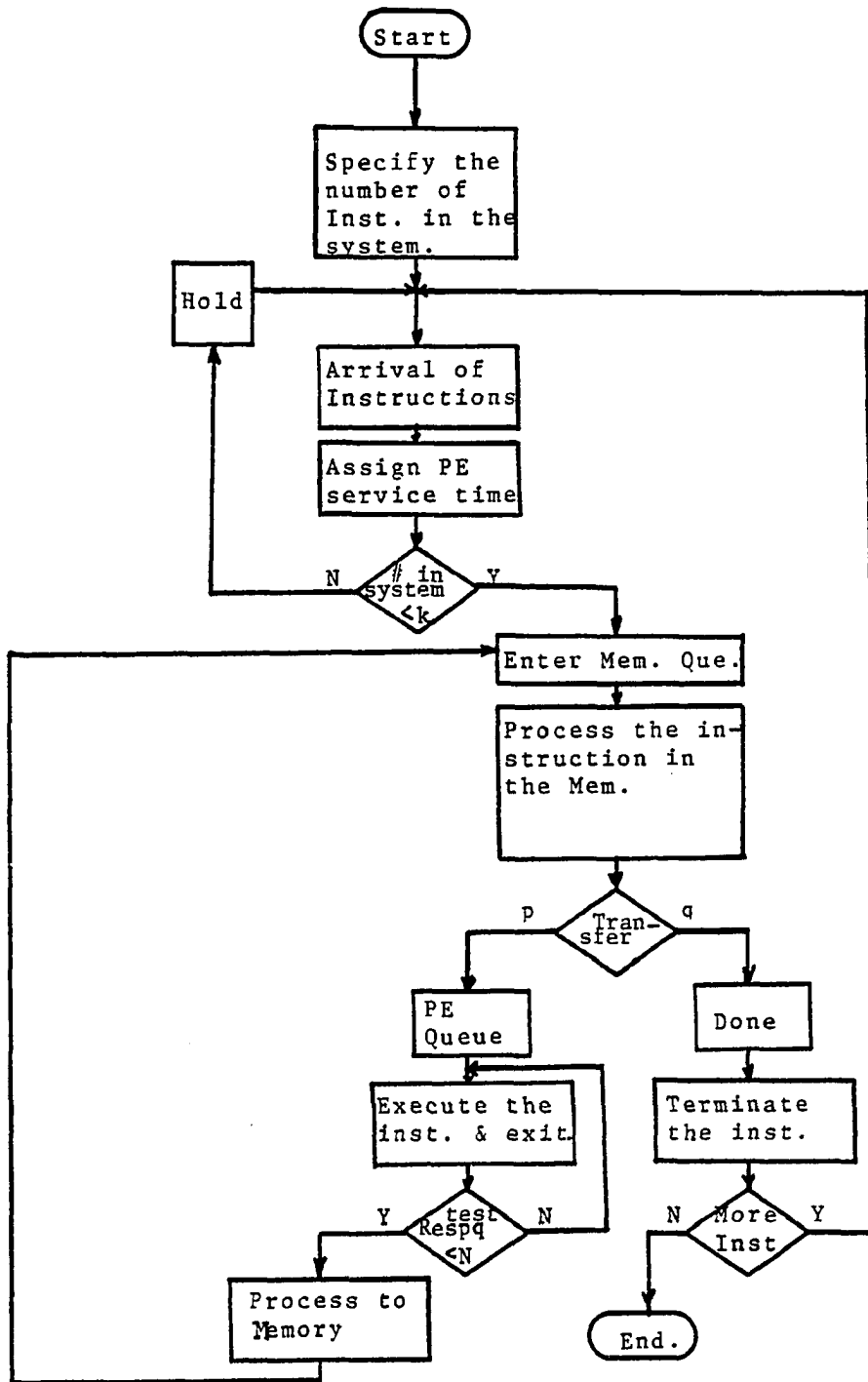


Figure 63: The Simulation Flow Chart for Case 1



cells that correspond to the program shown in Figure 64 are given in Figure 66. The numbers adjacent to each cell give the identity of the cell. The parameter definitions are given in the program listing in appendix G. With a little modification, the program can be used to simulate vector-oriented problems.

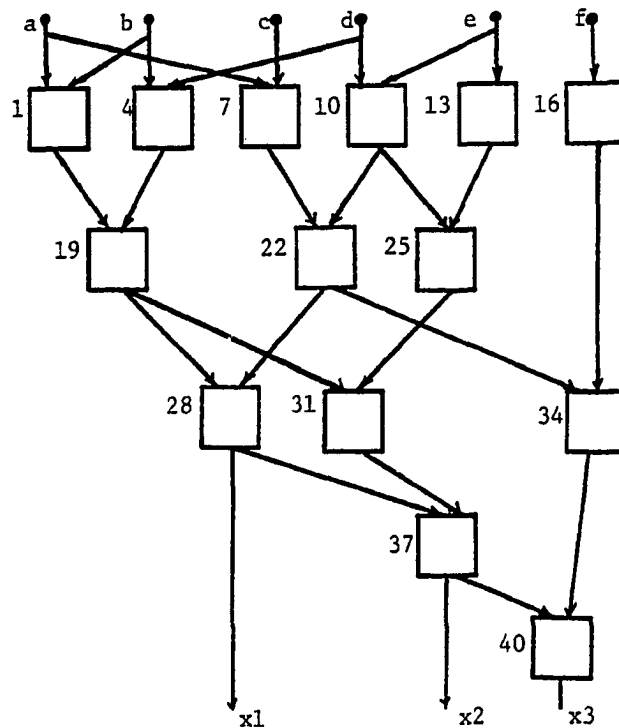


Figure 64: The Example Program to be Simulated

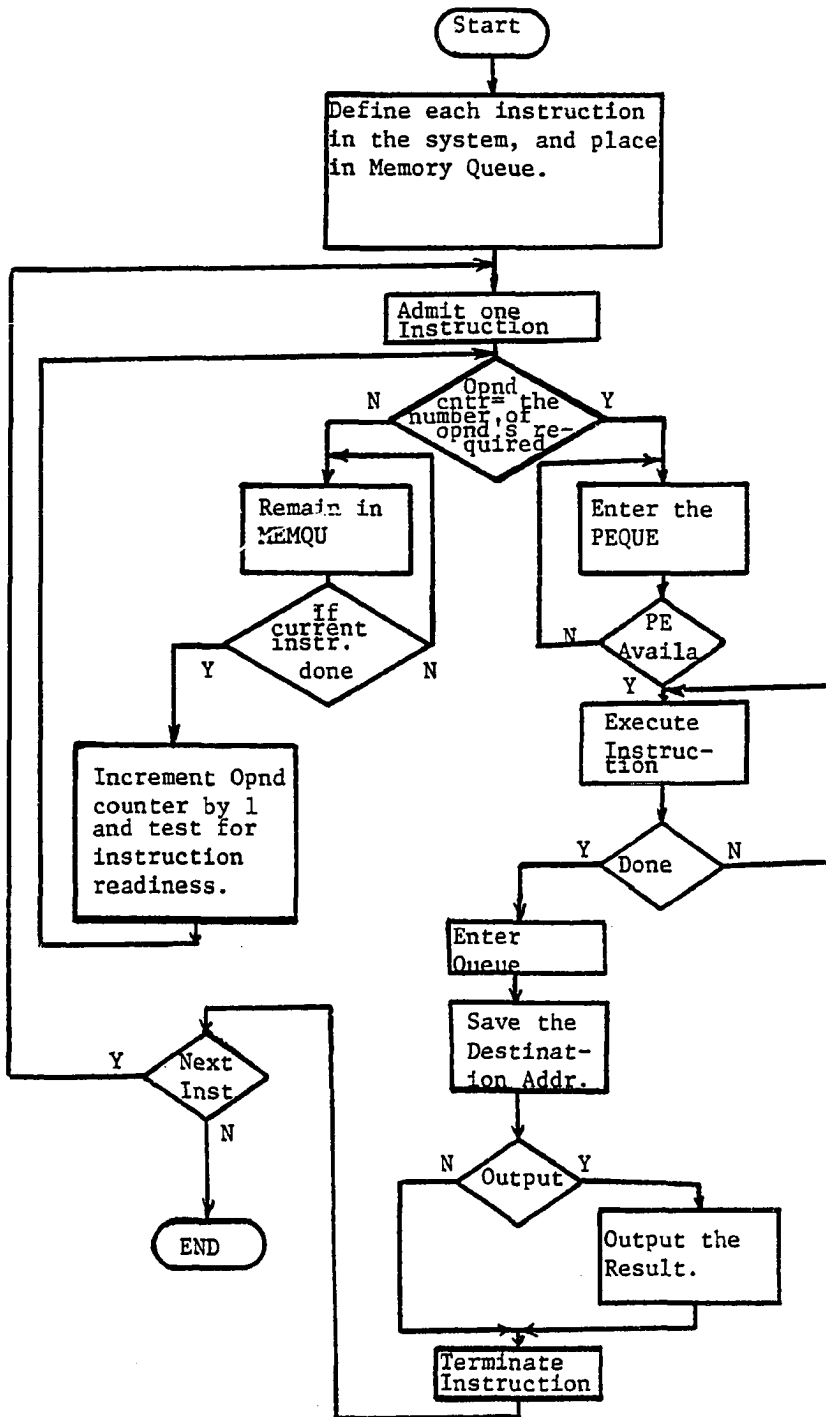


Figure 65: The Simulation Flowchart for Case 2

P1	P2		P4	P5	P6	P8	P9	P11	P12	P13	P16
1	1	1	-	2	2	2	3	20	-	-	-
		2									
		3									
4	2	4	-	2	2	5	6	21	-	-	-
		5									
		6									
7	3	7	-	2	2	8	9	23	-	-	-
		8									
		9									
10	4	10	-	2	2	11	12	24	26	-	-
		11									
		12									
13	5	13	-	2	2	14	15	27	-	-	-
		14									
		15									
16	6	16	-	1	1	17	18	36	-	-	-
		17									
		18									
19	7	19	-	2	0	20	21	29	32	-	-
		20									
		21									
22	8	22	-	2	0	23	24	30	35	-	-
		23									
		24									
25	9	25	-	2	0	26	27	33	-	-	-
		26									
		27									
28	10	28	-	2	0	29	30	38	-	-	50
		29									
		30									
31	11	31	-	2	0	32	33	39	-	-	-
		32									
		33									
34	12	34	-	2	0	35	36	42	-	-	-
		35									
		36									
37	13	37	-	2	0	38	39	41	-	-	51
		38									
		39									
40	14	40	-	2	0	41	42	-	-	-	52
		41									
		42									

Figure 66: The Memory Cells for the Program of Figure 64

## 7.5 ANALYSIS OF RESULTS

Case 1: As mentioned earlier, both analytic and simulation techniques are used in this case. The operand counter of each instruction is not specified explicitly, but rather probabilistic assumptions are employed. By specifying the probabilities  $p$ , and  $q$ , the readiness, or not readiness, and the completion of the instruction can be decided. Three possible values for  $p$  and  $q$  are tested,  $p, q$ ,  $p=q$ , and  $p, q$ . The corresponding results obtained for the PE utilization as a function of  $n$  are plotted in Figure 67. It is expected for the case when  $p$  is greater than  $q$ , that the PE utilization will be higher than for the other two cases. Furthermore, the corresponding throughput for the above cases are shown in Figure 68. The throughput is divided into two parts, that of the PE's and that of the whole system. The average contents of the different queues are obtained and plotted in Figure 69. Note that the queue with the greatest content is the memory queue. The average PEQUE content is seen to be zero for this particular example. In general it can be greater than zero.

Case 2: The results for this case are those of the simulation study only. As shown in Figure 67, the addition of new PE's to the system will reduce the overall utilization of the PE's and increase the overall system throughput. Figure 70 supports this fact.

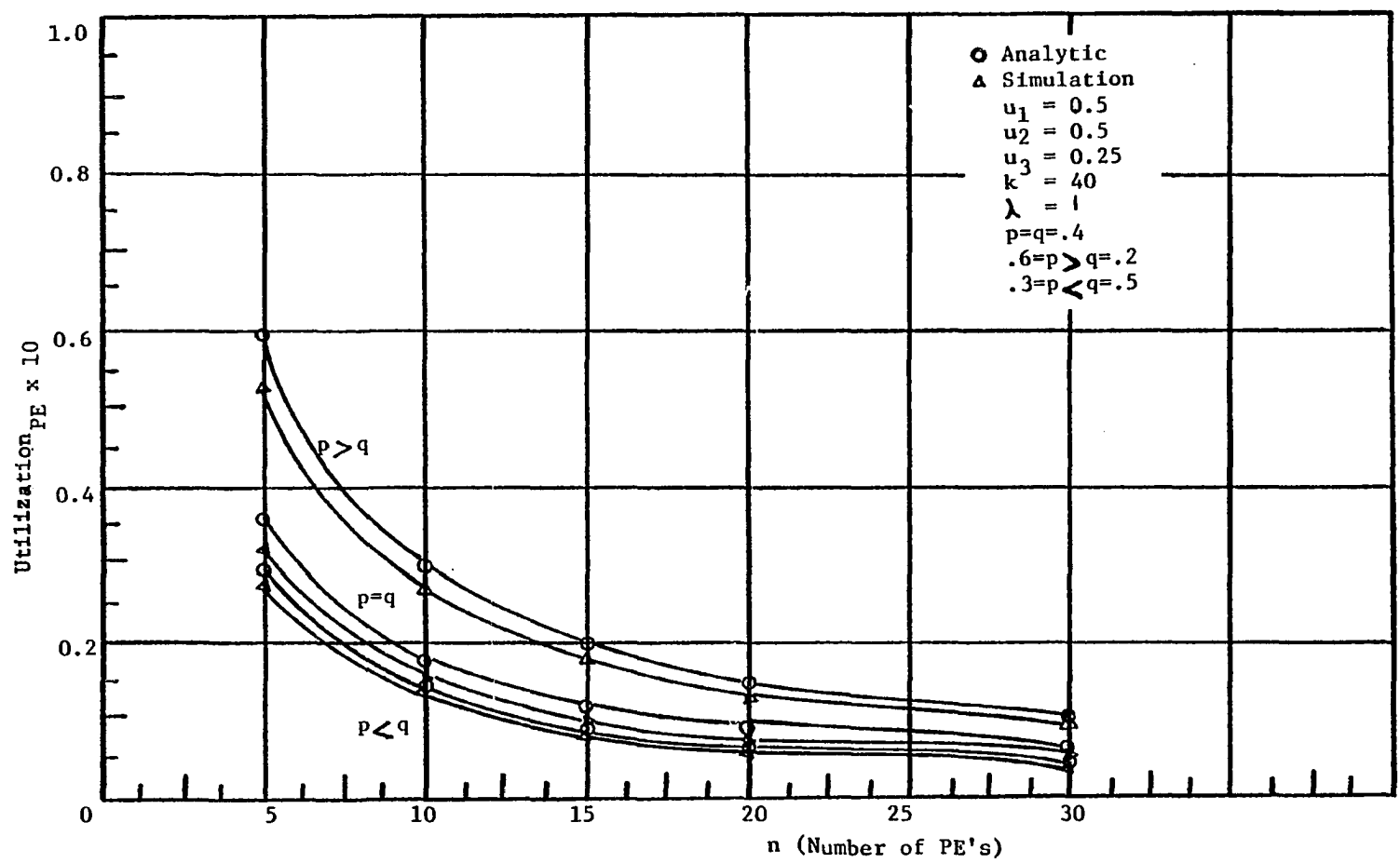


Figure 67: The Processing Elements Utilization as a Function of n

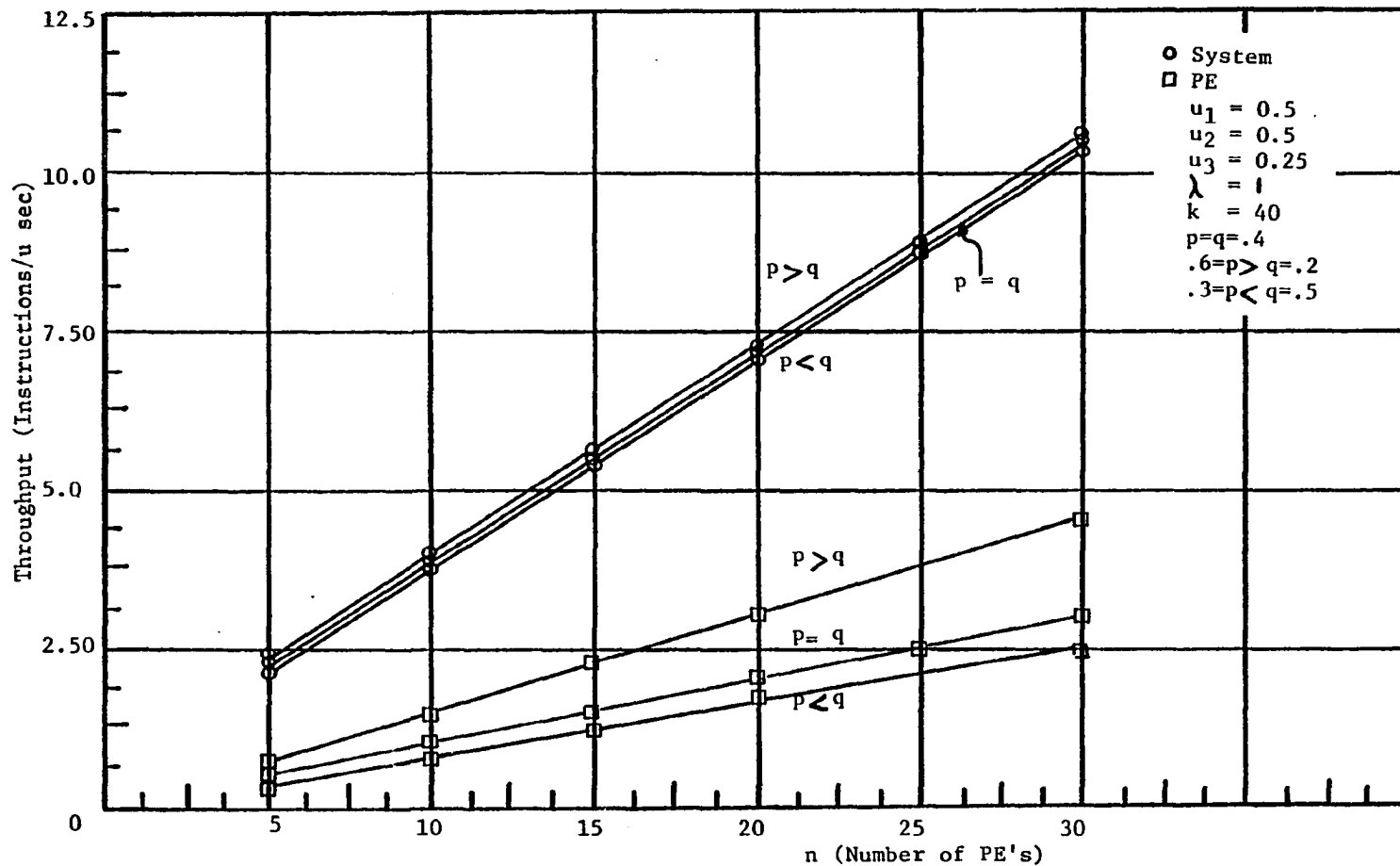


Figure 68: The Throughputs as a Function of n

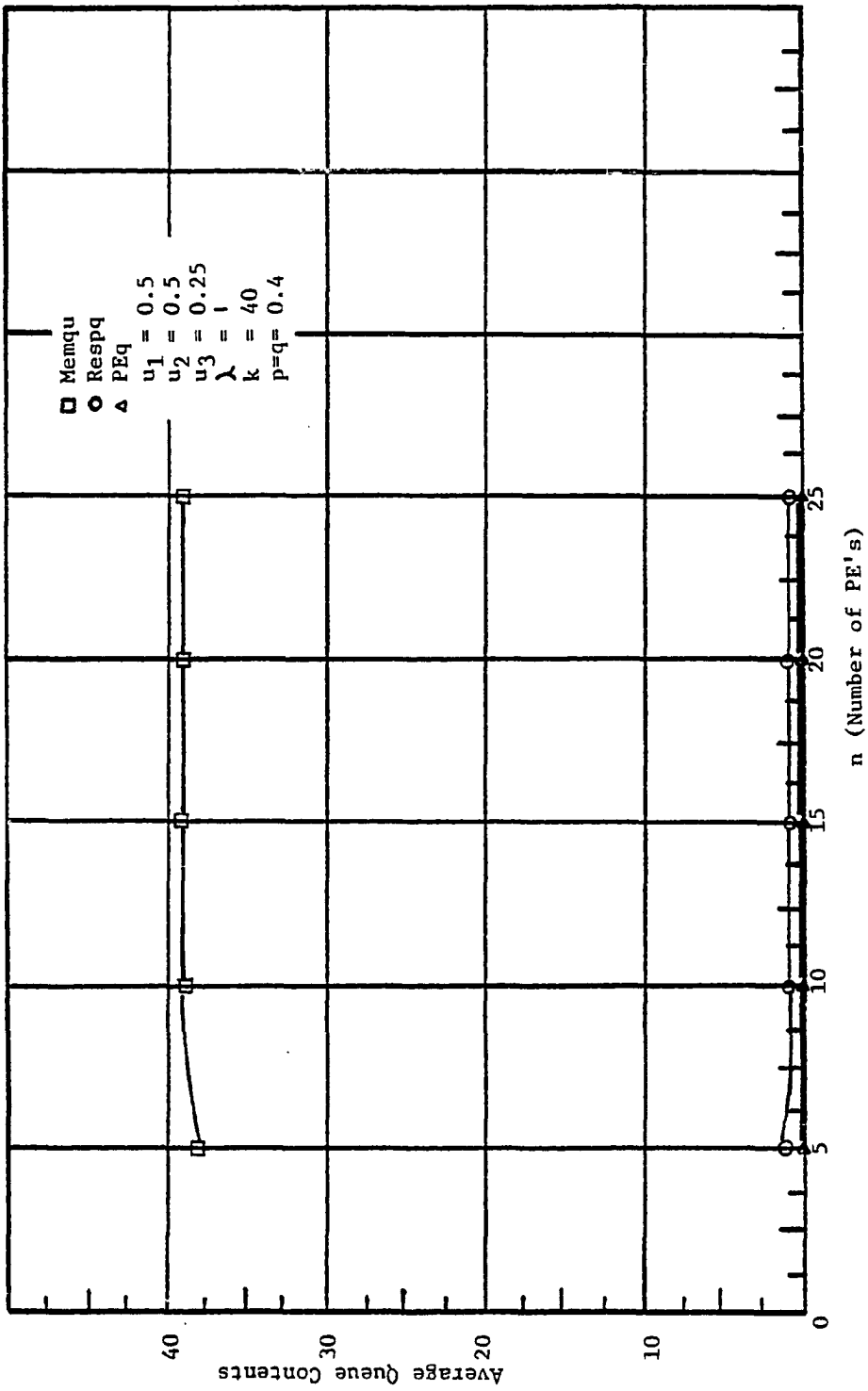


Figure 69: Average Queue Contents as a Function of n

The different queue contents as a function of the number of PE's are shown in Figure 71. The OUQUE content reduces with the introduction of new PE's. As more PE's are added, the probability that the ready instructions remain in the operation unit queue reduces. The content of the queue will approach zero when the number of PE's reaches the maximum degree of parallelism in the program.



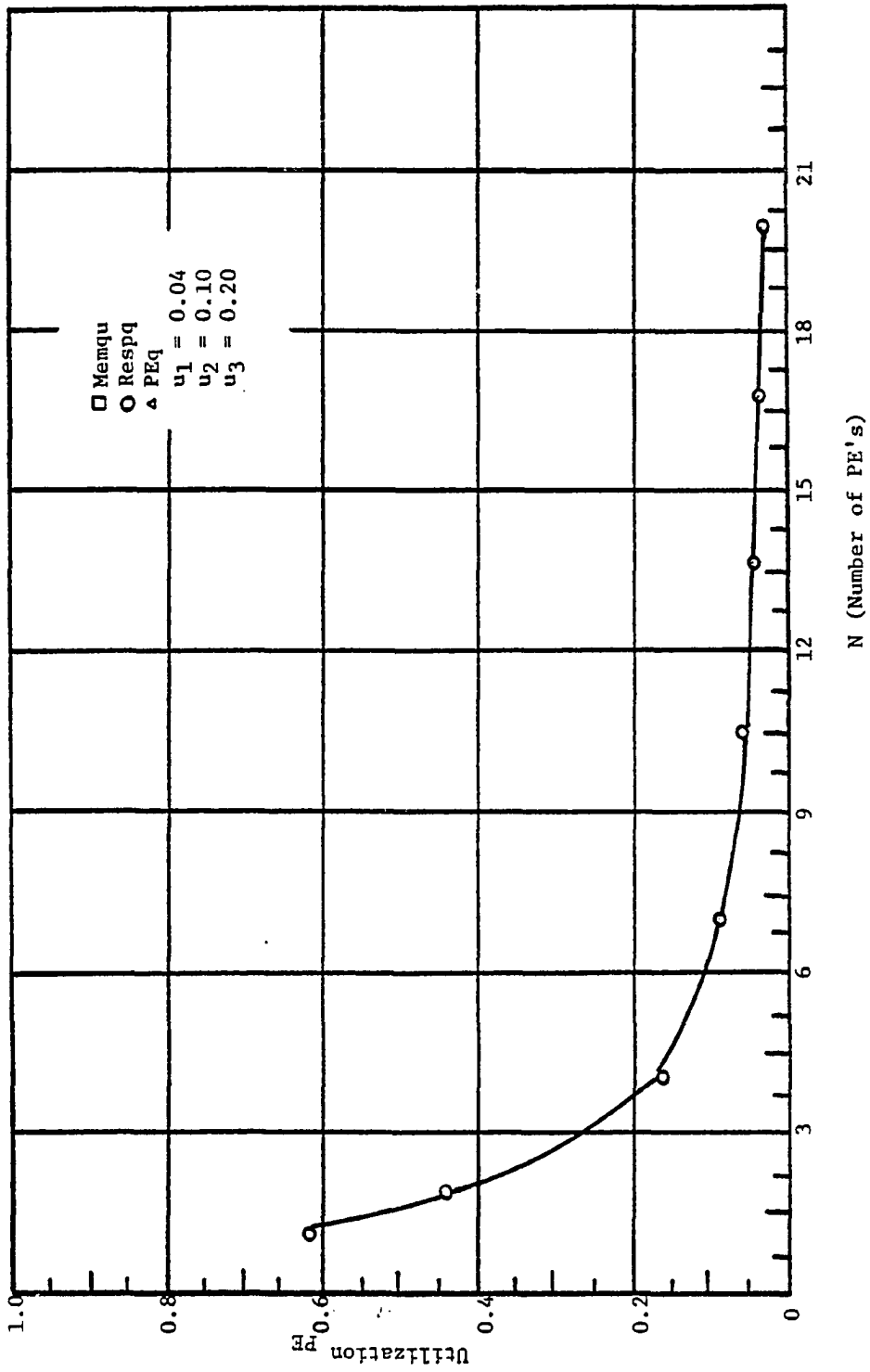


Figure 70: The PE Utilization for the Second Case

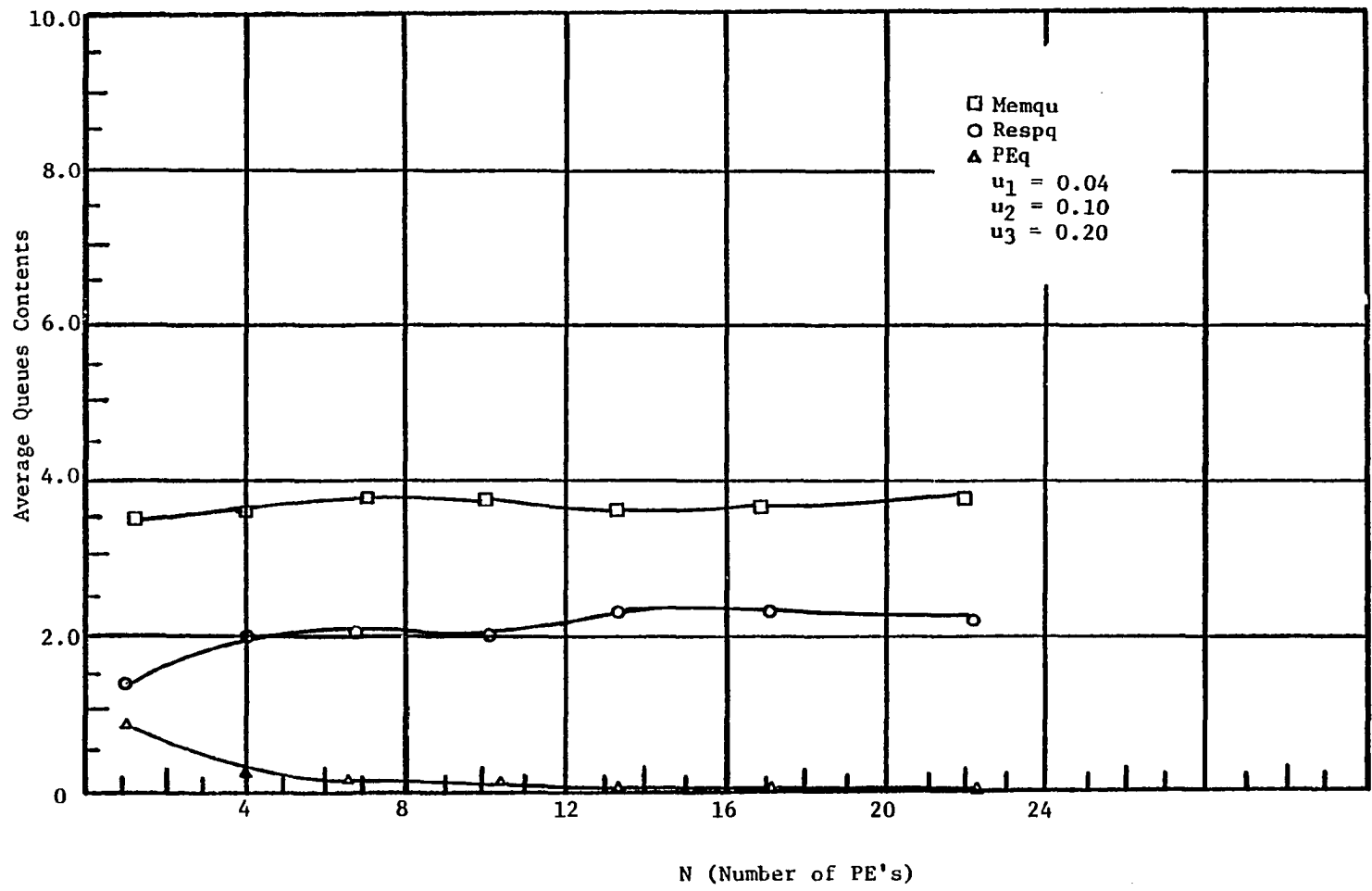


Figure 71: The Average Queue Contents for the Second Case

Chapter VIII  
SUMMARY AND CONCLUSION

8.1 SUMMARY

This dissertation presents a straightforward approach to the analysis of the performance evaluation of parallel architectures utilizing microprogrammable microprocessor elements. Out of the many existing configurations, three particular architectures are studied. We feel that these models and their analyses are representative of a wide class of generalized networks. The methodology presented should be transferable to different network models.

Numerical queueing techniques along with simulation studies are performed. The analytic model is less expensive to study than the simulation techniques; however, the price paid is the labor involved in developing the equations to set up the real model. Simulation techniques in contrast to the queueing techniques can be used to model more complex structures. Nevertheless, both queueing and simulation techniques play essential roles in computer system performance evaluation. Building the mathematical model can be a very difficult stage in the performance evaluation procedure. Emphasis should be placed on the factors that

influence the flow of information in the model, which in turn are subjected to certain assumptions.

Generally, there are two approaches for performance evaluation: deterministic models, and probabilistic models. The task arrival rate and the task service times are usually specified by probabilistic distribution functions. Probabilistic models provide the general overview of the system performance, especially when system parameters are not well developed.

We have modeled and analyzed three basic network architectures: the controlled multiserver model, the array model, and the data flow model. In the controlled multiserver model a single control unit is used to control several independent functional processing elements (each capable of performing specialized tasks). In the array model the control unit controls the whole group of PE's or a subset of the PE group simultaneously. In the third model, the data flow model, the PE's are selected based on their availability. Each PE is considered to be a stand-alone unit, which makes the overall system more reliable.

Certain assumptions were made in each case. Simplifying the queueing model is necessary in most analyses. The techniques used in this study can easily (with minor modifications and depending on the case under study) be applied to study other similar models.

The design procedure is summarized by the following

steps:

1. Setup the queueing (mathematical) model.
2. In order to simplify the model, if possible, combine the service time of two or more consecutive servers into one server (as in the CMM case).
3. Insight is gained by performing the analysis at two levels:
  - a) The job execution level: Some architectures are better analyzed at this level. In particular, in the array model, each job is assigned a different number of processing elements.
  - b) The instruction execution level: For some architectures such as the CMM and DFM this analysis will elaborate investigation of the system parameters. In the DFM, instructions are prepared for execution whenever their operands are ready.
4. Set up the simulation procedure which will supplement the queueing analysis. By repeating the simulation with the assumptions removed, insight will be gained into the effects of the assumptions made in the analytic case.

It is very difficult to directly compare these three models for each model has its own applications and environment of operation. Our intent is not to compare these models nor expect them to be universally applicable, but to provide building blocks and various approaches. We

hope that network researchers can use the ideas presented in these three examples to effectively construct and evaluate their own particular network models. We will, however, make some statements regarding these three models.

For example, in the controlled multiserver case, the analysis is based on the instruction execution, whereas, in the array model, the analysis is based on the job level. Depending on the job under execution, different PE's will be selected with different probabilities in the first model, whereas in the third model (the data flow) the selection of the PE's is done with equal probability. Unlike the array and the multiserver model, the data flow model will perform more reliably in environments where the failure of any of the processing elements pose degradation to the computation.

As a final note, we should emphasize that in general most computer architects agree on the following goals in designing general purpose computing networks:

1. Effective distribution of small pieces of computation over many processors in the system.
2. Enough modularity so that additional blocks of processing elements can be easily added.
3. A measure of fault tolerance so that hardware failure may decrease performance but will not necessarily halt the process.
4. No dependence on expensive interconnection schemes.

The three models of chapters 5-7 support most of the above

criteria.

## 8.2 FUTURE APPLICATION

With the advent of VLSI, we can foresee the important role that these chips play in the design of large computer networks. As computer structures continue to grow in complexity, in size and in diversity, we need to design tools to evaluate the relative merit of different aspects of machine architecture.

For a large computer network it is sometimes desirable to have different computers at different nodes. The NMSU-MBSE provides a basic unit in such network. Using bit-slice microprocessor elements provides better performance both by the speed of the chip in the data path and the capability in performing the emulation in microcode.

A number of applications that require parallel configuration exist such as in image processing, digital filtering, weather forecasts, seismic exploration systems, plus others. Reconfigurable parallel architectures may provide the flexibility that is needed by such systems.

By using a multiple processing elements system, throughput can be improved, and processing requirements and capabilities unobtainable by uniprocessor systems can be satisfied. However, the success of a multiple processor system greatly depends on successful modeling and performance analysis of the target network.

## LIST OF REFERENCES

- [ACKE78] Ackerman, W.B., A Structure Processing Facility for Data Flow Computers, IEEE conference on Parallel Processing, 1978, pp. 166-172.
- [AGER82] Agerwala, T., "Data Flow Systems", COMPUTER, February 1982.
- [ALEX78] Alexandridis, N. A., "Bit-Slice Microprocessor architecture", COMPUTER, June 1978.
- [BACK78] Backus, John, "Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs", Communication ACM, V21, no. 8, August 1978.
- [PAERSO] Eaer, Jean-Loup, Computer System Architecture, Maryland: Computer Science Press, Inc., 1980.
- [BASK75] Baskett, and K.M. Chandy, "Open Closed, and Mixed Network of Queues with Different Classes of Customers", Journal of ACM, Vol.22, no.2, April 1975.
- [BELL71] Bell, Gordon C. and Newell Allen, Computer Structures: Readings and Examples. New York: McGraw-Hill Book Company, 1971.
- [BOBI76] Bobillier, P. A. et al., Simulation with GPSS and GPSS V. Englewood Cliff, N.J.: Prentice-Hall Inc., 1976.
- [BRIG79] Briggs, F. A. et al., PM4- A Reconfigurable Multiprocessor system for Pattern Recognition and Image Processing, AFIPS, 1979, pp. 259-265.
- [BRIG81] Briggs, Faye A., M. D. Dubios, and Kai Hwang, Throughput Analysis and Configuration Design of a Shared-Resource Multiprocessor System: PUMPS, 8th Annual Symposium on Computer Architecture, Minneapolis, Minn., May 1981.
- [EURK56] Burke, P. J., The Output of a Queueing System Operation Research, 4, 1956, pp. 699-704.



- [CHAN81] Chandy, K. M., and C. H. Sauer, Computer Systems Performance Modeling, Englewood cliffs, N.J.: Prentice-Hall, Inc., 1981.
- [CHU81] Chu, Yochan, "Programmin Language and Direct Execution Computer Architecture", Computer, July 1981.
- [COFF80] Coffman, F. G. and Su Kimming So, On the Comparison Between Single and Multiple Processor Elements, IEEE 7th Annual Symposium on Computer Architecture 1980. pp. 72-79.
- [COTE78] Cote, W. F., and R. F. Riccelli, The Design of a Data Driven Processing Element, IEEE Conference on Parallel Processing 1978, pp. 173-180.
- [DAVI78] Davis, A. L., The Architecture and System Method of DDM1: A Recursive Structured Data Driven Machine, IEEE conference on Computer Architecture 1978, pp. 210-215.
- [DENN78] Denning, P. J., and J. P. Buzen, "The Operational Analysis of Queueing Network Models", Computing Surveys, Vol. 10, No. 3, September 1978.
- [DENN74] Dennis, J. B., and D. P. Misunas, A Preliminary Architecture for a Basic Data Flow Processor, IEEE 2nd Annual Conference on Computer Architecture, January 1974, pp. 126-132.
- [DENN80a] Dennis, J. B., et al., Building Blocks for Data Flow Prototypes, IEEE 7th Annual Symposium on Computer Architecture, 1980.
- [DENN80b] Dennis, J. E., "Data Flow Supercomputers", Computer, November 1980, pp. 48-56.
- [DITZ81] Ditzel, D. R., "Reflections on the High Level Language Symbol Computer System", Computer, July 1981.
- [ENSL74] Enslow, P. H., Multiprocessors and Parallel Processing. New York: John Wiley & Sons, 1974.
- [ENSL77] \_\_\_\_\_, "Multiprocessor Organization - A Survey", ACM Computing Surveys, Vol. 9, No. 1, March 1977, pp. 103-129.
- [FENG72] Feng, T., Some Characteristics of Associative/Parallel Processing, Proceedings of the 1972 Sagamore Comp. Conf., Syracuse University, 1972, pp. 5-16.

- [FERR78] Ferrari, Domenico, Computer System Performance Evaluation. Englewood Cliffs, New Jersey: Prentice-Hall, Inc, 1978.
- [FLYN66] Flynn, M. J., Very High Computing Systems, Proceedings of the IEEE 54, 1966, pp. 1901-1909.
- [FLYN72] Flynn, M. J., "Some Computer Organization and Their Effectiveness", IEEE Transaction on Computers, Vol. C21, No. 9, September 1972.
- [GOST79] Gestelow, K. P. and R. E. Thomas, "A View of Data Flow", AFIPS, 1979. pp. 629-636.
- [HAND77] Handler, W., The Impact of Classification Schemes on Computer Architecture, Proceeding of the 1977 International Conference on Parallel Processing IEEE, August 1977, pp. 7-15.
- [HOBB70] Hobbs, L. C., and D. J. Theis, editors, Parallel Processor Systems, Technologies, and Application. New York: Spartan Books, 1970.
- [HUSS70] Husson, S. S., Microprogramming: Principles and Practice, Englewood Cliffs, N.J.: Prentice-Hall, 1970.
- [HWAN79] Hwang, Kai and L. M. Ni, Performance Evaluation of Resource Optimization of Multiple SIMD Computer Organizations, IEEE conference on Parallel Processing, 1979, pp. 86-94.
- [HWAN81] Hwang, K., and L. M. Ni, "Performance Modeling of Shared-Resource Array Processors", IEEE Transaction on Software Engineering, Vol. SE-7, No 4, July 1981.
- [JACK57] Jackson, J. R., Network of Waiting Lines, Operation Research, 5, 1957, pp. 518-521.
- [JOHN80] Johnson, Douglas et al., Automatic Partitioning of Programs in Multiprocessor System, IEEE Comcon Spring, 1980.
- [KAIN75] Kain, R. Y., and K. V. Sastry, "On the Performance of Certain Multiprocessor Computer Organizations", IEEE Transactions on Computers, Vol. C-24, No.11, November 1975.
- [KELL80] Keller, R. M., et al., Data Flow for Hardware Design, IEEE Comcon, 1980.
- [KLEI75a] Kleinrock, L., Queueing Systems Volume I: Theory. New York: John Wiley and Sons, 1975.

- [KLEI75b] \_\_\_\_\_, Queueing Systems Volume II: Computer Applications. New York: John Wiley and Sons, 1975.
- [KUMASO] Kumar, E. S., and E. S. Davidson, "Computer System Design Using a Hierarchical Approach to Performance Evaluation", Communication of the ACM, Vol.23 N9, September 1980, pp. 511-512.
- [LEES0] Lee, Ruby Bei-Loh, Empirical Results on the Speed, Efficiency, Redundancy and Quality of Parallel Computation, Conference on Parallel Processing, 1980, pp. 91-100.
- [LOUI81] Louie, Thelma, "Array Processors: A Selected Bibliography", Computer, September 1981, pp. 53-57.
- [MATT79] Mattheyses, R. M., and S.E. Corny, Models for Specification and Analysis of Parallel Systems, Proceeding of the Conference on Simulation, Measurement and Modelling of Computer Systems, 1978.
- [MEHR80] Mehra, S. K., et al., "A Comparative Study of Some Two-Processor Organization", IEEE Transactions on Computers, Vol. C-29, No.1, January 1980.
- [MICK78] Mick, J. R., Microprogramming Techniques Using the Am2910 Sequencer, San Francisco: Compcon Spring 1978.
- [MICK80] Mick, J., and James Brick, Bit-Slice Microprocessor Design. New York: McGraw-Hill Book Company, 1980.
- [MISU76] Misunas, D. P., Performance Analysis of a Data-Flow Processor, IEEE conference on Parallel Processing, 1976, pp. 100-105.
- [RAMAS0] Ramamoorthy, C. V., and Gary S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets", IEEE Transactions on Software Engineering, Vol. SE-6, No. 5, September 1980.
- [RAUS80] Rauscher, T. G., and P. M. Adams, "Microprogramming: A Tutorial and Survey of Recent Developments", IEEE Transactions on Computers, Vol. C-29, No.1, January 1980.
- [REDD76] Reddi, S. S., and E. A. Feustel, "A Conceptual Framework for Computer Architecture", Computing Surveys, Vol.8, No.2, June 1976, pp. 277-300.

- [REIS80] Reiser, M., and S. S. Lavenberg, "Mean-Value Analysis of Closed Multichain Queueing Networks", Journal of the Association for Computing Machinery, Vol. 27, No. 2, April 1980, pp. 313-323.
- [ROBL81] Robledo, C. S., Computer System Performance Evaluation, Ph.D. Dissertation, Department of Electrical and Computer Engineering, New Mexico State University, Las Cruces, New Mexico 1981.
- [RUMB75] Rumbaugh, J., A Parallel Asynchronous Computer Architecture for Data Flow Programs, Ph.D. Dissertation, MIT, 1975.
- [RUMB77] \_\_\_\_\_, "A Data Flow Multiprocessor", IEEE Transactions on Computers, Vol. C-26, February 1977.
- [SAST73] Sastry, K. V., Markovian Models for Performance Evaluation of Multiprocessor Multimemory Computer System, Ph.D. Dissertation, University of Minnesota, June 1973.
- [SATY80] Satyanarayanan, M., "Multiprocessing: An Annotated Bibliography", Computer, May 1980, pp. 101-116.
- [SAUR75] Sauer, C. H., and K. M. Chandy, "Approximate Analysis of Central Server Models", IBM Journal Research and Developments, May 1975.
- [SCHR74] Schriber, Thomas J., Simulation Using GPSS. New York: John Wiley and Sons, 1974.
- [SIEG79] Siegel, H. J., "A Model of SIMD Machines and a Comparison of Various Interconnection Networks", IEEE Transactions on Computers, Vol. C-28, No. 12, December 1979.
- [SPIR79] Spirn, J. R., "Queueing Networks with Random Selection for Service", IEEE Transactions on Software Engineering, Vol. SE-5, No. 3, May 1979.
- [STON80] Stone, Harold S. editor, Introduction to Computer Architecture. Chicago: Science Research Associates, Inc., 1980.
- [SWAN77] Swan, R. J., et al., Cm\*- A Modular Multi-microprocessor, AFIPS 1977.
- [SWAR79] Swartzlander, E. E., "Microprogrammed Control for Specialized Processor", IEEE Transactions on Computers, Vol. C-28, No. 12, December 1979.

- [VICK80] Vick, C. R., et all, "Adaptable Architecture for Supercomputers", Computer, November, 1980, pp. 17-35.
- [WATS79] Watson, Ian, and John Gurd, A Prototype Data Flow Computer with Token Labelling, AFIPS Conference Proceeding, June 1979, pp. 623-628.
- [WHIT75] White, J. A., et al., Analysis of Queueing Systems. New York: Academic Press Inc, 1975.
- [WONG78] Wong, J. W., "Queueing Network Modeling of Computer Communication Networks", Computing Surveys, Vol. 10, No. 3, September 1978, pp. 343-351.
- [ZEMAS0] Zeman, Jan, and H. T. Nagle, "A High Speed Microprogrammable Digital Signal Processor Employing Distributed Arithmetic", IEEE Transactions on Computers, Vol. C-29, No.2, February 1980, pp. 134-144.

## Appendix A

### SOLUTION FOR THE CASE N=3 AND C=3 OF THE CONTROLLED MULTISERVER MODEL OF CHAPTER V

Let us start with the probability state vector,

$$\begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \end{bmatrix} = [B]^{-1} \begin{bmatrix} A \\ \\ \end{bmatrix} \quad \begin{bmatrix} P_{21} \\ P_{22} \\ P_{23} \end{bmatrix} + P_0 \lambda \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad \left. \vphantom{\begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \end{bmatrix}} \right\} \text{-----} 1$$

$$\begin{bmatrix} P_{21} \\ P_{22} \\ P_{23} \end{bmatrix} = [B]^{-1} \begin{bmatrix} A \\ \\ \end{bmatrix} \quad \begin{bmatrix} P_{31} \\ P_{32} \\ P_{33} \end{bmatrix} + [L] \begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \end{bmatrix} \quad \left. \vphantom{\begin{bmatrix} P_{21} \\ P_{22} \\ P_{23} \end{bmatrix}} \right\} \text{-----} 2$$

and finally:

$$\begin{bmatrix} P_{31} \\ P_{32} \\ P_{33} \end{bmatrix} = \lambda \begin{bmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \mu_3 \end{bmatrix}^{-1} \begin{bmatrix} P_{21} \\ P_{22} \\ P_{23} \end{bmatrix} \quad \text{-----} 3$$

let

$$\mu_i = \begin{bmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \mu_3 \end{bmatrix}$$

Substitute Eq 3 into Eq 2 and solving for  $[P_{2i}'s]$

$$\begin{bmatrix} P_{21} \\ P_{22} \\ P_{23} \end{bmatrix} = \left\{ [I] - [B]^{-1} [A] \lambda [\mu_i]^{-1} \right\}^{-1} [B]^{-1} [L] \begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \end{bmatrix} \quad \text{--- 2'}$$

Now substitute Eq 2' into Eq 1 and solving for  $[P_{1i}'s]$  we obtain

$$\begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \end{bmatrix} = \left( [I] - [B]^{-1} [A] \left[ \left\{ [I] - [B]^{-1} [A] \lambda [\mu_i]^{-1} \right\} [B]^{-1} [L] \right] \right)^{-1} P_0 \lambda \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \dots$$

all the quantities in Eq 1' are known except  $P_0$ . Plugging Eq 1' back into Eq 2' then  $[P_{2i}'s]$  are solved for also in terms of  $P_0$ .

Finally plugging in for  $[P_{2i}'s]$  in Eq 3 then  $[P_{3i}'s]$  are again solved for in terms of  $P_0$ . By using the normalizing ratio in Eq 5-23  $P_0$  is then found. Lastly, by substituting for  $P_0$  in Eq's 1', 2', and 3, the state probabilities are found. Once known, the state probabilities are then used to find the different utilizations, i.e. for each PE and for the whole system. Moreover the different parameters like throughput can be computed.

## Appendix B

### THE ANALYTIC PROGRAM FOR THE CONTROLLED MULTISERVER MODEL OF CHAPTER V

```

▽MODEL3[[]]▽
▽ MODEL3
[1]  A THIS PROGRAM IS FOR THE CONTROLLED MULTISERVER CASE
[2]  A -----
[3]  'ENTER THE NO. OF PE'S.'
[4]  'N = ',(+N←[])
[5]  'ENTER THE CAPACITY OF THE SYSTEM, I.E. THE QUEUE LENGTH '
[6]  'PLUS THE NUMBER OF SERVERS , '
[7]  'C = ',(+C←[])
[8]  'ENTER THE ARRIVAL RATE, XACT, /UNIT TIME'
[9]  'K = ',(+K←[])
[10] 'ENTER THE SERVICE RATE FOR EACH PE ,SHOULD BE OF DIMENSION Nx1'
[11] 'UV = '
[12] UV←(N,1)P[]
[13] 'ENTER THE PROBABILITY VECTOR, PV, IT SHOULD HAVE THE '
[14] ' DIMENSION OF Nx1, WHERE N IS THE NUMBER OF PE'S.'
[15] PV←(N,1)P[]
[16] A PV←(N,1)P .1 .1 .1 .1 .1 .1 .1 .1 .1
[17] I←(N,N)P1,NP0 A GENERATE AN IDENTITY MATRIX
[18] UT←(N,N)PUV
[19] U1←IXUT
[20] T1←(N,N)P PV
[21] A1←QT1
[22] A←A1+.XU1
[23] B1←(N,N)PK,NP0
[24] B←B1+U1
[25] S←(B)+.XA
[26] L←KXI
[27] T←(B)+.XL
[28] W←(0,N,N)P0 A INITIALIZATION OF THE W AND P ARRAYS
[29] A -----
[30] P←(0,N,1)P0
[31] J←1
[32] W←(1,N,N)P(I-S+.X(KX(BU1)))
[33] WS;W←W,[1](1,N,N)P(I-S+.X(BW[J;]))+.XT)
[34] J←J+1
[35] →OUTX1(J)(C-1)
[36] →WS
[37] OUT;J←(C-1)
[38] H←1
[39] P←(1,N,1)P((BW[J;])+.X(B)+.X(KXPV))
[40] PS;J←J-1

```



```

[41] P←P,[1](1,N,1)P((BW[U;])+.XT+.XP[H;])
[42] H←H+1
[43] →EXITX\ (J;1)
[44] →PS
[45] EXIT:P←P,[1](1,N,1)P((KX(U1))+.XP[H;])
[46] PT←0
[47] H←1 A CALCULATION OF P0
[48] A -----
[49] COMPUT:PT←PT+(+/P[H;1])
[50] H←H+1
[51] →DONEX\ (H;C)
[52] →COMPUT
[53] DONE:P0←(1÷(1+PT))
[54] 'P0 = ',(P0)
[55] UTILIZ←(1-P0)X100
[56] 'UTILIZ = ',(UTILIZ)
[57] PP←PXP0
[58] 'THE STATE PROBABILITIES ARE '
[59] 'PP = '
[60] A QPP
[61] A IN ORDER TO FIND THE UTILIZATION OF EACH PE THEN DO:
[62] A P0+P11+P21+P31...+PN1=1; THEN P0=1-SUM PI FOR I=1 TO N
[63] A THEN THE UTILIZATION OF PE1=SUM PI FOR ALL I'S
[64] H←1 A BEGIN; THE CALCUL. OF UTILIZ. FOR EACH PE
[65] A -----
[66] PE←(0,1,1)P0
[67] SUMI:TOT←(+/PP[;H;1])
[68] 'P',(P), 'PE = ',(TOT)
[69] PE←PE,[1](1,1,1)PTOT
[70] H←H+1
[71] →OKX\ (H;N)
[72] →SUMI
[73] OK:'PE = ' A END; THE CALCUL. OF UTILIZ. FOR EACH PE
[74] A -----
[75] A PE
[76] I←1 A BEGIN; CALCULATION OF THE EXPECTED NO IN THE SYSTEM,
[77] A -----
[78] TEMP←0
[79] NOIN:TEMP←TEMP+(IX(+/QPP[I;1]))
[80] I←I+1
[81] →OUTLX\ (I;C)
[82] →NOIN A END; CALCUL. OF THE EXPECTED NO IN THE SYSTEM,
[83] A -----
[84] OUTL:'EXPECTED NO IN THE SYSTEM = ',(TEMP), ' TRANSACTIONS '
[85] A CALCULATION THE AVERAGE NO IN QUEUE
[86] A -----
[87] NOINQUE←0
[88] I←2

```

```

[89] QUE;NOINQUE←NOINQUE+((I-1)X(+/PF[I;1]))
[90] I←I+1
[91] →QUEOUTX\{I>C}
[92] →QUE
[93] QUEOUT:'AVG, NO IN QUEUE = ',(+NOINQUE),' INSTRUCTIONS'
[94] A CALCULATION THE AVG, RESPONSE TIME
[95] A -----
[96] PROBFUL←(+/PF[C;1])
[97] RESPTIME←(TEMP÷(KX(1-PROBFUL)))
[98] 'AVG, RESPONSE TIME = ',(+RESPTIME),' TIME UNIT'
[99] A CALCULATION THE AVG, TIME IN THE QUEUE
[100] A -----
[101] QUETIME←(NOINQUE÷(KX(1-PROBFUL)))
[102] 'AVG, TIME IN QUEUE = ',(+QUETIME),' TIME UNIT'
[103] A SYSTEM THROUGHPUT
[104] A -----
[105] THROU←(UTILIZ÷100)X((+/UV[1])÷N)
[106] 'SYSTEM THROUGHPUT = ',(+THROU),' INSTR/TIME UNIT'
[107] 'SYSTEM UTILIZATION = ',(+UTILIZ),' %'
[108] 'ARRIVAL RATE = ',(+K),' INSTRUCTION /TIME UNIT'
[109] 'SERVICE RATE VECT, = '
[110] QUV
[111] 'PROB, SELCTION VECT, = '
[112] QPV
[113] 'NO OF PE'S = ',(+N),' ,AND SYSTEM CAPACITY = ',(+C)

```

▽

▽MODEL2[[]]▽

## Appendix C

### THE SIMULATION PROGRAM FOR THE CONTROLLED MULTISERVER MODEL OF CHAPTER V

```

SIMULATE
*****
*   MODEL31
*   THIS PROGRAM SIMULATES THE EXECUTION OF INSTRUCTIONS
*   ON THE CONTROLLRDD MULTI-SEVER MACHINE AS WELL AS JOB
*   ARRIVALS. IN THIS PART (I) WE ASSUME THAT
*   THE NUMBER OF PROGRAMS (JOBS) IN THE MAIN MEMORY TO BE
*   FIXED AND GIVEN BY THE 1ST SEGMENT OF THE PROGRAM
*****
      RMULT      111,333,555,777
*   FUNCTIONS SPECIFICATION
XPDIS FUNCTION  RN1,C24      EXPONENTIAL DISTRIBUTION
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
*
PENO FUNCTION  RN1,D9      FOR THE PE NUMBER TO BE ALLOCATED.
0,0/.04,3/.09,4/.15,5/.25,6/.4,7/.55,8/.75,9/1.,10
*
*   THE ABOVE FUNCTION IS USED TO CALCULATE THE PROBABILITY OF
*   SELECTING THE NEXT PE TYPE, I.E. THE PROB. OF SELECTING
*   PE NO 1 IS 0%, THE PROBABILITY OF SELECTING PE NO 3 IS 4%,
*   AND THE PROBABILITY OF SELECTING PE NO 4 IS 5% ... ETC.
*
PESER FUNCTION  RN4,C2      FOR THE PE SEVICE TIME ASSIGNMENT
0,75/1.,135
*
*   NOTE THAT EVERY PE WILL HAVE
*   DIFFRENT SEVICE TIME.
CUSER FUNCTION  RN1,C2      FOR THE CONTROLLER SERVICE TIME
0,180/1.,220
*
*   ASSIGNMENT.
INSTR FUNCTION  RN2,C2      FOR THE ASSIGNMENT OF THE NUMBER OF
0,50/1.,70
*
*   INSTRUCTIONS.
ENTRY VARIABLE  Q$RDY
SUM VARIABLE    FN$PESER+FN$CUSER
INITIAL         X$INST,0    INITIALIZATION OF THE NUMBER OF ISTRU-
CUPE EQU        50,F
*
*   CTIONS IN EACH JOB .
*   STORAGE     S$CPU,10    SPECIFY THE NUMBER OF PE'S IN THE
*                           SYSTEM.

```

```

*      SEGMENT 1      (ASSOCIATED WITH THE NUMBER OF JOBS IN THE
*                    SYSTEM.)
JOBS  GENERATE      ,,,5      JOB ARRIVAL
      QUEUE        MEM
      GATE LR      SYS          THIS WILL MAKE SURE THAT ONLY ONE JOB IS
      LOGIC S      SYS          ACTIVE AT ANY GIVEN TIME
      DEPART      MEM
      SAVEVALUE    INST,FN$INSTR  ASSIGNMENT OF THE NUMBER
*                                     OF INSTRUCTIONS.
      TERMINATE    1
*
*      SEGMENT 2
*
ARIVE GENERATE      200,FN$XPDIS  GENERATE THE MACRO INSTRUCTIONS
      ENTER        SYSTM
      QUEUE        DUMMY
CAPCY TEST L        V$ENTRY,10
      DEPART      DUMMY
      ASSIGN      1,FN$CUSER      PARAMETERS ASSIGNMENTS FOR CU
      ASSIGN      2,FN$PESER      AND PE SERVICE TIMES.
      ASSIGN      3,V$SUM         THE COMBINED PE AND CU SERVICE TIME
      ASSIGN      4,FN$PENO      ASSIGN THE PE NUMBER TO THE
*                                     INSTRUCTION.
      GATE LR      NEXT
*      LOGIC S      NEXT
      GATE NU      CUPE
      SEIZE        CUPE
*      DEPART      RDY
*      ADVANCE     P1,FN$XPDIS
      SEIZE        P4
      DEPART      RDY
      ADVANCE     P3,FN$XPDIS
      SAVEVALUE    INST-,1
      TEST LE     X$INST,0,OUT
*OUT  LOGIC R      SYS
      OUT  LOGIC R  NEXT
      RELEASE     CUPE
      RELEASE     P4
      LEAVE       SYSTM
      TERMINATE
*
*      THE TIMER
*
*      GENERATE    50000
*      TERMINATE   1
*
*      THE CONTROL CARDS
*
*      START      5
*
*      RMULT      111,333,555,777
*      CLEAR

```

```
*CAPCY TEST L      V$ENTRY,10
PENO FUNCTION      RN1,C2
0,1/1.,16
  START           5
*
  RMULT           111,333,555,777
  CLEAR
*CAPCY TEST L      V$ENTRY,15
PENO FUNCTION      RN1,C2
0,1/1.,21
  START           5
*
  RMULT           111,333,555,777
  CLEAR
*CAPCY TEST L      V$ENTRY,20
PENO FUNCTION      RN1,C2
0,1/1.,26
  START           5
  END
```

```

SIMULATE
*****
*   MODEL33   *
*   THIS PROGRAM SIMULATES THE EXECUTION OF INSTRUCTIONS *
*   ON THE CONTROLLRDD MULTI-SEVER MACHINE AS WELL AS JOB *
*   ARRIVALS. IN THIS CONFIGURATION (III) WE ASSUME THAT *
*   THE NUMBER OF PROGRAMS (JOBS) IN THE MAIN MEMORY TO BE *
*   FIXED AND GIVEN BY THE 1ST SEGMENT OF THE PROGRAM *
*   THE CU AND THE SPE SERVICE TIMES ARE SEPARATE. *
*   THE MODIFICATION MADE IN THIS CASE IS THAT THE CU CAN *
*   SERVE MORE THAN ONE INSTRUCTION AT ANY GIVEN TIME *
*****
      RMULT      111,333,555,777
*   FUNCTIONS SPECIFICATION
*   XPDIS FUNCTION  RN1,C24   EXPONENTIAL DISTRIBUTION
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
*
      PENO FUNCTION  RN1,C2   FOR THE PE NUMBER TO BE ALLOCATED.
0,1/1.,6
*0,0/.04,3/.09,4/.15,5/.25,6/.4,7/.55,8/.75,9/1.,10
*
*   THE ABOVE FUNCTION IS USED TO CALCULATE THE PROBABILITY OF
*   SELECTING THE NEXT PE TYPE, I.E. THE PROB. OF SELECTING
*   PE NO 1 IS 0%, THE PROBABILITY OF SELECTING PE NO 3 IS 4%, AND
*   THE PROBABILITY OF SELECTING PE NO 4 IS 5% ... ETC.
*
      PESER FUNCTION  RN4,C2   FOR THE PE SEVICE TIME ASSIGNMENT
0,75/1.,135
*
*   NOTE THAT EVERY PE WILL HAVE
*   ITS OWN SERVICE TIME.
      CUSER FUNCTION  RN1,C2   FOR THE CONTR. SERVICE TIME ASSIGNMENT.
0,180/1.,220
      INSTR FUNCTION  RN2,C2   FOR THE ASSIGNMENT OF THE NUMBER OF
0,50/1.,70
*
*   INSTRUCTIONS.
      ENTRY VARIABLE  Q$RDY
      SUM   VARIABLE  FN$PESER+FN$CUSER
      INITIAL X$INST,0/X$CPUS,0   INITIALIZATION OF THE
      INITIAL X$CNTR,0/X$CHCK,0   NUMBER OF INSTRUCTIONS AND
      INITIAL X$NEW,0/X$LAST,0   THE NUMBER OFPE'S IN THE
      CUNT  EQU      50,F        SYSTEM.
      STORAGE S$CPU,10        SPECIFY THE NUMBER OF PE'S
*
*   .. THE SYSTEM.
*
*   SEGMENT 1 (ASSOCIATED WITH THE JOBS IN THE SYSTEM.)
*
      ARIVE GENERATE  ,,5   JOB ARRIVAL
      QUEUE          MEM
      GATE LR        SYS   THIS WILL MAKE SURE THAT ONLY ONE JOB
      LOGIC S        SYS   IS ACTIVE.

```

```

DEPART      MEM
SAVEVALUE  INST, FN$INSTR  ASSIGNMENT OF THE NUMBER
SAVEVALUE  CHCK, X$INST   INSTRUCTIONS.
TERMINATE  1

*
*
*   SEGMENT 2
*
*   GENERATE 150, FN$XPDIS  GENERATE THE MACRO INSTRUCTIONS
*   QUEUE    DUMMY        IS A CONCEPTUAL QUEUE
CAPAC TEST L V$ENTRY, 30  TESTING FOR THE SYSTEM CAPACITY
*   THE QUEUE LENGTH +SYSTEM CAPACITY
*
*   ENTER    SYSTEM
DEPART      DUMMY
ASSIGN      1, FN$CUSER    THE CONTROLLER SERVICE TIME
ASSIGN      2, FN$PESER    AND PE SERVICE TIMES.
ASSIGN      3, V$SUM       THE COMBINED PE AND CU SERVICE TIME
ASSIGN      4, FN$PEN0    ASSIGN THE PE NUMBER TO THE
QUEUE       RDY           INSTRUCTIONS.
SAVEVALUE   LAST, P4
SEIZE       CUNT
DEPART      RDY
ADVANCE     P1, FN$XPDIS
*   TEST NE  X$NEW, X$LAST  SINCE EACH PE DOES NOT HAVE ITS OWN
*   SAVEVALUE NEW, X$LAST  QUEUE, THEN IT IS NECESSARY TO CHECK
GATE NU     P4
RELEASE     CUNT          FOR THE IDLE STATE OF THE PE.
SEIZE       P4
ADVANCE     P2, FN$XPDIS
RELEASE     P4
SAVEVALUE   INST-, 1
TEST G      X$INST, 0, TERM
LEAVE      SYSTEM
TRANSFER   , DONE
TERM       LOGIC R
DONE      SYS
TERMINATE

*
*   GENERATE 50000
*   TERMINATE 1
*
*   START    1, NP
*   RESET
*
*   ARIVE GENERATE , , , 5
*   START    5
*
*   RMULT    111, 333, 555, 777
*   CLEAR
*   PENO FUNCTION RN1, C2
0, 1/1., 11
*   START    5
*
*   RMULT    111, 333, 555, 777

```

```
      CLEAR
*CAPAC TEST L      V$ENTRY,14
  PENO FUNCTION    RN1,C2
0,1/1.,16
  START           5
*
      RMULT         111,333,555,777
      CLEAR
*CAPAC TEST L      V$ENTRY,18
  PENO FUNCTION    RN1,C2
0,1/1.,21
  START           5
*
      RMULT         111,333,555,777
      CLEAR
*CAPAC TEST L      V$ENTRY,20
  PENO FUNCTION    RN1,C2
0,1/1.,26
  START           5
*
      RMULT         111,333,555,777
      CLEAR
*CAPAC TEST L      V$ENTRY,30
  PENO FUNCTION    RN1,C2
0,1/1.,31
  START           5
      END
```



## Appendix D

### THE ANALYTIC PROGRAM FOR THE ARRAY PROCESSING MODEL OF CHAPTER VI

```

▽ MODEL_2
[1]  OFF←4
[2]  A THIS PROGRAM REPRESENTS THE ANALYTIC SOLUTION FOR AN SIMD MODEL
[3]  A -----
[4]  'ENTER THE NO OF PROCESSING ELEMENTS, IT SHOULD BE '
[5]  'GREATER THAN 3.'
[6]  'N=',(↑N←□)
[7]  'ENTER THE SYSTEM CAPACITY, I.E. QUEUE LENGTH + SERVER'
[8]  'C = ',(↑C←□)
[9]  'ENTER THE ARRIVAL RATE'
[10] 'K=',(↑K←□) # JOBS PER UNIT TIME
[11] 'ENTER THE SERVICE RATE'
[12] 'U=',(↑U←□) # JOBS PER UNIT TIME
[13] KK←K+U
[14] 'ENTER PV ; THE PROBABILITY VECTOR, IT SHOULD HAVE THE LENGTH OF N '
[15] 'THE PROBABILITY OF THE JOB SELECTING A CERTAIN NO. OF PE'S.'
[16] A PV←(N,1)P□
[17] PV←(N,1)P 0 0 0.04 0.05 0.06 0.1 0.15 0.15 0.2 0.25
[18] I←(N,N)P1,NP0
[19] B←(N,N)P(K+U),NP0
[20] 'THE B MATRIX: = '
[21] A B
[22] T1←(N,N)P PV
[23] A1←QT1
[24] 'THE A1 MATRIX: = '
[25] A A1
[26] A←UXA1
[27] L←KXI
[28] W←(0,N,N)P0 #INITIALIZATION OF THE W AND P ARRAYS
[29] A -----
[30] P←(0,N,1)P0
[31] Z←1
[32] E←C-4
[33] S←(EB)+.XA
[34] T←(EB)+.XL
[35] #I-SXKK #BEGIN: CALCULATION OF W ARRAY
[36] A -----
[37] Y←I-(S+.X(EB)+.XT)
[38] W←W,[1](1,N,N)P(I-S+.X(EB)+.XT)
[39] CONTINUE:W←W,[1](1,N,N)P(I-S+.X(EB[Z;]);)+.XT)

```

```

[40] E←E-1
[41] Z←Z+1
[42] →PSX\ (E≤0)  AEND: CALC. OF W ARRAY
[43] A -----
[44] →CONTINUE
[45] PS; J+(C-3)  ABEGIN: CALC. OF P ARRAY
[46] A -----
[47] A NOTE THAT ALL THE ELEMENTS OF THE P VECTOR ARE IN TERMS OF P0
[48] H←1
[49] P←P, [1](1,N,1)P((BW[J;§])+.XT+.XFV)
[50] BACK; P←P, [1](1,N,1)P((BW[(J-1);§])+.XT+.XF[H;§])
[51] J←J-1
[52] H←H+1
[53] →OUTX\ (J≤1)
[54] →BACK
[55] OUT; →EXITX\ (N≤3)  A THIS STATEMENT SHOULD NEVER BE EXECUTED
[56] A SINCE N IS > 3
[57] P←P, [1](1,N,1)P((BY)+.XT+.XF[(C-3);§])
[58] P←P, [1](1,N,1)P((BX)+.XT+.XF[(C-2);§])
[59] P←P, [1](1,N,1)P((KX)+.XT+.XF[(C-1);§])
[60] EXIT: 'THE FOLLOWING ARE IN TERMS OF P0'  AEND: CALC. OF P ARRAY
[61] A -----
[62] A  QP
[63] H←0  A BEGIN: COMPUTE P0
[64] A -----
[65] PT←0
[66] SUM; H←H+1
[67] PT←PT+(+/P[H;§1])
[68] →RESULTX\ (H≤C)
[69] →SUM
[70] RESULT; P0←(1/(1+PT))
[71] 'P0 = '  AEND: COMPUTE P0
[72] A -----
[73] P0
[74] UTILIZ←(1-P0)X100
[75] THROU←(UTILIZXU)÷100
[76] 'THE FINAL PROBABILITIES ARE : '
[77] 'PP = PXP0 = '
[78] PP←PXP0
[79] A  QPP
[80] ''
[81] A CALCULATION OF THE TOTAL CPU UTILIZATION
[82] J←1
[83] UTI←0
[84] REPEAT; UTI←UTI+(JX(+/PP[§J;1]))
[85] J←J+1
[86] →STOPX\ (J>N)
[87] →REPEAT

```

```

[88] STOP;UTILI←(UTI+N)X100
[89] PE←(0,1,1)PO A BEGIN: CALCUL, THE UTILIZATION OF A CERTAIN
[90] A -----
[91] A NUMBER OF PE'S
[92] A -----
[93] H←1
[94] SUMI;TOT←(+/PP[;M;1]) A SUMMATION OF P1I+P2I+P3I+...+PNI FOR ALL I'S
[95] A 'P',(↑H),'PE', ' = ',(↑TOT)
[96] PE←PE,[1](1,1,1)PTOT
[97] →DOEX1(M;N)
[98] H←H+1
[99] →SUMI
[100] DONE;'PE[1] REPRESENTS THE PROB. OF 1 PE BEING UTILIZED;'
[101] ' PE[2] REPRESENTS THE PROB. OF 2 PE'S BEING UTILIZED, ETC.
[102] 'PE = '
[103] PE
[104] A END: THE CALCULATION OF THE UTILIZATION OF PE'S
[105] A -----
[106] I←1 A BEGIN: CALCULATION OF THE EXPECTED NO IN THE SYST, AND IN THE QUE
[107] A -----
[108] EXPECT←0
[109] QUELEN←0
[110] ECUST;EXPEC←EXPEC+(IX(+/PP[I;1]))
[111] QUELEN←QUELEN+((I-1)X(+/PP[I;1]))
[112] I←I+1
[113] →OUTCX1(I;C)
[114] →ECUST A END: CALCULA, OF THE EXPECTED NO OF TRANSAC, IN THE SYST.
[115] A -----
[116] OUTC;'THE EXPECTED NO IN THE SYSTEM = ',(↑EXPEC),' TRANSACTIONS'
[117] 'THE AVG, QUEUE LENGTH', ' = ',(↑QUELEN),' TRANSACTIONS'
[118] A QUELEN = NQ
[119] WAIT←+/PP[C;1]
[120] WAIT←1-WAIT
[121] TWAIT←(1+QUELEN)+(K;WAIT)
[122] 'AVERAGE JOB RESPONSE TIME = ',(↑TWAIT),' TIME UNITS'
[123] 'SYSTEM UTILIZATION = ',(↑UTILIZ)
[124] 'CPU'S UTILIZATION = ',(↑UTILI)
[125] 'SYSTEM THROUGHPUT = ',(↑THROU)
[126] 'ARRIVAL RATE = ',(↑K),' JOBS /TIME UNIT'
[127] 'SERVICE RATE = ',(↑U),' JOBS /TIME UNIT'
[128] 'PROB, ALLOCATION VECT, = '
[129] QPV
[130] 'NUMBER OF PROCESSORS = ',(↑N),' AND SYSTEM CAPACITY = ',(↑C)
[130] 7

```

## Appendix E

### THE SIMULATION PROGRAMS FOR BOTH ARRAY MODELS OF CHAPTER VI

```

SIMULATE
*****
*   MODEL2A (RUN #1 C=5) )=.7                               *
*   MACROANALYSIS OF THE ARRAY SYSTEM                       *
*   THIS PROGRAM SIMULATES THE EXECUTION OF JOBS           *
*   ON THE ARRAY MACHINE .IT IS THE SIMULATION VERSION OF  *
*   THE ANALYTIC CASE.                                     *
*****
      RMULT      11,33,55,77
*   FUNCTIONS SPECIFICATION
XPDIS FUNCTION  RN1,C24      EXPONENTIAL DISTRIBUTION
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
*
NOPES FUNCTION  RN1,D9      FOR THE NUMBER OF PE'S TO BE ALLOCATED.
0,0/.04,3/.09,4/.15,5/.25,6/.4,7/.55,8/.75,9/1.,10
*
*   THE ABOVE FUNCTION IS USED TO CALCULATE THE PROBABILITY OF
*   ALLOCATING THE NEXT GROUP OF PE'S, I.E. THE PROB. OF
*   allocating 1 PE IS 0%, THE PROBABILITY OF ALLOCATING 3 PE'S
*   is 4%, andTHE PROBABILITY OF ALLOCATING 4 PE'S IS 5%...ETC.
*
PESER FUNCTION  RN4,C2      FOR THE PE SEVICE TIME ASSIGNMENT
0,70/1.,135
*
*   NOTE THAT ALL THE PE S WILL HAVE
*   THE SAME SERVICE TIME.
CUSER FUNCTION  RN1,C2      FOR THE CONTRL. SERVICE TIME ASSIGN.
0,180/1.,220
*
*   VARIABLE SPECIFICATION
*
ENTRY VARIABLE  Q$MAIN
EXPEC FVARIABLE (7*(ST$SYSTEM))      EXPECTED NUMBER IN THE SYSTEM
*                                     EQUALS THE ARRIVAL RATE X AVG.
*                                     TIME EACH XACT STAYS IN THE
*                                     SYSTEM. DEVIDE THE RESULT BY
*                                     1000 FOR WE MULTIPLIED BY 1000.
THROU FVARIABLE (SR$CPU*(SC$CPU/ST$CPU))
THROS FVARIABLE (SR$SYSTEM*(SC$SYSTEM/ST$SYSTEM))*100
INITIAL        X$CPUS,0          INITIALIZATION OF THE NUMBER

```



```

*KEY3 SAVEVALUE SYCAP,5
      STORAGE S$CPU,5
NOPE$ FUNCTION RN1,D5
0,0/.1,2/.3,3/.6,4/1.,5
      START 60
*
      RMULT 11,33,55,77
      CLEAR
*EXPEC FVARIABLE (4*(ST$SYSTEM))
*KEY GENERATE 25, FN$XPDIS
*KEY3 SAVEVALUE SYCAP,6
      STORAGE S$CPU,8
NOPE$ FUNCTION RN1,D7
0,0/.05,3/.1,4/.25,5/.4,6/.6,7/1.,8
      START 60
*
      RMULT 11,33,55,77
      CLEAR
*EXPEC FVARIABLE (5*(ST$SYSTEM))
*KEY GENERATE 20, FN$XPDIS
*KEY3 SAVEVALUE SYCAP,8
      STORAGE S$CPU,12
NOPE$ FUNCTION RN1,D9
0,0/.02,5/.06,6/.1,7/.2,8/.3,9/.5,10/.7,11/1.,12
      START 60
*
      RMULT 11,33,55,77
      CLEAR
*EXPEC FVARIABLE (7*(ST$SYSTEM))
*KEY GENERATE 14, FN$XPDIS
*KEY3 SAVEVALUE SYCAP,29
      STORAGE S$CPU,15
NOPE$ FUNCTION RN1,D11
0,0/.04,6/.1,7/.15,8/.2,9/.25,10/.3,11/.4,12/.5,13/.7,14/1.,15
      START 60
*
      RMULT 11,33,55,77
      CLEAR
      STORAGE S$CPU,20
NOPE$ FUNCTION RN1,D11
0,0/.04,11/.1,12/.15,13/.2,14/.25,15/.3,16/.4,17/.5,18/.7,19/1.,20
      START 60
      END

```

```

SIMULATE
*****
* THE MICRO-ANALYSIS OF THE ARRAY SYSTEM *
* C=5 AND VARY ) *
* THIS PROGRAM SIMULATES THE EXECUTION OF INSTRUCTIONS *
* ON THE ARRAY MACHINE AS WELL AS JOB ARRIVALS. WE ASSUME *
* THE NUMBER OF PROGRAMS (JOBS) IN THE MAIN MEMORY TO BE *
* FIXED AND GIVEN BY THE 1ST SEGMENT OF THE PROGRAM *
*****
      RMULT      11,33,55,77
* FUNCTIONS SPECIFICATION
XPDIS FUNCTION  RN1,C24  EXPONENTIAL DISTRIBUTION
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
*
NOPEX FUNCTION  RN1,D9      FOR THE NUMBER OF PE'S TO BE ALLOCATED.
0,0/.04,3/.09,4/.15,5/.25,6/.4,7/.55,8/.75,9/1.,10
*
* THE ABOVE FUNCTION IS USED TO CALCULATE THE PROBABILITY OF
* ALLOCATING THE NEXT GROUP OF PE'S, I.E. THE PROB. OF
* ALLOCATING 1 PE IS 0%, THE PROBABILITY OF ALLOCATING 3
* PE'S IS 4%, AND THE PROBABILITY OF ALLOCATING 4 PE'S IS
* 5% . . . ETC.
*
PESER FUNCTION  RN4,C2      FOR THE PE SERVICE TIME ASSIGNMENT
0,75/1.,135
*
* NOTE THAT ALL THE PE S WILL HAVE
* THE SAME SERVICE TIME.
CUSER FUNCTION  RN1,C2      FOR THE CONTRL. SERVICE TIME ASSIGN.
0,180/1.,220
INSTR FUNCTION  RN2,C2      FOR THE ASSIGNMENT OF THE NUMBER OF
0,50/1.,70
*
* INSTRUCTIONS TO THE JOB.
ENTRY VARIABLE  Q$RDY
EXPEC FVARIABLE (3*(FT$CUPE))  EXPECTED NUMBER IN THE SYSTEM
*
* EQUALS THE ARRIVAL RATE X AVG.
* TIME EACH XACT STAYS IN THE
* SYSTEM. DEVIDE THE RESULT BY
* 1000 FOR WE MULTIPLIED BY 1000.
EXINS FVARIABLE (3*(ST$SYSTEM))
THROU FVARIABLE (SR$CPU*(SC$CPU/ST$CPU))*100
INITIAL X$INST,0/X$CPUS,0  INITIALIZATION OF THE NUMBER
*
* OF INSTRUCTIONS IN EACH JOB
* AND NUMBER OF PE'S ALLOCATED
* TO EACH ONE.
STORAGE S$CPU,10  SPECIFY THE NUMBER OF PE'S IN
*
* TH SYSTEM.
SEGMENT 1 (ASSOCIATED WITH THE JOBS IN THE SYSTEM.)
*
GENERATE ,,,5  JOB ARRIVAL
QUEUE MEM

```

```

GATE LR   SYS      THIS GUARANTEES THAT ONLY ONE JOB
LOGIC S   SYS      IS ACTIVE.
DEPART    MEM
SAVEVALUE INST,FN$INSTR  ASSIGNMENT OF THE NUMBER OF
SAVEVALUE CPUS,FN$NOPE$ INSTRUCTIONS AND THE NUMBER OF
*
*          TERMINATE 1
*
*          SEGMENT 2
*
KEY        GENERATE 333,FN$XPDIS  GENERATE THE MACRO INSTRUCTIONS
ENTER     SYSTM
QUEUE     DUMMY
KEY2      TEST L    V$ENTRY,15
DEPART    DUMMY
ASSIGN    1,FN$CUSER  PARAMETERS ASSIGNMENTS FOR.CU
ASSIGN    2,FN$PESER  AND PE SERVICE TIMES.
QUEUE     RDY
GATE LR   NEXT
LOGIC S   NEXT
SEIZE     CUPE        SEIZE THE CONTROLLER
DEPART    RDY
ADVANCE   P1,FN$XPDIS  CONT. SERVICE TIME
RELEASE   CUPE
ENTER     CPU,X$CPUS   SEIZE THE SPECIFIED NO. OF PE'S
ADVANCE   P2
LEAVE     CPU,X$CPUS
SAVEVALUE INST-,1
TEST LE   X$INST,0,OUT
LOGIC R   SYS
OUT       LOGIC R   NEXT
LEAVE     SYSTM
SAVEVALUE THRPT,V$THROU
SAVEVALUE XACNO,V$EXPEC
TERMINATE

*
*          THE TIMER SEGMENT
*
*          GENERATE 10000
*          TERMINATE 1
*
*          THE CONTROL CARDS
*
*          START    5,NP
*          RESET
*          START    5
*
*          RMULT    11,33,55,77
*          CLEAR
*KEY2      TEST L    V$ENTRY,7
EXPEC     FVARIABLE 4*(FT$CUPE)
EXINS     FVARIABLE 4*(ST$SYSTM)

```



```

KEY   GENERATE  250 ,FN$XPDIS
      START    5
*
      RMULT    11,33,55,77
      CLEAR
EXPEC FVARIABLE  5*(FT$CUP)
EXINS FVARIABLE  5*(ST$SYSTEM)
KEY   GENERATE  200 ,FN$XPDIS
      START    5
*
      RMULT    11,33,55,77
      CLEAR
EXPEC FVARIABLE  6*(FT$CUPE)
EXINS FVARIABLE  6*(ST$SYSTEM)
KEY   GENERATE  166 ,FN$XPDIS
      START    5
*
      RMULT    11,33,55,77
      CLEAR
EXPEC FVARIABLE  7*(FT$CUPE)
EXINS FVARIABLE  7*(ST$CUPE)
KEY   GENERATE  153 ,FN$XPDIS
      START    5
*
      END

```

...

.

## Appendix F

### THE ANALYTIC PROGRAM FOR THE DATA FLOW MODEL OF CHAPTER VII

```

▽ DFLOW
[1]  DPF←4  A ONLY TOW DECIMAL PLACES
[2]  'U1 = '
[3]  (↑U1←0)  A SERVICE TIME FOR DF1
[4]  'U2 = '
[5]  (↑U2←0)  A SERVICE TIME FOR DF2
[6]  'U3 = '
[7]  (↑U3←0)  A SERVICE TIME FOR DF3
[8]  'ENTER Y3 (THE ARRIVAL RATE)'
[9]  (↑Y3←0)  A ARIVAL RATE
[10] 'ENTER N , NO. OF PE 'S'
[11] N←0
[12] 'ENTER K (THE SYSTEM CAPACITY) IT MUST BE GREATER THAN N'
[13] (↑K←0)  A SYSTEM CAPACITY, K MUST BE GREATER THAN N
[14] 'ENTER P'  A THE PROB. INST. BEING READY
[15] (↑P←0)
[16] 'ENTER Q'  A THE PROB. INST. BEING COMPLETED AND GET OUT
[17] (↑Q←0)
[18] Y1←U3XF
[19] Y2←NXU1
[20] A TO GET A FEELING FOR DF1,DF2, AND DF3 SEE THE
[21] A QUEUEING MODEL OF CHAPTER V.
[22] 'FOR DF3 ', 'U3=', (↑U3), ' AND Y3 = ', (↑Y3)
[23] '-----'
[24] L←1
[25] P3L←(0,1,1)F0
[26] TEMP←((U2+Y3)÷(U3X(F+Q)))
[27] P03←(1-TEMP)÷(1-(TEMPX(K+1)))
[28] BACK3;AP3L←(F03)X(TEMPX L)
[29] L←L+1
[30] P3L←P3L,[1](1,1,1)FAP3L
[31] →OUT3X\ (L>K)
[32] →BACK3
[33] OUT3: 'THE ABOVE P3L ARE FOR 0≤L≤K'
[34] 'AND P3L=0 OTHERWISE'
[35] 'P3L = '
[36] (5,K÷5)FAP3L
[37] 'F03 = ', (↑F03)
[38] 'FOR DF2 ', ' U2 = ', (↑U2), ' AND Y2=', (↑Y2)
[39] '-----'
[40] L←1

```

```

[41] F2L←(0,1,1)F0
[42] F02←(1-(Y2÷U2))÷(1-(Y2÷U2)×(K+1))
[43] BACK2:AF2L←((Y2÷U2)×L)×F02
[44] L←L+1
[45] F2L←F2L,[1](1,1,1)F AF2L
[46] →OUT2X\ (L>N)
[47] →BACK2
[48] OUT2:'THE ABOVE IS FOR 0< L ≤K'
[49] 'AND F2L = 0 OTHERWISE'
[50] 'F2L = '
[51] (5,N÷5)F AF2L
[52] 'F02 = ',(F02)
[53] 'FOR DF1 ', ' U1 = ',(U1), ' AND Y1=',(Y1), ' N= ',(N)
[54] '-----'
[55] R←(Y1÷(N×U1))
[56] XX←(1-(R×(K-N+1)))÷(1-R)
[57] J←0
[58] SUM1←0
[59] BACK1:SUM1←SUM1+(((Y1÷U1)×J)÷(!J))
[60] J←J+1
[61] →OUT1X\ (J≥N)
[62] →BACK1
[63] OUT1:F01←(1÷(((Y1÷U1)×N)÷(!N))×(XX))+SUM1)
[64] L←1
[65] F11L←(0,1,1)F0
[66] F12L←(0,1,1)F0
[67] BACK11:AF11L←F01×(Y1×L)÷(!L)×(U1×L)
[68] L←L+1
[69] F11L←F11L,[1](1,1,1)F AF11L
[70] →OUT11X\ (L)>(N-1))
[71] →BACK11
[72] OUT11:'F11L='
[73] AF11L
[74] 'THE ABOVE F11L ARE FOR 0 ≤ L ≤ N-1'
[75] L←N
[76] BACK12:AF12L←F01×(((Y1÷U1)×L)×(N×(N-L))÷!N))
[77] L←L+1
[78] F12L←F12L,[1](1,1,1)F AF12L
[79] →OUT12X\ (L)>K)
[80] →BACK12
[81] OUT12:'F12L = '
[82] AF12L
[83] 'F01 = ',(F01)
[84] 'THE ABOVE F12L ARE FOR N' ≤ L ≤ K'
[85] I←1
[86] SUM11L←0
[87] MORE:SUM11L←SUM11L+IX(F11L[I;1;1])
[88] I←I+1
[89] →OUT11LX\ (I)>(N-1))

```

```

[90]  →MORE
[91]  OUT11L:J+1
[92]  SUM12L←0
[93]  MORE2:SUM12L←SUM12L+(Jx(F12L[J;1;1]))
[94]  J←J+1
[95]  →OUT12Lx\ (J) (K-N))
[96]  →MORE2
[97]  OUT12L;SUML←SUM11L+SUM12L
[98]  FEUTIL←(SUML÷N)
[99]  'PE UTILI = ',(↑FEUTIL)
[100] UTIL1←(1-F01)÷N
[101] UTIL2←1-F02
[102] UTIL3←1-F03
[103] UTILSYS←(UTIL1+UTIL2+UTIL3)x100÷3
[104] 'UTIL1 = ',(↑UTIL1),' THIS IS THE PE'S UTILIZATION'
[105] 'UTIL2 = ',(↑UTIL2)
[106] 'UTIL3 = ',(↑UTIL3),' THIS IS THE MEMORY UTILIZATION'
[107] 'UTILSYS = ',(↑UTILSYS)
[108] 'U1 = ',(↑U1),' AND Y1 = ',(↑Y1),' INSTRUCTIONS/UNIT TIME'
[109] 'U2 = ',(↑U2),' AND Y2 = ',(↑Y2),' INSTRUCTIONS/UNIT TIME'
[110] 'U3 = ',(↑U3),' AND Y3 = ',(↑Y3),' INSTRUCTIONS/UNIT TIME'
[111] TPE←(NxU1)x(FEUTIL)
[112] 'THROU PE'S = ',(↑TPE)
[113] TSYS←((NxU1)+U2+U3)x(UTILSYS)÷100
[114] 'THROU SYS = ',(↑TSYS)
▽

```

## Appendix G

### THE SIMULATION PROGRAMS FOR BOTH MODELS OF THE DATA FLOW SYSTEM OF CHAPTER VII

```

SIMULATE
*****
*   THIS PROGRAM SIMULATES A DATA FLOW SYSTEM   *
*   IN THE SAME MANNER AS THE ANALYTIC MODEL DOES *
*   IT GIVES THE OVERALL PICTURE OF A DATA FLOW MACHINE *
*****
      RMULT      111
*   FUNCTIONS SPECIFICATION
XPDIS FUNCTION  RN1,C24  EXPONENTIAL DISTRIBUTION
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
*
SER1 FUNCTION   RN1,C2      FOR THE PE SERVICE TIME ALLOCATION
0,5/1.,15
  INITIAL       X$INST,0    INITIALIZATION OF THE NUMBER OF ISTRU-
  INITIAL       X$PPP,.3/X$QQQ,.4
  INITIAL       X$SER1,20/X$SER2,20/X$SER3,40
*
*                                     WHERE SER1 IS FOR THE PE'S
*                                     AND SER2 IS FOR THE RESPACKET
*                                     AND SER3 IS FOR THE MEMORY.
*
  STORAGE       S$PES,5    SPECIFY THE NUMBER OF PE'S IN THE
*                                     SYSTEM.
*
  THE MAIN PROGRAM SEGMENT
*
INST GENERATE   ...,10    GENERATE THE MACRO INSTRUCTIONS
  ENTER        SYSTM
  QUEUE        CONCP      THIS IS A CONCEPTUAL QUEUE
CAPCY TEST L    Q$MEMQ,40
  DEPART       CONCP
  ASSIGN       1,X$SER1    PARAMETERS ASSIGNMENTS FOR PE SERVICE
  ADVANCE      25,FN$XPDIS TIME. THIS REPRESENTS THE ARRIVAL RATE
  STAY         QUEUE      MEMQ    FOR THE MEMORY SUBSECTION
  SEIZE        MEM
  DEPART       MEMQ
  ADVANCE      X$SER3,FN$XPDIS
  RELEASE      MEM
  TRANSFER     .4,,QUEPE   % OF INST. THAT GO TO THE PE
*                                     I.E., THE READY INSTRUCTIONS.
  TRANSFER     .7,STAY,OUT
  QUEPE        QUEUE      PEQ    FOR THE PE SUBSECTION

```

```

ENTER      PES
DEPART     PEQ
ADVANCE    P1, FN$XPDIS
TEST L     Q$RESPQ, S$PES
LEAVE      PES
QUEUE      RESPQ
SEIZE      RES
DEPART     RESPQ
ADVANCE    X$SER2, FN$XPDIS
RELEASE    RES
TRANSFER   ,STAY
OUT SAVEVALUE DONE+, 1
LEAVE      SYSTM
TERMINATE  1

*
* THE TIMER
*
* GENERATE 1000
* TERMINATE 1
*
* THE CONTROL CARDS
*
START      10
RESET
INST GENERATE ,,,200
START      200
*
RMULT      111
CLEAR      X$SER1, X$SER2, X$SER3
STORAGE    S$PES, 10
START      200
*
RMULT      111
CLEAR      X$SER1, X$SER2, X$SER3
STORAGE    S$PES, 15
START      200
*
RMULT      111
CLEAR      X$SER1, X$SER2, X$SER3
STORAGE    S$PES, 20
START      200
*
RMULT      111
CLEAR      X$SER1, X$SER2, X$SER3
STORAGE    S$PES, 30
START      200
END

```

```

SIMULATE
*****
* PROGRAM NAME: (DFLOW SIMUL) THIS PROGRAM IS FOR DATA FLOW *
* PROGRAM SIMULATION. IT DIFFERES FROM THE PROGRAM DFLOW *
* ANALYTIC IN THE SENSE THAT IT SPECIFIES THE FINE DETAILS *
* OF EXECUTION I.E. IT IS BASED ON INDIVIDUAL INSTRUCTION *
* EXECUTION. *
*****
PESER FUNCTION RN1,C2
0,20/1.,30
STORAGE S$PROCS,1
CONTR VARIABLE Q$MEMQU-1
INITIAL X$OUT,0/X$STOR1,0/X$STOR2,0/X$STOR3,0/X$STOR4,0
INITIAL X$QUCHK,0/X$WRONG,0
INITIAL X1-X6,0
OUT EQU 20,X
STOR1 EQU 21,X
STOR2 EQU 22,X
STOR3 EQU 23,X
STOR4 EQU 24,X
WRONG EQU 25,X
QUCHK EQU 26,X
*
*
*
GENERATE ,,,1,,30 GENERATE AN INSTRUCTION
ASSIGN 1,1 THE CELL NUMBER OR ADDRESS
ASSIGN 2,1 THE INSTRUCTION NUMBER
ASSIGN 4, FN$PESER THE PROCESSING TIME IN THE PE
ASSIGN 5,2 THE NUMBER OF OPERANDS
ASSIGN 6,2 THE OPERAND COUNTER
ASSIGN 8,2 THE ADDRESS OF 1ST OPND IN THIS INSTR CEL
ASSIGN 9,3 THE ADDRESS OF 2ND OPND IN THIS INSTR CEL
ASSIGN 11,20 THE 1ST DEST. ADDRESS
ASSIGN 12,10 THE 2ND DEST. ADDRESS
ASSIGN 16,20 THE 3RD DEST. ADDRESS
* PRIORITY P6 THE PRIORITY IS ASSIGNE IN ACCORDANCE
* TRANSFER ,MEMR1
*
* WITH THE NUMBER OF OPERANDS IN THE
* OPERAND COUNTER.
* EACH GROUP HEREAFTER WILL RESEMBLE THE ABOVE (I.E., IT WILL
* HAVE THE SAME NUMBER OF PARAMETERS BUT WITH DIFFERENT VALUES)
* AND EACH NODE IN THE DATA FLOW GRAPH WILL HAVE A DISTINCTIVE
* SET OF VALUES.
*
GENERATE ,,,1,,30
ASSIGN 1,4
ASSIGN 2,2
ASSIGN 4, FN$PESER
ASSIGN 5,2
ASSIGN 6,2
ASSIGN 8,5

```

```

ASSIGN      9,6
ASSIGN     11,21
*          ASSIGN     12,15
*          ASSIGN     13,18
*          ASSIGN     16,27
*          PRIORITY   P6
*          TRANSFER   ,MEMR1
*
*
GENERATE    ,,,1,,30
ASSIGN     1,7
ASSIGN     2,3
ASSIGN     4,FN$PESER
ASSIGN     5,2
ASSIGN     6,2
ASSIGN     8,8
ASSIGN     9,9
ASSIGN    11,23
*          PRIORITY   P6
*          TRANSFER   ,MEMR1
*
*
GENERATE    ,,,1,,30
ASSIGN     1,10
ASSIGN     2,4
ASSIGN     4,FN$PESER
ASSIGN     5,2
ASSIGN     6,2
ASSIGN     8,11
ASSIGN     9,12
ASSIGN    11,24
ASSIGN    12,26
*          PRIORITY   P6
*          TRANSFER   ,MEMR1
*
*
GENERATE    ,,,1,,30
ASSIGN     1,13
ASSIGN     2,5
ASSIGN     4,FN$PESER
ASSIGN     5,2
ASSIGN     6,2
ASSIGN     8,14
ASSIGN     9,15
ASSIGN    11,27
*          PRIORITY   P6
*          TRANSFER   ,MEMR1
*
*
GENERATE    ,,,1,,30
ASSIGN     1,16
ASSIGN     2,6
ASSIGN     4,FN$PESER
ASSIGN     5,1

```



```

ASSIGN      6,1
ASSIGN      8,17
ASSIGN      9,18
ASSIGN      11,36
* PRIORITY  P6
* TRANSFER  ,MEMR1

GENERATE    ,,,1,,30
ASSIGN      1,19
ASSIGN      2,7
ASSIGN      4, FN$PESER
ASSIGN      5,2
ASSIGN      6,0
ASSIGN      8,20
ASSIGN      9,21
ASSIGN      11,29
ASSIGN      12,32
* TRANSFER  ,MEMR1

GENERATE    ,,,1,,30
ASSIGN      1,22
ASSIGN      2,8
ASSIGN      4, FN$PESER
ASSIGN      5,2
ASSIGN      6,0
ASSIGN      8,23
ASSIGN      9,24
ASSIGN      11,30
ASSIGN      12,35
* TRANSFER  ,MEMR1

*
*
GENERATE    ,,,1,,30
ASSIGN      1,25
ASSIGN      2,9
ASSIGN      4, FN$PESER
ASSIGN      5,2
ASSIGN      6,0
ASSIGN      8,26
ASSIGN      9,27
ASSIGN      11,33
* ASSIGN    12,32
* TRANSFER  ,MEMR1

GENERATE    ,,,1,,30
ASSIGN      1,28
ASSIGN      2,10
ASSIGN      4, FN$PESER
ASSIGN      5,2
ASSIGN      6,0
ASSIGN      8,29
ASSIGN      9,30

```

```

      ASSIGN      11,38
*     ASSIGN      12,32
      ASSIGN      16,50
*     TRANSFER    ,MEMR1

      GENERATE    ,,,1,,30
      ASSIGN      1,31
      ASSIGN      2,11
      ASSIGN      4,FN$PESER
      ASSIGN      5,2
      ASSIGN      6,0
      ASSIGN      8,32
      ASSIGN      9,33
      ASSIGN      11,39
*     ASSIGN      12,32
*     TRANSFER    ,MEMR1

      GENERATE    ,,,1,,30
      ASSIGN      1,34
      ASSIGN      2,12
      ASSIGN      5,2
      ASSIGN      6,0
      ASSIGN      8,35
      ASSIGN      9,36
      ASSIGN      11,42
*     ASSIGN      12,32
*     TRANSFER    ,MEMR1

      GENERATE    ,,,1,,30
      ASSIGN      1,37
      ASSIGN      2,13
      ASSIGN      4,FN$PESER
      ASSIGN      5,2
      ASSIGN      6,0
      ASSIGN      8,38
      ASSIGN      9,39
      ASSIGN      11,41
*     ASSIGN      12,32
*     ASSIGN      16,50
*     TRANSFER    ,MEMR1

      GENERATE    ,,,1,,30
      ASSIGN      1,40
      ASSIGN      2,14
      ASSIGN      4,FN$PESER
      ASSIGN      5,2
      ASSIGN      6,0
      ASSIGN      8,41
      ASSIGN      9,42
*     ASSIGN      11,29
*     ASSIGN      12,32
      ASSIGN      16,50

```

```

TRANSFER      ,MEMR1
*
MEMR1 ENTER    SYSTEM
LOGIC R       LOCK
LOGIC R       NEXIN
MEMRY TEST E  P6,P5,CHECK  TEST THE OPERAND COUNTER TO SEE
*                                     IF IT HAS THE REQUIRED NUMBER
*                                     OF OPERANDS, IF YES PASS THE
*                                     INSTRUCTION TO THE PE QUEUE.
*
EXEC  QUEUE    OUQUE
ENTER  PROCS
DEPART OUQUE
ADVANCE P4      THE PROCESSOR AT WORK
LEAVE  PROCS
QUEUE  DISQU
GATE LR NEXIN
LOGIC S NEXIN
DEPART DISQU
ADVANCE 10
SAVEVALUE STOR1,P11
SAVEVALUE STOR2,P12
SAVEVALUE STOR3,P13
SAVEVALUE STOR4,P14
SAVEVALUE QUCHK,Q$MEMQU
* TEST G      X$QUCHK,0,NOMEM
LOGIC S LOCK
TRANSFER ,OUTST
*NOMEM LOGIC R NEXIN
OUTST TEST E  P16,0,OUTPT  CHECKING FOR AN OUTPUT INSTRUCTION
LEAVE SYSTEM
TERMINATE 1
OUTPT SAVEVALUE P2,P16
SAVEVALUE OUT+,1
LEAVE SYSTEM
TERMINATE 1
*
* THE FOLLOWING SEGMENT REPRESENT THE MEMORY. AS SOON
* AS AN INSTRUCTION IS EXECUTED IN THE PE THE MEMORY
* STARTS CHECKING IF THE RESULT SHOULD BE ASSIGNED TO
* ANY OTHER INSTRUCTIONS AND DECREMENTING THE OPERAND
* COUNTER AS A RESULT
*
CHECK QUEUE    MEMQU
* TEST LE     Q$DUMY,0,MORE
LOGIC R       CONCP
* GATE LS     LOCK
SAVEVALUE    TEMP,Q$MEMQU
SEIZE        MEM
ADVANCE      5
* ASSIGN     7+,1      FOR THE CASE THAT THE INSTRUCTION
*                                     LOOPS.THIS SEGMENT WILL TRY TO CATCH IT

```

```

*      TEST LE      P7,30,EXIT
      TEST E      P8,X$STOR1,NEXT1
      ASSIGN      6+,1
NEXT1  TEST E      P8,X$STOR2,NEXT4
      ASSIGN      6+,1
*NEXT2 TEST E      P8,X$STOR3,NEXT3 THIS AND THE NEXT TEST ARE FOR THE
*      ASSIGN      6+,1                3RD AND 4TH DESTINATION ADDRESSES
*NEXT3 TEST E      P8,X$STOR4,NEXT4 IF THEY EXIST.
*      ASSIGN      6+,1
NEXT4  TEST E      P9,X$STOR1,NEXT5
      ASSIGN      6+,1
NEXT5  TEST E      P9,X$STOR2,NEXTF
      ASSIGN      6+,1
*NEXT6 TEST E      P9,X$STOR3,NEXT7
*      ASSIGN      6+,1
*NEXT7 TEST E      P9,X$STOR4,NEXTF
*      ASSIGN      6+,1
NEXTF  RELEASE    MEM
*      SAVEVALUE   TEMP-,1
      TEST LE      V$CONTR,0,REPET
      LOGIC R      LOCK
      LOGIC R      NEXIN
      LOGIC S      CONCP
REPET  DEPART      MEMQU
      TEST E      P5,P6,WAIT
      TRANSFER     ,EXEC
WAIT   QUEUE      DUMY
      GATE LS      CONCP
      DEPART      DUMY
      TEST LE      Q$DUMY,0,EMPTY
      LOGIC R      CONCP
EMPTY  TRANSFER     ,CHECK
EXIT  SAVEVALUE   WRONG,P2
      TERMINATE   1          WHERE P2 IS THE INSTRUCTION NUMBER

*
*      THE TIMER SEGMENT
*
*      GENERATE    1000
*      TERMINATE   1
*
*      CONTROL CARDS
*
*      START      14          THIS SHOULD EQUAL TO THE NUMBER
*                                OF INSTRUCTIONS IN THE SYSTEM.
      STORAGE     S$PROCS,4  FOR THE SECOND RUN
      CLEAR
      START      14
*
*      STORAGE     S$PROCS,7  FOR THE THIRD RUN
      CLEAR
      START      14
*

```

```
STORAGE S$PROCS,10 FOR THE FORTH RUN
CLEAR
START 14
*
STORAGE S$PROCS,13 FOR THE FIFTH RUN
CLEAR
START 14
*
STORAGE S$PROCS,20 FOR THE SIXTH RUN
CLEAR
START 14
*
END
```