A MICROPROCESSOR BASED HIGH SPEED PACKET SWITCH FOR SATELLITE COMMUNICATIONS

(NASA-CR-163357) A MICROPROCESSOR BASED HIGH SPEED PACKET SWITCH FOR SATELLITE COMMUNICATIONS Final Report, 15 Apr. 1978 - 30 May 1980 (Clarkson Coll. of Technology) 347 p HC A15/MF A01 CSCL 17B G3/32

N80-27557

Unclas 28062

Prepared for

National Aeronautics & Space Administration
Lewis Research Center
21000 Brookpark Road
Cleveland, Ohio 44135

Final Report

on

Grant No. NSG-3191
James Rotnem - Project Officer
April 15, 1978 - May 30, 1980

Mohammed Arozullah - Principal Investigator
Stephen C. Crist - Co-Investigator

Grant Title: Design of a Microprocessor-Based High Speed Space Borne Message Switch.

CLARKSON COLLEGE OF TECHNOLOGY
POTSDAM, NEW YORK 13676

ABSTRACT

This report is concerned with design and evaluation of a microprocessor based high speed space-borne packet switch. Three designs namely, a single, three and multiple processor designs are presented. System architectures for these three designs are presented. Further, the hardware circuits, and software routines required for implementation of the three and multiple processor designs are also presented. A bit-slice microprocessor is used. This processor has been designed and microprogrammed. Maximum throughput has been calculated for all three designs. Queue theoretic models for these three designs have been developed and utilized to obtain analytical expressions for the average waiting times, overall average response times and average queue sizes. From these expressions graphs have been obtained showing the effect on the system performance of a number of design parameters.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

In the past decade packet switching has revolutionalized data communication. In 1968 virtually all interactive data communication networks used circuit switching, which is the current technology used in telephone networks [1]. Circuit switching networks preallocate channel bandwidth for an entire message. However, since most interactive data traffic occurs in short bursts, a large portion of the bandwidth is wasted. Thus, as digital electronics became inexpensive and the need for more digital data communication networks grew as computer technology expanded, the redesign of data communication networks became economically feasible and desirable. Packet switching was introduced since it allows for the dynamic allocation of bandwidth, which permits users to share the same transmission line previously assigned to only one user. Packet switching has improved the economics of data communication systems, network reliability and functional flexibility [1].

Packet switching networks divide the users' messages into small segments, or packets, of data which move through the network towards their destination. All packets are fixed-length and serial in structure. Packets consist of a header and a body. The header, which precedes the body, contains the routing control information which indicates the packet's source and destination. In addition, the header also contains message reconstruction information for use at the destination. Since a complete message may occupy more than one

packet, each header contains a message number and a packet sequence number. Thus, any packets arriving in a scrambled sequence can be rearranged to correctly yield the entire message received. The body of a packet contains the data being transmitted. The length of each packet within a network is fixed for the entire system.

The routing of these packets is handled by the packet switches implemented in the network. These special switches replace the previous circuit switches found in telephone networks and older data communication networks. The scope of the work presented in the following chapters consists of the design and evaluation of these packet switches using microprocessors to control the switching functions.

1.1 Problem Definition

This report examines the problem of designing and evaluating multiprocessor-controlled packet switches. (The design and evaluation of a single processor version is presented in [2,3].) The work presented in the following chapters will investigate the question of how large a multiprocessor packet switch can be constructed before the problem of resource contention erodes the system's performance. The performance of these multiprocessor designs will be evaluated in terms of their maximum throughput with respect to the number of users and the number of processors implemented, average delay within the switch, and queue sizes.

These packet switches must be capable of routing packets among any number of up to several hundred users. In addition,

all designs must allow the use of these packet switches in communication satellites as well as in networks using only land lines. The problems of protocols and error-correction codes are briefly reviewed in this work.

1.2 Approach to the Problem

System design considerations are examined first. These considerations include protocols, prior work, workload divisions and resource contention among processors. A review of protocols and their effects on throughput is presented. Using the information from this investigation of protocols, a decision is made on how to handle this problem.

After the protocol problem is solved, a review of the prior single processor design is presented. Using the prior design as a foundation, the requirements and goals of the multiprocessor designs are formulated. A review of the prior design at the functional level allows the workload division for the three processor design to be made.

Once the workload division is made, the contention problems relating to the shared resources are investigated. In this investigation, each shared resource is identified and their specific contention problems are examined. Various solutions to these problems are found and presented.

Once all the design considerations that influence the actual implementation are examined, the system architecture of the three processor packet switch is designed. The design of the architecture, its operation and functional requirements

allows the detailed design of the system hardware to be completed.

Once the system hardware is designed, the processors and their software requirements are defined and designed in detail.

After the design of the three processor system is completed, the same design procedure is repeated for the design of the multiple processor packet switch.

With both designs complete, an evaluation of each system is carried out. The evaluation determines the maximum throughput of both architectures. A queue theoretic model is developed that facilitates analysis of delay and queue sizes within the packet switch.

2.0 SYSTEM DESIGN CONSIDERATIONS

The final architectural designs of the multiprocessor-based packet switches are influenced by several system design constraints and goals. Some of these are considered in the single processor architecture [2,3]. Thus, those particular considerations will be reviewed briefly in this chapter. The remaining design considerations arose directly from the use of multiple microprocessors, and shall be discussed in detail. The review of each design constraint and design goal will lend an explanation to the approach taken in the development of the new system architectures.

2.1 Protocols

Much attention was given to the analysis of various protocols and their effects on the packet switch in the previous work [2 , 3]. Implementation of a full forward error correction (FEC) scheme, an End-to-End Automatic-Repeat-Request (ARQ) scheme, and an Up-Link ARQ scheme were considered. The results of this research were used to select a protocol scheme for the multiprocessor architectures.

Since large system throughput is a major goal, any protocol which was shown to reduce system throughput was eliminated from further consideration.

A reduction in throughput was found to be linked to all protocols requiring the packet switch to maintain special software. Thus, only protocols which are transparent to the packet switch will be supported by the multiprocessor

architectures. Two protocols which fulfill this requirement are the FEC scheme and the End-to-End ARQ scheme.

In addition to improving system throughput, "transparent" protocols offer the users flexibility. Users can custom tailor protocols to meet their needs. Transparent protocols could be changed or altered even after the network is completed and operational. Also, different protocols could be implemented between different users in the same network.

2.2 Packet Construction

The packet format consists of a body and a header. Packets are serial in structure with the header preceding the body. The body length of a packet is fixed for a given system. However, the selection of this length is generally made from a range of 256 bits up to 10240 bits. In order to maximize the throughput of the multiprocessor-based packet switches under investigation in this report, the recommended body length is 10240 bits.

The packet header contains information required to route the packets to their proper destinations. In addition, the header also contains special information needed by the destination. Since entire messages may exceed the length of a single packet, they must be divided into packet-length segments before transmission. The last packet of a message will be "padded" with blank characters to fill unused bits in the packet should the message not require an integer number of packets. The special header information is used by the destination

to reconstruct entire messages which have been sent via several packets. Therefore, should the packets arrive in a scrambled sequence, the message is still recoverable. This information and the routing information is arranged in various fields. These fields contain, in coded form, the packet's source, destination, message number and sequence number.

Since the header information is vital for proper packet transmission, its protection is a system design requirement. Thus, the header is protected by an error-correcting code. The Bose-Charedhuri-Horquenghen (BCH) code was chosen for this task in the original design and is implemented again in the multiprocessor systems. The cost of this protection is increased hardware and software for both the system users and the packet switch. However, this increased overhead has been deemed necessary in order to maintain the integrity of the network. The header is the only part of the packet which has error-correction protection that is used directly in conjunction with the packet switch. Error protection of the packet body is optional to system users and must be implemented at the ground stations.

2.3 The Prior Architecture

There are several important design philosophies which have shaped the architecture of the packet switch. They are incorporated in the multiprocessor architectures as well as in the single processor architecture. The following are the design guidelines used for all architectures:

1) A fixed packet length must be used by the network. This simplifies hardware and software requirements.
2) All packet transfers through the switch are done serially. This eliminates any need for Serial-to-Parallel and Parallel-to-Serial conversions. (Although all packet transfers are done serially, the processor accesses the header in parallel.)
3) Since all packet transfers are serial, this operation is to be managed by dedicated hardware. Processor control of this function would decrease system throughput due to the comparatively slow speed of software. In addition, the use of dedicated hardware to perform this task allows the processor to spend more time making decisions and controlling other system operations.
4) The full capacity of the processor must be utilized to avoid throughput reduction. This goal is achieved by requiring that the processor never wait for hardware. This requires parallel hardware for certain functional blocks. These blocks are initiated into action by the software. This hardware completes its assigned task automatically without further software supervision. All architectures permit several simultaneous operations to be performed, since the processor is free to move on to new tasks once the hardware is activated.

5) To further increase system throughput, the processor is only allowed to access the header of each packet. While in the switch, the packet bodies are left untouched by the processor. Since the routing information needed by the processor is found only in the header, this design goal is easy to implement.

The final system architecture of the single processor packet switch is presented in Figure 2.1. This packet switch handles N users who are allocated one line each. Operation of the system consists of each user transmitting their packets to the switch which routes the packets to the proper destination. The packets arrive at the switch as serial bit streams. The switch is configured such that any user may communicate with any other user in the network.

The routing of the users' messages begins with the buffering of all incoming packets. Each input line is double buffered. Even with double buffering, the processor service response time must be short. Buffer overflow will destroy packets left too long in a buffer. In order to avoid packet losses, a minimum of processing is done at the input buffers. As soon as a full buffer is detected, the processor immediately stores the packet in temporary storage. This storage area is constructed of shift registers arranged in an array.

Once stored in the shift register array, each packet receives additional service. Their headers are decoded by the processor to determine each packet's destination. The routed packets are assigned to software output queues. Use

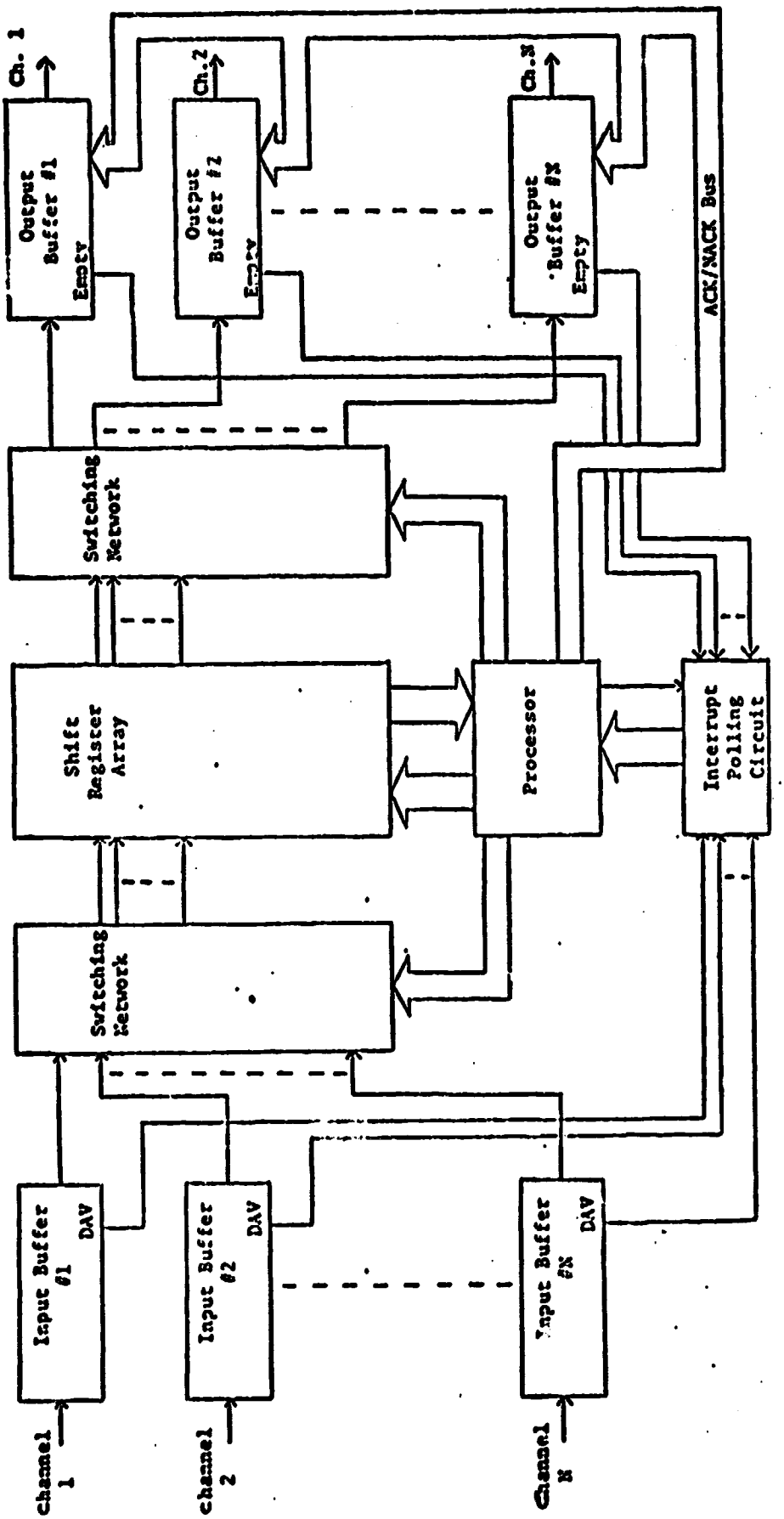


Fig. 2.1 Single Processor System Architecture
(Courtesy of James Burnell)

of software queues eliminates the need for additional packet transfers required by hardware queues. Each queue corresponds to one unique output buffer.

When an output buffer becomes empty, the processor accesses the associated queue for the next packet awaiting transmission. Each queue contains the location of each routed packet in the array awaiting transmission to that queue's corresponding output buffer. Using this information, the processor begins the transfer of the queue's oldest packet to the proper buffer. Once in the buffer, the packet is then transmitted onto the network channel under hardware control.

The software required to control the packet switch consists of three routines: The input service routine, the background service routine and the output service routine.

The input service routine is interrupt driven. Execution of this routine begins when the Data Available (DAV) line of an input buffer becomes active and is detected by the input interrupt polling circuit. Equal priority among all users is ensured by the sequential scanning of these DAV lines.

The first task of this software is the linking of a free data path in the input switching network to the full buffer. Next, the address of an empty shift register is fetched from the Empty Shift Register List (ELIST). This shift register is then linked to the full buffer via the data path. Finally, the processor initiates the packet's transfer into the array. This routine has the highest priority and is uninterruptable.

The background service routine continually scans the shift register array in search of packets requiring service. Upon finding one, the processor fetches the header. The header is corrected, if necessary, by using error pattern data stored in a Syndrome Decoder ROM. Next, the packet's destination is determined. The packet's address in the array is then placed in the proper output queue list. However, if this list is empty and its corresponding buffer is also empty, the processor will load the packet directly into the buffer. The packet's array address will then be placed in ELIST. This routine has the lowest priority since it is not interrupt driven.

Like the input routine, the output service routine is interrupt driven. Detection of an empty output buffer by the output interrupt polling circuit forces the execution of this software package. This routine must first check the output queue associated with the buffer requesting service. If this list is empty, the service request "flag" for this particular buffer is reset and the processor exits from this routine. However, if the queue is not empty, the processor then fetches the array address of the oldest packet in the queue. Using this address, the processor then links the proper shift register to the empty buffer. This link is established via an available data path in the output switching network. Once the link is complete, the data transfer begins. This routine has the second highest priority.

2.4 Processor Workload Divisions

In most multi-microprocessor systems, the primary design goal is the identification and separation of all tasks which are relatively independent [4]. Ideally, this allows each processor to perform a dedicated task. Thus, each processor can operate mostly independently of the others. As a result, very little data needs to be exchanged among processors relative to the total system data flow.

This design philosophy is implemented in the determination of the processor workload division for the multiprocessor-based packet switches. The first step in the implementation is the identification of each "independent" task. A review of the single processor design shows that the operation of the packet switch consists of three major tasks. Each of these are controlled by independent software routines. The three tasks are:

1) Storage of received packets (Input Function)
2) Routing of each received packet (Routing or Background Function)
3) Transmission of each routed packet (Output Function)

Now that the "independent" tasks have been identified, the workload division can be made; one processor is assigned to each of the three tasks. The architecture supports an Input Processor, a Routing Processor and an Output Processor. Each processor supervises dedicated hardware, executes custom software and shares a minimum amount of common resources.

Resource sharing presents many problems and is the next topic of discussion.

2.5 Resource Contention Among Processors

In most multiprocessor systems, shared resources are necessary. Unfortunately, they present many control problems and may cause reduced throughput. Therefore, they must be kept to a minimum.

Concern over shared resources arises whenever the possibility of processor contention exists. Contention occurs when two or more processors simultaneously request access to the same resource. This is known as a race condition [5]. Contention also occurs when one or more processors request access to a resource currently in use by another processor.

A system's throughput can be severely reduced by contention in two ways. Simultaneous access of a resource by two or more processors will cause havoc in the system. Therefore, special hardware and/or software is required to schedule resource allocation. Only one processor must be granted access to a particular resource at any given time. This requires that the other processors be "locked out." Implementation of any resource locking scheme requiring special system software will reduce throughput. In addition, processors which become "locked out" are forced to wait for the busy resource. Processor idleness due to contention reduces throughput.

Since increased throughput is the primary goal in the design of a multiprocessor packet switch, contention must be

minimized. This goal is achieved by first identifying each shared resource. The following is a list of shared resources compiled from a review of the system architecture:

1) The shift register array
2) ELIST
3) The output queue lists
4) The Output switching network
5) The output buffers

An analysis of contention problems for each of these resources is now needed.

All three processors use the shift register array. Each shift register must assume one of the following states:

1) Empty
2) Holding an unserviced packet
3) Holding a routed packet
4) Shifting out or in a packet in transit

Empty shift registers with their addresses in ELIST can only be accessed by the Input Processor. Shift registers containing unserviced packets can only be accessed by the Routing Processor. The Output Processor can only service shift registers containing routed packets. Thus, any shift register in one of these three states is free from contention problems.

However, shift registers containing packets in transit from the array to the output buffers present a contention problem. As stated earlier, once a packet transfer is

initiated by a processor, dedicated hardware takes control. Therefore, the processor is now free to start a new task. In the case of the output processor, the next task is the updating of ELIST with the address of the packet in transit. ELIST now contains the address of a shift register who's contents are only partially transferred. A resource contention could occur if the Input Processor uses this location to store a new packet.

Two solutions to this problem exist. One solution is to require the Output Processor to temporarily hold the address of each packet in transit. This scheme needs hardware to signal the completion of transfers, address storage and additional control software. Since additional software reduces throughput, this scheme is not used.

Instead, the scheme used requires the array hardware to allow the simultaneous transmission of an old packet and the storage of a new packet at the same location. Although the shift register array is a shared resource, contention problems have been avoided.

The shared resources remaining to be examined all have one thing in common: Each resource is accessed by the Routing Processor. However, only the output queue lists are accessed by this processor under normal operation. The Routing Processor only requires access to ELIST, the output buffers and the output switching network when a special event occurs. This event takes place whenever the Routing Processor finds a packet destined to an output buffer which is empty and who's

output queue list is also empty. The Routing Processor responds by transmitting the packet directly.

In order to deal with this one special operation, allocation of many shared resources is required. This increases the risk of reduced throughput due to contention. In addition, throughput will be reduced by the system software required to manage the resource allocations. Therefore, a decision must be made whether or not to allow the Routing Processor to transmit packets as done previously by the background routine in the single processor design.

Since system throughput is at stake, the Routing Processor must not be permitted to transmit packets. Although a new scheme must be devised to handle this special event, contention has been completely eliminated from the output buffer system and the output switching network. (Specific details on the new scheme are presented next in the contention analysis of the Output Queue Lists.) These resources are now solely controlled by the Output Processor. In addition, ELIST is now only accessed by the Input Processor and the Output Processor.

Each output queue list is associated with one unique output buffer. These lists contain the addresses of routed packets in the array awaiting transmission. The Routing Processor must access these lists to update them with the addresses of newly routed packets. Meanwhile, the Output Processor must access the lists to find the next packet requiring transmission.

Since the Routing Processor always writes to the lists while the Output Processor always reads from the lists, dual

port RAM's can be used [6]. Dual port RAM's permit two processors to access them simultaneously provided at least one processor performs a read operation. Only one processor is allowed to perform a write operation at one time.

The data structure of the output queue lists is designed such that if the processors are accessing the same location the queue is considered empty. Only after the Routing Processor updates the list and moves on to the next location can the output processor read from the once empty list. Thus, the situation of a concurrent read and write operation at a single location is avoided. At first glance, the problem of contention appears to be solved. However, further investigation is needed to ensure that this is true.

A new output buffer status word with three states, "busy," "empty," and "idle" is now used. An output buffer in the busy state is in the process of receiving a packet from the array, receiving output processor service or transmitting a packet. Once a packet is transmitted, the buffer enters the empty state which indicates the buffer requires service. An output buffer is placed in the idle state by the Output Processor when the buffer becomes empty if its queue list is also empty.

When the Routing Processor encounters a packet destined for an idle buffer, it must first update the buffer's queue list. Then the processor must change the buffer's status word to indicate the buffer is empty and requires service from the output processor. This operation replaces the prior scheme of transmitting packets directly. As stated earlier,

many contention problems are eliminated by this new scheme. However, a new subtle problem has arisen.

Table 2.1 lists the sequence of events which leads to the problem. The first line in the table shows that an empty output buffer is receiving processor service. The buffer's output queue is currently empty. The Output Processor is presently accessing this queue. Meanwhile, a packet destined for this buffer is being routed by the Routing Processor. The Routing Processor has just read the status word of this buffer which indicates the buffer is empty. However, just after the status word was read, the Output Processor updated it to correctly indicate that the buffer is idle. Line two in the table now shows the new status word. In addition, the Routing Processor, acting on incorrect information, has placed the packet's address into the queue list without updating the status word. Line three in the table displays the packet's address residing in the queue list while the status word still indicates the buffer is idle. Since the Output Processor can only service buffers in the empty state, this packet is trapped in the system. This packet will remain trapped until a new packet arrives for the same destination. The remaining lines in the table depict the events leading to the recovery of the trapped packet. Since the recovery time may be quite long, a solution to this problem must be found.

In order to solve this problem, a locking scheme is implemented for the output queue lists and the associated output buffer status words. Any time one processor gains access to

EVENTS	OUTPUT QUEUE LIST	OUTPUT BUFFER	OUTPUT STATUS WORD	ROUTED PACKETS	STATUS READ BY ROUTING PROCESSOR
1	Empty	Empty	Empty→Idle	one Arrives (A)	Empty
2	Not Empty	Empty	Idle	(A) Enqueued	-
3	Not Empty	Empty	Idle	(A) Trapped	-
4	Not Empty	Empty	Idle	(B) Arrives	Idle
5	Not Empty	Empty	Idle→Empty	(A)+(B) Enqueued	-
6	Not Empty	Full	Empty→Busy	(A) Sent, (B) Waits	-
7	Not Empty	Empty	Busy→Empty	(B) Waits	-
8	Empty	Full	Empty→Busy	(B) Sent	-
9	Empty	Empty	Busy→Empty	-	-
10	Empty	Empty	Idle	-	-

Table 2.1 Contention Problems in a Shared Flag System

one of the queue lists, the other processor is locked out from that particular queue and its associated buffer status word. Thus, when a processor is granted access to these resources, the processor is ensured of obtaining correct data. Although the queue lists are never accessed by more than one processor at one time, the dual port RAM's are still used since they simplify other hardware and software requirements. The locking scheme requires some additional hardware and software. Some processor idleness may also be encountered. However, although some system throughput is sacrificed, the packet switch's integrity has been preserved.

The ELIST is the last shared resource to be analyzed for contention problems. ELIST is shared by both the Input Processor and the Output Processor. The Input Processor must access this list to find available empty shift registers in the array. The Output Processor updates this list with the addresses of shift registers released by transmitted packets. As in the case of the Output Queue Lists, one processor always writes to ELIST while the other always reads from ELIST. Therefore, ELIST can be built using dual port RAM's. These RAM's allow a simultaneous read operation and write operation to take place at different locations without interference.

The data structure of ELIST is designed such that no simultaneous read and write operations can be performed at the same location unless the list becomes empty. If the list becomes empty, the system faces a far graver problem than contention. However, ELIST should never become empty under

normal operation. Thus, another shared resource is spared from contention problems, since the processors using it are transparent to one another.

In summary, the output queue lists are the only shared resources which face contention problems. Details concerning how this problem is handled are found in 3.1.5, 3.3.2, and 3.3.3.

3.0 THE THREE PROCESSOR DESIGN

The system architecture of the three processor packet switch is presented in Figure 3.1. As in the single processor design, this packet switch handles N users who are allocated one line each. Again, the switch is configured such that any user may communicate with any other user in the network. Although the workload is divided among three processors, the function of the switch remains unchanged from the original design. Thus, a detailed description of the packet switch's operation is not presented. Instead, this chapter focuses on the actual hardware, software and processors required to implement this new architecture.

3.1 System Hardware

Under processor control, the system hardware carries out the assigned tasks of the packet switch. Since the packet switch architecture now supports multiple processors, a new control signal labelling scheme has to be adopted. This scheme is designed to help eliminate any confusion regarding the source and destination of each control signal. Table 3.1 contains each control signal code format with an example and an explanation. Detailed explanations of the circuits and their operations are presented in the following sections.

3.1.1 The Input Buffers

Displayed in Figure 3.2 is the circuitry required by an input buffer for one user. All received packets remain in

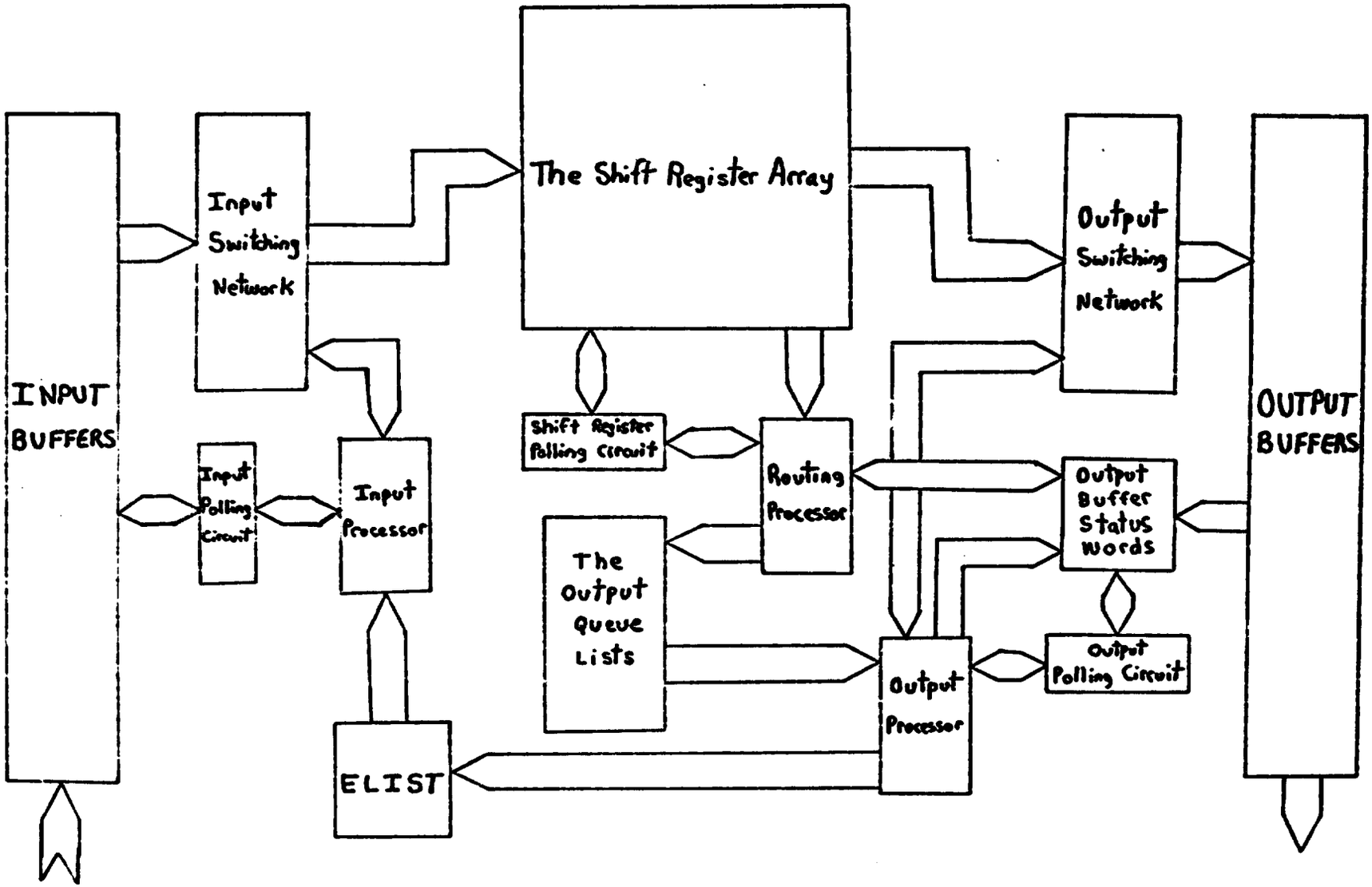


Fig. 3.1 Three Processor System Architecture

Signal Code	Format	Example	Source	Destination
Source/Destination Letter Code	A - Signal Name B - Signal Name C - Signal Name	A-RESET B-REQUEST C-RELEASE	Input Processor µW Routing Processor µW Output Processor µW	System Hardware
	Signal Name - A Signal Name - B Signal Name - C	IBSR - A STATUS-B EMPTY-C	System Hardware	Input Processor Imp Logic Routing Processor Imp Logic Output Processor Imp Logic
µP Address Code	µP# - A µP# - B µP# - C	µP1 - A µP1 - B µP1 - C	Input Processor IEU Address Decoder Routing Processor IEU Address Decoder Output Processor IEU Address Decoder	System Hardware
Matrix Address Code	M# - A M# - B	M1 - A M1 - B	Input Processor IEU Matrix Address Output Processor IEU Matrix Address	System Hardware
Instruction Signal Code	Signal Type	COUNT FIRE	System Hardware	System Hardware

Table 3.1 Hardware Control Signal Codes

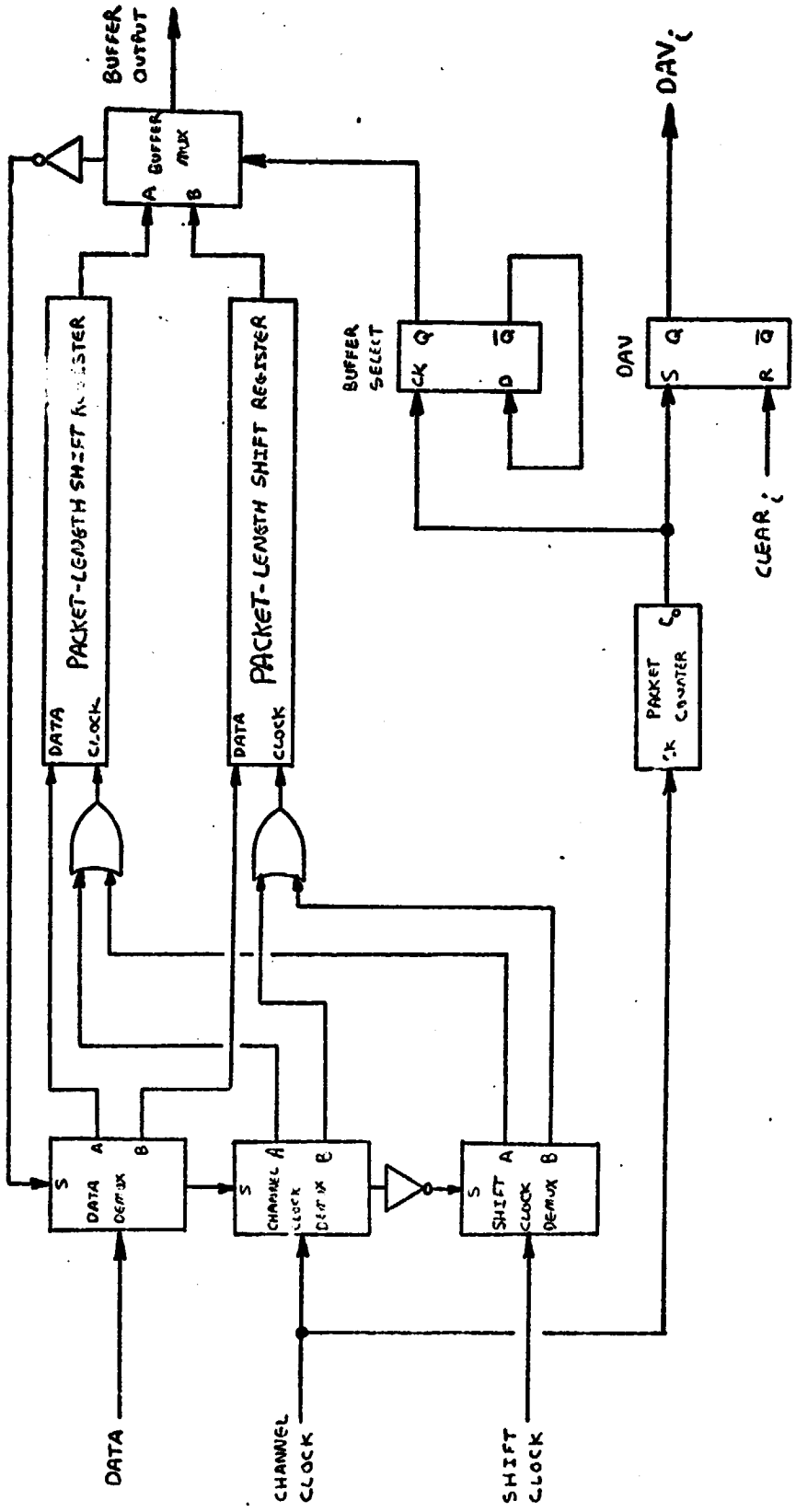


Fig. 3.2 Input Buffer for One User

the input buffer until they are transferred into the shift register array. In order to reduce the possibility of overflow, each input channel is double buffered. The two buffers at each input channel are packet-length shift registers. Buffer select logic determines which buffer is to be linked to the input channel while the other buffer is linked to the Input Switching Network. This select logic is driven by a packet counter. The packet counter monitors the arrival of packets by counting each bit. Once an entire packet has been received, the counter rolls over, activating the buffer select hardware. The select logic then switches the buffer assignments. Concurrently, the counter sets the Data Available (DAV) flag indicating a full input buffer. This flag is scanned by the Input Buffer Polling Circuit, which is the next topic presented.

3.1.2 The Input Buffer Polling Circuit

The Input Buffer Polling Circuit appears in Figure 3.3. This circuit sequentially scans each input buffer's DAV flag searching for a full buffer. A counter, which cycles through N values, drives the poller. The counter's output is supplied to the DAV multiplexer (MUX). Selection of one of the N MUX inputs is controlled by the counter's value. Each of the MUX inputs is a DAV signal from an input buffer circuit. The selected DAV signal is passed onto the Stop Scan flip-flop. When an active DAV signal is encountered, the Stop Scan flip-flop is set. Once set, this flip-flop halts the counter. Simultaneously, it brings the Input Buffer Service Request

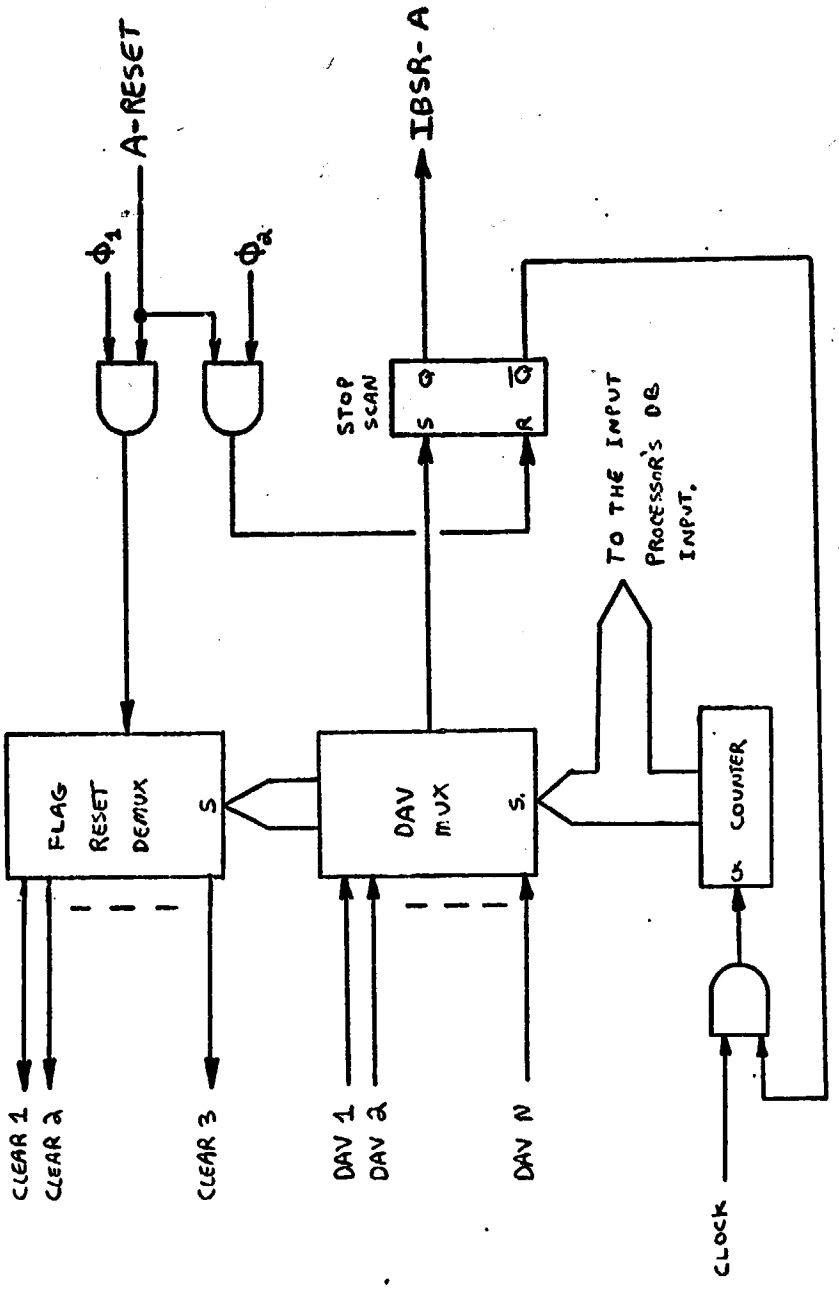


Fig. 3.3 Input Buffer Polling Circuit

(IBSR) line high, informing the Input Processor that a full buffer has been found.

The stabilized counter value represents the address of that full buffer. This value is sent to the Input Processor for processing. In addition, the counter's output is supplied to the Flag Reset Demultiplexer (DEMUX). This DEMUX allows the processor to send the A-RESET (see Table 3.1) signal to clear the proper buffer DAV signal. The A-RESET signal also clears the Stop Scan flip-flop, thus restarting the polling circuit.

3.1.3 The Input Switching Network

The Input Switching Network can provide a programmable data path between any input buffer and any location in the shift register array. This network consists of multiple, programmable data paths permitting the system to handle simultaneous packet transfers. A single data path is illustrated in Figure 3.4.

In order to establish a complete data path in the network, the Input Processor must first place the address of the input buffer being serviced into Latch A. The contents of this latch are supplied to the Data Mux and the Input Buffer Shift Clock Demux. The Data Mux links the selected input buffer to the switching network. The Shift Clock Demux supplies the shift clock to the selected input buffer. Once this half of the data path is established, the Input Processor sends the address of the empty shift register to Latch B. This latch provides the Data DeMUX, the Shift Register Array Clock

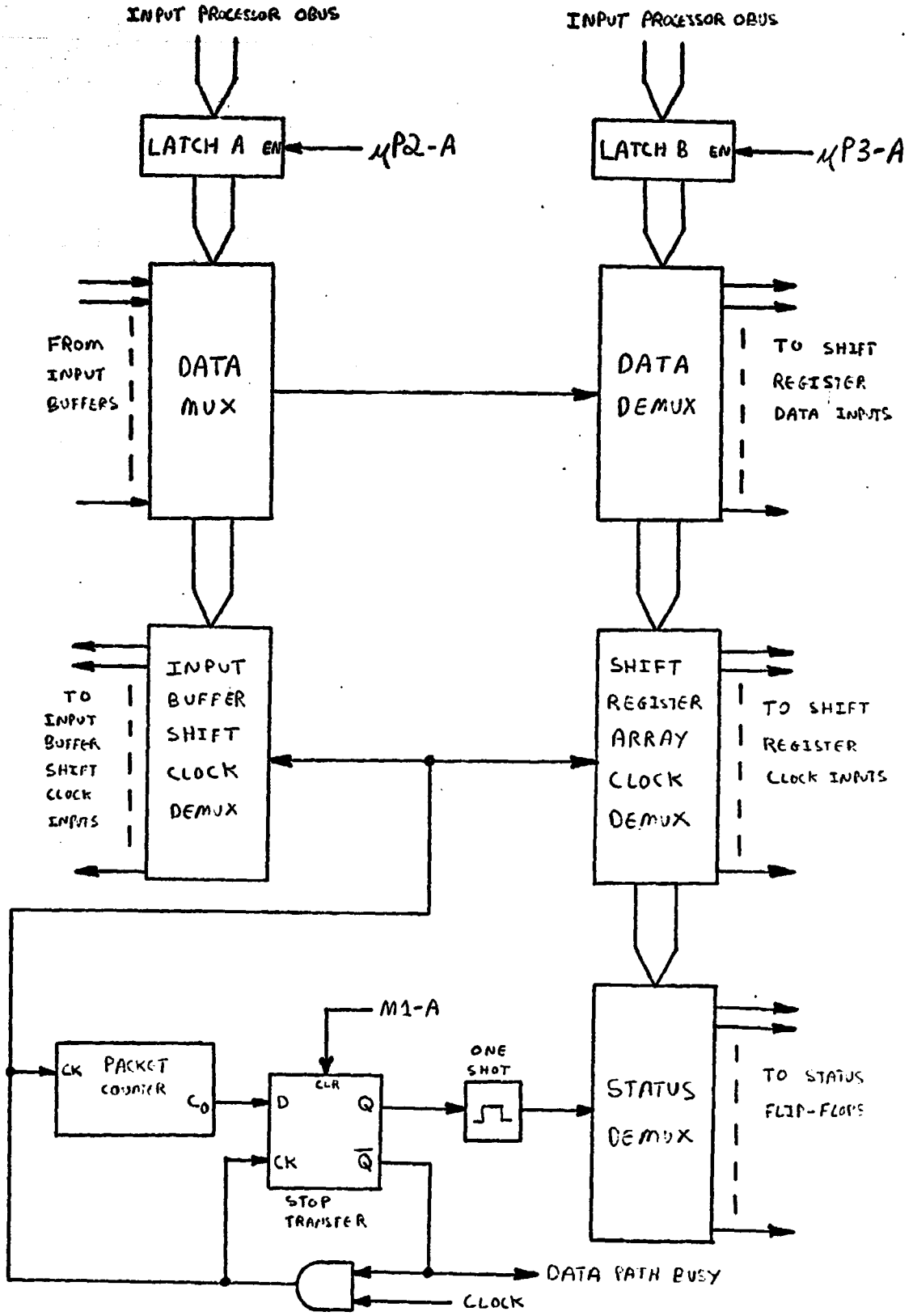


Fig. 3.4 Single Data Path in the Input Switching Network

Demux and the Status Demux with the select lines needed to complete the data path. The Data Demux links the selected shift register to the network, completing the actual data path for the packet transfer. The Shift Register Array Clock Demux supplies the shift clock to the shift register. The function of Status Demux and its associated hardware is explained in 3.1.4.

When a data path has been completed, the Input Processor initiates the packet's serial transfer through the data path by clearing the Stop Transfer flip-flop. This flip-flop halts the packet transfer once the packet counter rolls over. The packet counter counts each bit of the packet in transit by monitoring the shift clock pulses. This scheme permits every packet transfer into the array to be hardware terminated. In addition to halting the packet transfers, the Stop Transfer flip-flop generates the Data Path Busy signal. This signal indicates the status of the data path, which can be either idle or busy. Each Data Path Busy signal is sent to hardware which provides the Input Processor with the address of a free data path. Figure 3.5 is a diagram of this hardware circuit.

3.1.4 The Shift Register Array

The function of the Shift Register Array is to provide temporary storage for received packets which are waiting to be routed and transmitted. A single location in the array is shown in Figure 3.6.

As packets arrive from the input buffers, they are shifted into the shift registers in the array. Each location actually

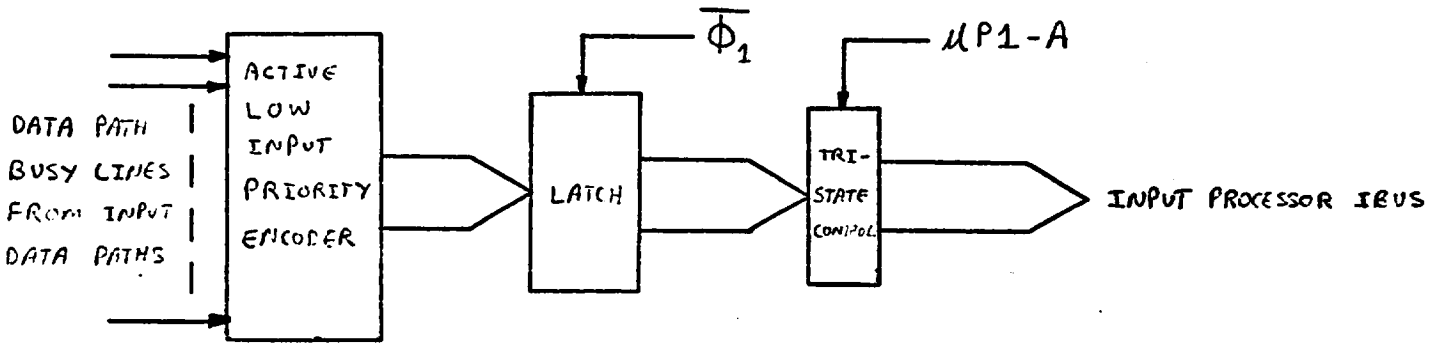


Fig. 3.5 Input Data Path Busy Port

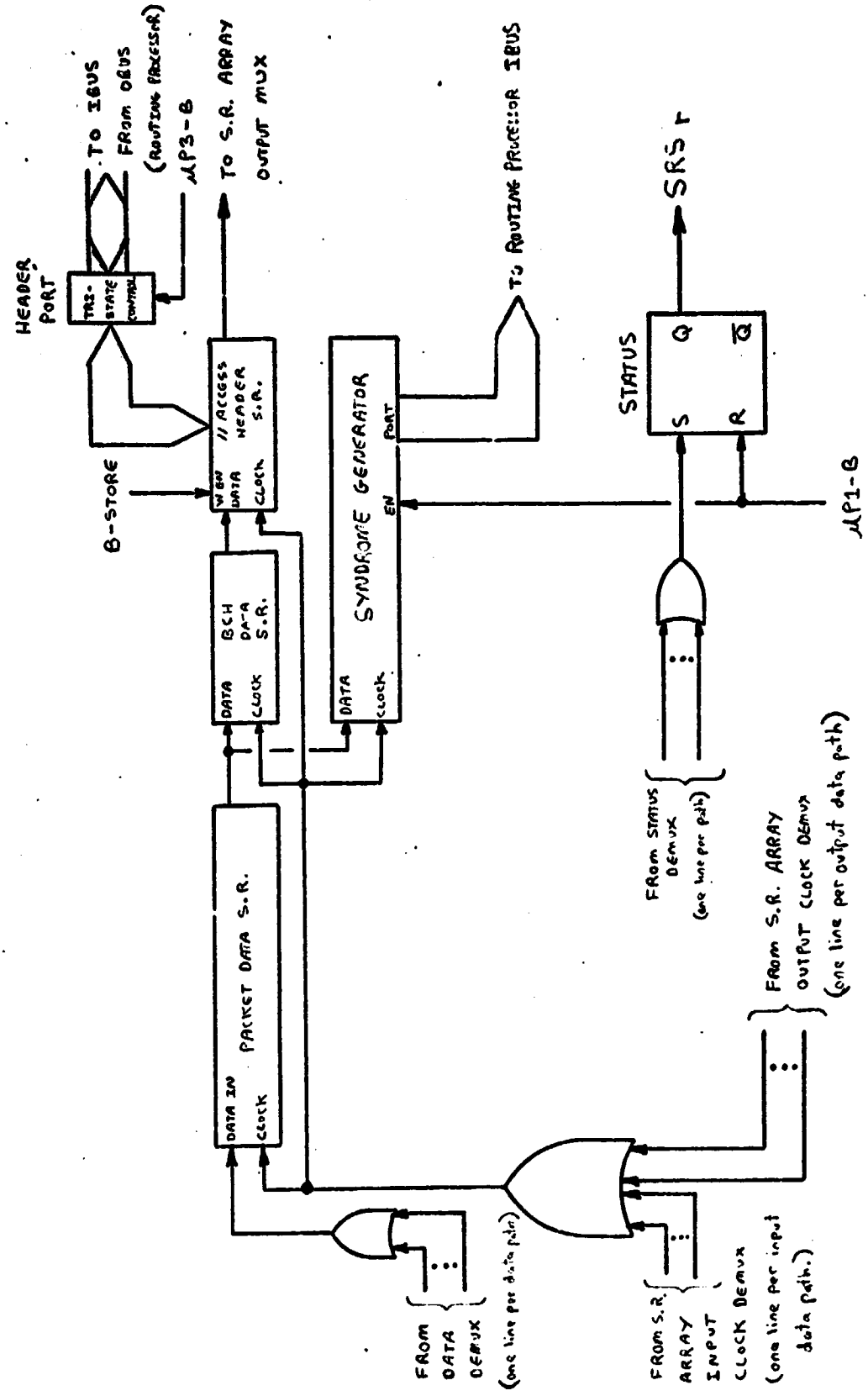


Fig. 3.6 One Location in the Shift Register Array

uses three shift registers linked together to form the packet-length storage area required. As the packets are transferred, the header arrives first and eventually resides in the Packet Header Shift Register. Unlike the shift registers which contain the actual packet data bits and the header error protection bits, this shift register allows parallel accessing of the header. Since the processor fetches each packet header and also returns the corrected header to the shift register, the parallel access feature is a system requirement.

As packets are sent to the array, their headers and header correction bits are also sent to the Syndrome Generator [3]. Each shift register location has its own Syndrome Generator. This hardware circuit decodes the header information into a syndrome. A non-zero syndrome indicates an error in the header data. The syndrome is available to the Routing Processor which corrects the header using this error pattern information.

All packet transfers into the array from the input buffers are hardware terminated. When the Stop Transfer flip-flop in the Input Switching Network is set, it halts the packet transfer hardware. In addition, the activated flip-flop is sent to the input of the Status Demux. This Demux passes the flip-flop's signal onto the selected shift register's Status flip-flop. The activated signal sets the Status flip-flop indicating that a packet transfer has been completed and that this location in the array now contains a packet requiring service. Every array Status flip-flop is scanned by

the Shift Register Polling Circuit, which appears in Figure 3.7. This poller searches for unserviced packets, notifies the Routing Processor once one is found, and supplies the array address of the unserviced packet to the processor. A set Status flip-flop is cleared once the Routing Processor accesses the Syndrome Generator at that location. Next, the poller is restarted by the processor.

Previously, the polling of the array was carried out by the processor. This scheme required additional software and consumed processor execution time even when empty locations were scanned [3]. Thus, the proposed use of a hardware poller increases the Routing Processor's throughput.

3.1.5 The Output Queue Lists

The Output Queue Lists are the software lists containing the shift register array address of each routed packet awaiting transmission. Each list contains the addresses of routed packets destined for that list's associated output buffer. The Routing Processor always writes to the lists, adding the addresses of newly routed packets. Meanwhile, the Output Processor always reads from these lists, fetching the next packet to be transmitted. The lists are organized in a First-In-First-Out (FIFO) format, resulting in the transmission of the oldest packet in the selected list. Figure 3.8 contains the data structure of the Output Queue Lists.

The index pointer or "Input Pointer" (IPTR) used by the Routing Processor points to the next address to be filled. Once a location is filled, the Routing Processor updates the

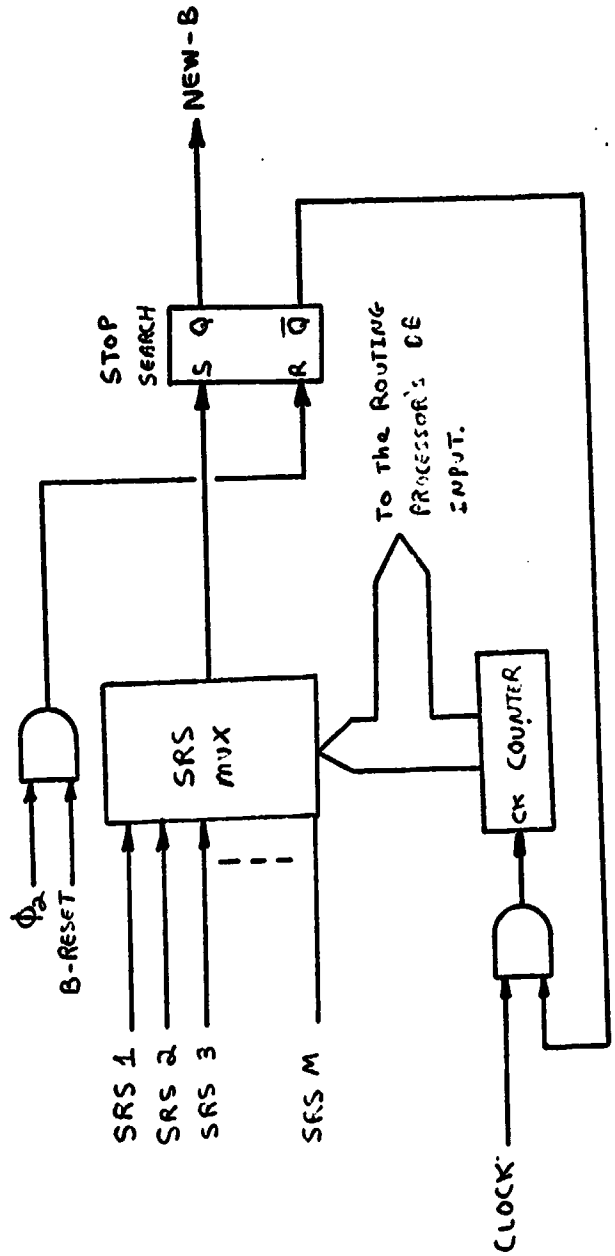
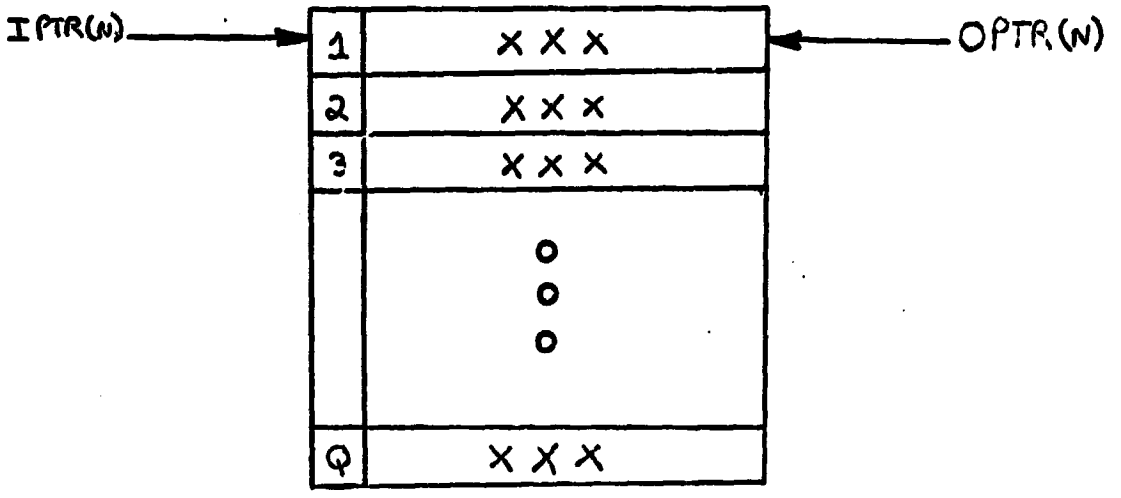


Fig. 3.7 Shift Register Polling Circuit



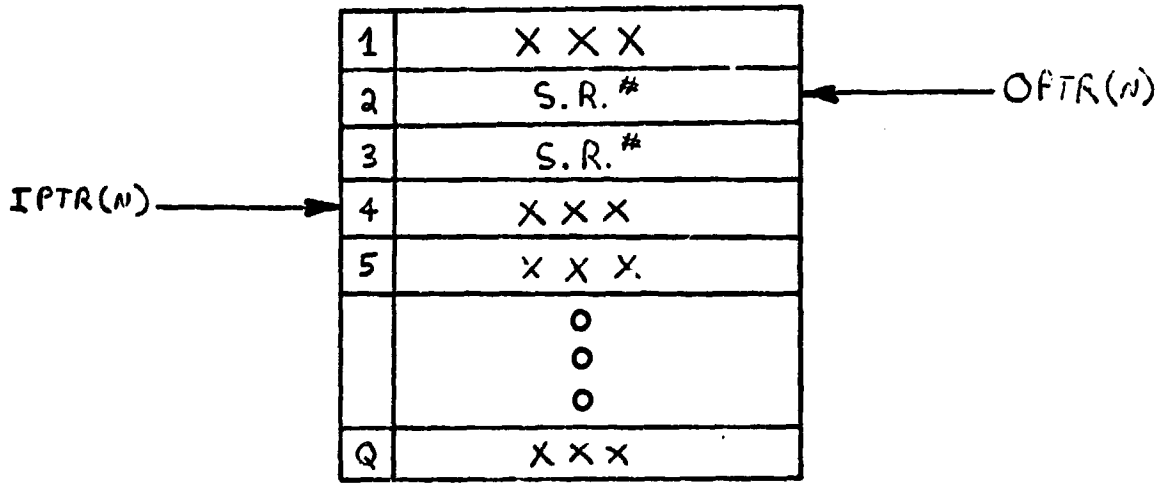


Fig. 3.8 Output Queue List Data Structure

IPTR by incrementing it. When the IPTR reaches the end of the list, it rolls over, returning to the top of the queue list. The index pointer or "Output Pointer" (OPTR) used by the Output Processor points to the next location to be read. In order to fetch the next address from a list, the Output Processor must perform the read operation and then it must increment the OPTR.

The data structure of the lists is designed such that when IPTR is equal to OPTR the list is assumed to be empty. Under special circumstances, this assumption may cause packet losses. This problem is explored further in Chapter 5.

In the single processor design, the output queue lists are stored in local RAM and the index pointers are stored in the processor's register file. However, in the multiprocessor environment of the new designs, this scheme no longer meets system demands. Both the Routing Processor and the Output Processor must access these lists. Therefore, the Output Queue lists must be stored in RAM's that are available to both processors. In order to reduce contention problems, each list is stored in a physically different RAM structure. This permits the two processors to simultaneously access different lists without interference. Special locking hardware is required to prevent simultaneous access of one RAM should the processors fail to access different lists. As mentioned earlier, the RAM's used are TWO-PORT RAM's. The logic diagram of the AM29705 chips used is presented in Figure 3.9 [6]. Several chips can be arranged to form a RAM structure of required width and length.

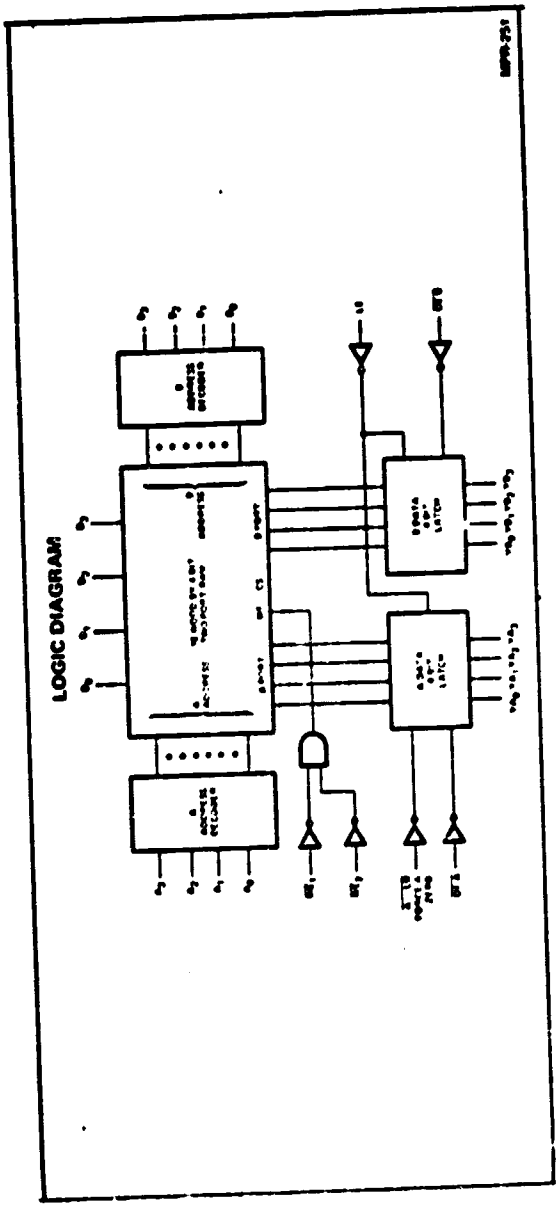


Fig. 3.9 An 29705 Two-Port RAM

An additional constraint in the design of the queue lists is the requirement that the value of each index pointer be available to hardware test logic. The function of the test logic is to notify the Output Processor when an output queue list becomes empty (OPTR = IPTR). Fulfilling this requirement results in the storing of all queue list index pointers in hardware counters. Figure 3.10 is the logic diagram of one Output Queue List structure. The operation of this circuit is best explained by tracing the procedure followed by the Routing Processor and the Output Processor as they access a queue list.

Once the Routing Processor has determined the destination of a packet, it activates the μP4-B (see Table 3.1) control line which selects the desired Output Queue List. These control lines, when activated, enable the selected RAM, the associated locking circuit, and the IPTR updating circuit. Next, the processor places the shift register array address into the Output Queue List Data Port. The Routing Processor then activates the B-REQUEST lines to request access to the queue list. This control signal is sent to all the queue lists, but is enabled only at the queue list selected by the μP4-B signal.

If the selected queue list is available, the B-REQUEST signal sets the WRITE ACCESS CONTROL flip-flop. This flip-flop then activates the READ LOCK-OUT line, which disables the READ ACCESS CONTROL flip-flop. Disabling this flip-flop locks out the Output Processor from this list. In addition,

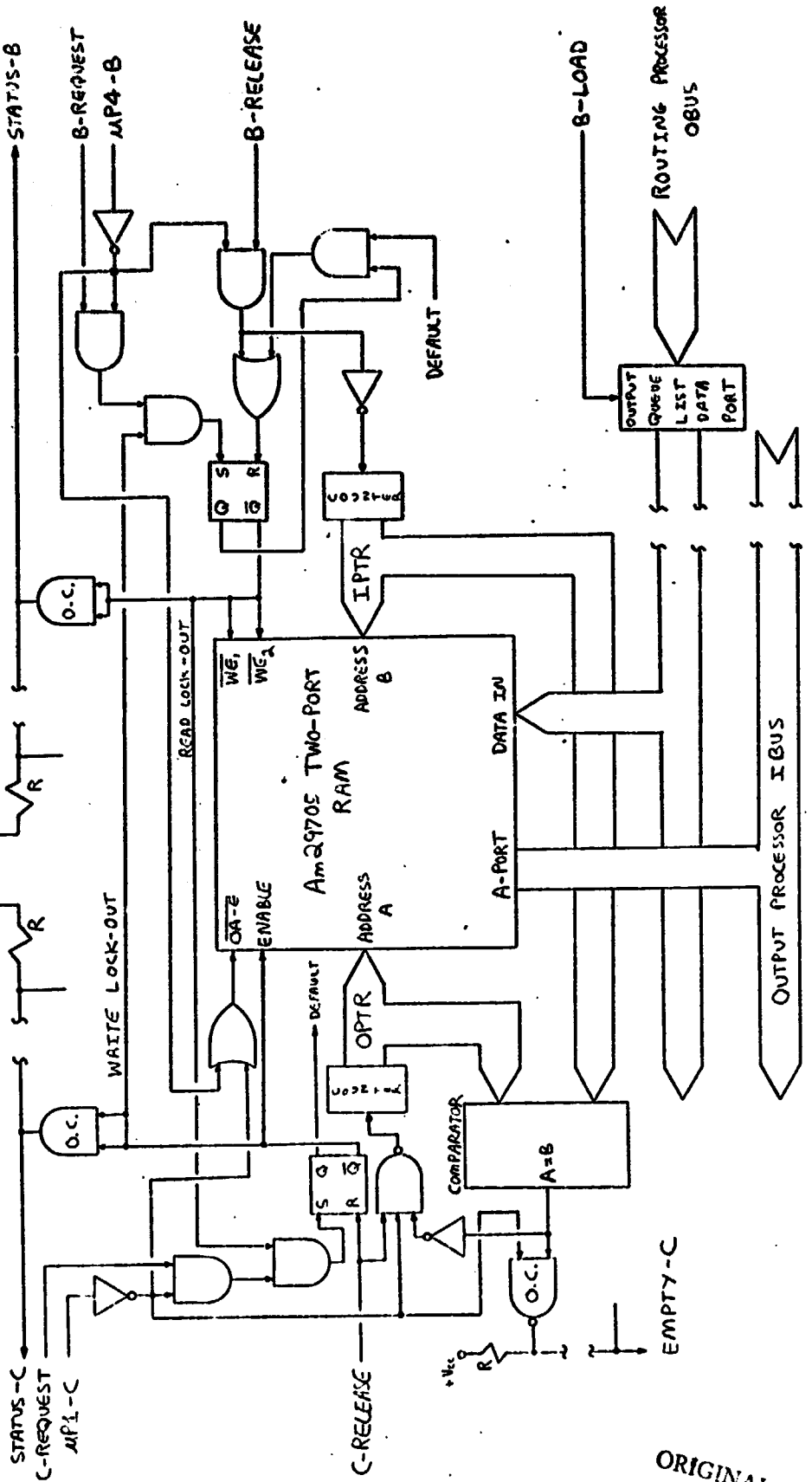


Fig. 3.10 One Output Queue List

ORIGINAL PAGE IS
OF POOR QUALITY

the set WRITE ACCESS CONTROL flip-flop activates the WRITE signal, which enables the RAM in the write mode. The address data latched in the Output Queue List Data Port is then strobed into the RAM location selected by the IPTR. The IPTR counter has as many unique values as the RAM has locations. The Routing Processor is informed of a completed write operation by the STATUS-B signal, which goes low when the WRITE ACCESS CONTROL flip-flop is set. Upon receiving the active-low STATUS-B signal, the Routing Processor reads the associated Output Status Word (OSW). (The function and operation of the OSW is discussed in 3.1.8.) The reading of the OSW before the release of the queue list is required since access to the OSW is also controlled by the queue list lock hardware. Thus, only one processor can access both the queue list and the associated OSW. Once the OSW read operation is performed, the Routing Processor generates the B-RELEASE signal. This signal clears the WRITE CONTROL ACCESS flip-flop. Clearing this flip-flop frees the list since the READ LOCK-OUT signal and the WRITE signal are de-activated. In addition, the B-RELEASE signal activates the IPTR UPDATE signal which increments the IPTR counter.

If the Output Queue List selected is locked by the Output Processor, the B-REQUEST line is disabled by the WRITE LOCK-OUT line. The Routing Processor is informed of its denied access via the STATUS-B line, which remains high after the access request. The action taken by the Routing Processor in this event is discussed in 3.3.2.

The Output Processor must access an Output Queue List each time it services an empty output buffer. In order to access the queue list associated with the output buffer being serviced, the Output Processor must first activate the proper µP1-C line. The activated µP1-C line enables the seclected RAM, the associated locking circuit and the OPTR updating circuit. Next, the Output Processor generates the C-REQUEST signal. If the selected queue list is locked by the Routing Processor, the C-REQUEST line is disabled by the READ LOCK-OUT line. The active-low STATUS-C line will remain high after the request, notifying the Output Processor of its access denial. The action taken by the Output Processor is discussed in 3.3.3.

If the requested queue list is available, the enabled C-REQUEST signal will set the READ ACCESS CONTROL flip-flop. Setting this flip-flop activates the WRITE LOCK-OUT, READ, and STATUS-C lines. The activated WRITE LOCK-OUT disables the B-REQUEST signal, locking the Routing Processor out from this queue list. Also activated is the READ signal, which places the RAM in the enabled read mode. In addition, the STATUS-C line goes low informing the Output Processor that access has been granted. Once access has been granted, the Output Processor checks the EMPTY-C line to determine if the list is empty. This line is driven by a comparator whose inputs are the values of IPTR and OPTR. If the two pointers are equal, the comparator activates the EMPTY-C line.

If the list is empty, the Output Processor releases the list by generating the C-RELEASE line. The OPTR is not

incremental in this situation since the list is empty. Should the list not be empty, the Output Processor reads the packet's address from the location selected by the OPTR. Once the read operation is complete, the Output Processor activates the C-RELEASE line freeing the list. In addition, the activated C-RELEASE signal increments the OPTR.

Should both the Routing Processor and the Output Processor request the same queue list simultaneously, a Default Circuit locks out the Routing Processor while granting access to the Output Processor.

An important point to note about this component is that although the hardware implementation of the index pointers is a system requirement, the system is enhanced by this feature. The first benefit of this scheme is the reduction of software due to the decrease in index pointer management overhead. The second benefit is the reduced number of register files required by the processors since the index pointers are stored externally. This reduces the processor's complexity. An additional point about this scheme is that it can be implemented in single processor systems as well as in multiprocessor systems.

3.1.6 The Output Switching Network

Illustrated in Figure 3.11 is one data path in the Output Switching Network. As in the Input Switching Network, the function of this network is to provide the Output Processor with programmable data paths. These data paths are used to link shift registers in the array to output buffers. Packet

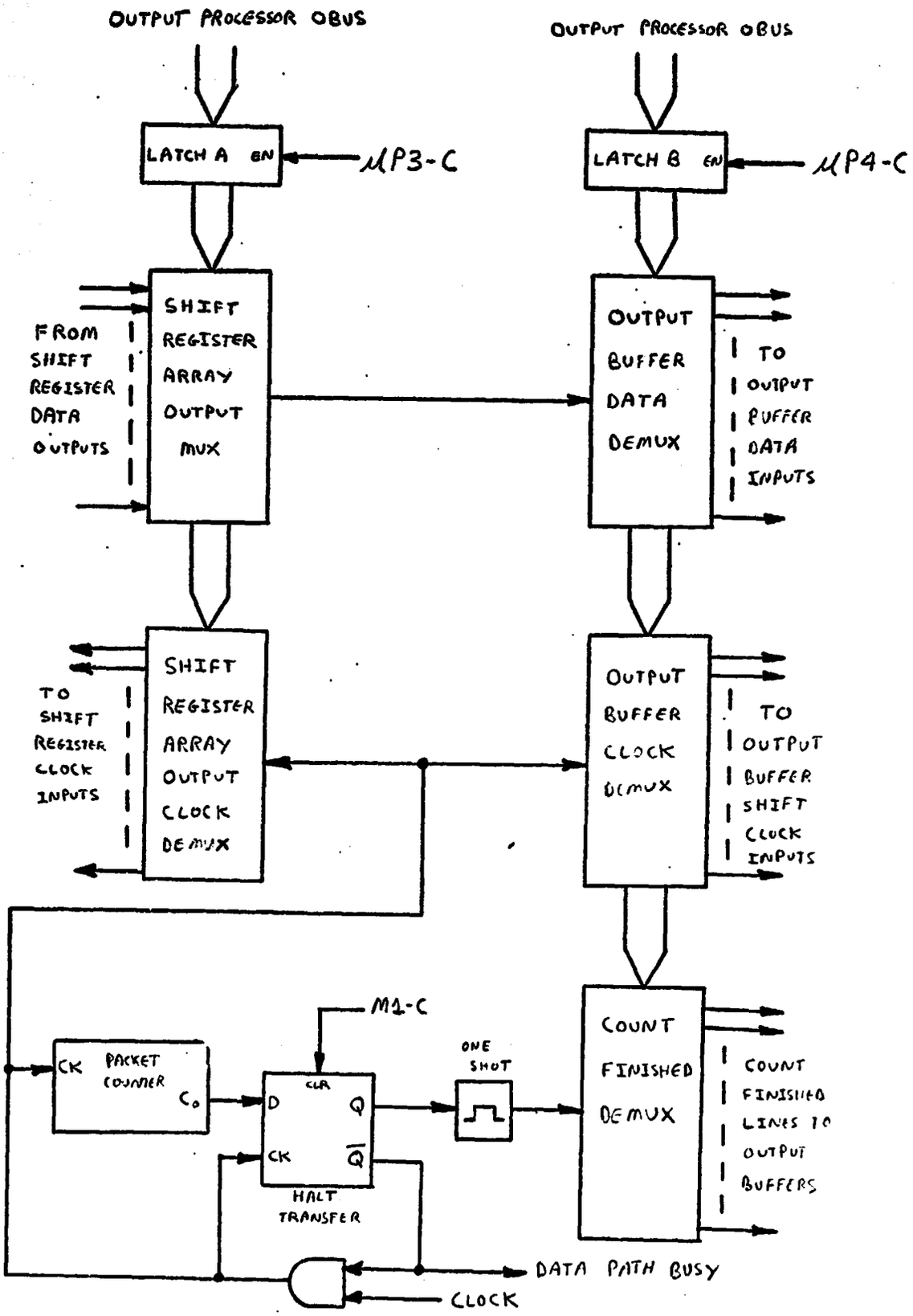


Fig. 3.11 One Data Path in the Output Switching Network

transfers through the switching network are processor initiated and hardware terminated. There are multiple data paths allowing simultaneous packet transfers. The circuitry required to monitor the status of all data paths in the switching network is presented in Figure 3.12. This circuit provides the Output Processor with the address of a free data path when one is needed.

3.1.7 The Output Buffers

The function of the output buffers is to receive packets transferred from the shift register array and to then transmit those packets to the external channel hardware. Packets arrive at a rate determined by the internal shift clock. Packets then leave at the rate maintained by the external line clock. The logic diagram for one Output Buffer is given in Figure 3.13.

The central component of the buffer is a packet-length shift register where the packets are stored. While the COUNT FINISHED line is inactive, the packet is shifted into the shift register by the internal shift clock. Meanwhile, the INHIBIT XMIT flip-flop remains set, disabling the external shift clock. Once the COUNT FINISHED line is activated by the Output Switching Network, the INHIBIT XMIT flip-flop is cleared. This action enables the external shift clock, which then begins to shift the packet onto the channel line. The packet counter monitors the complete transfer. As soon as the last packet bit is shifted out of the buffer, the counter rolls over. The carry out line from the counter sets the

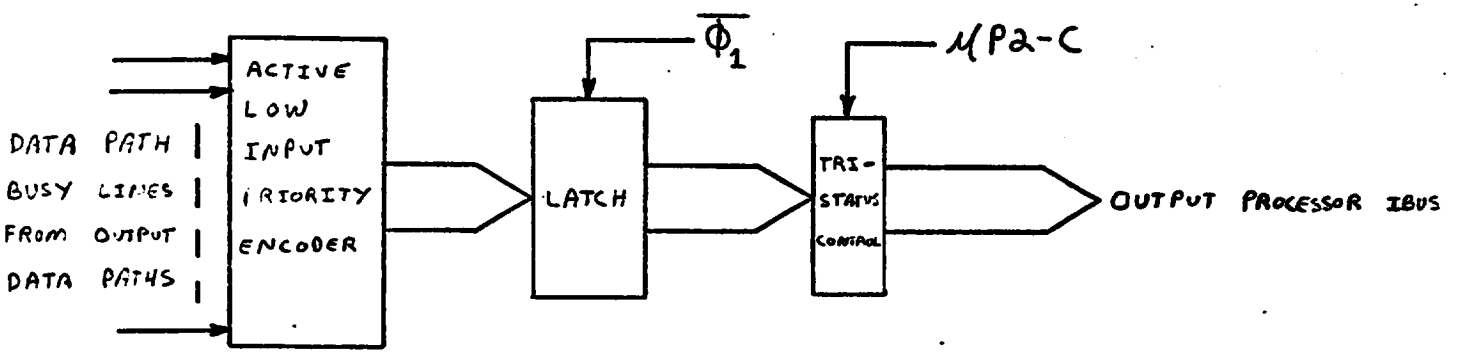


Fig. 3.12 Output Data Path Busy Port

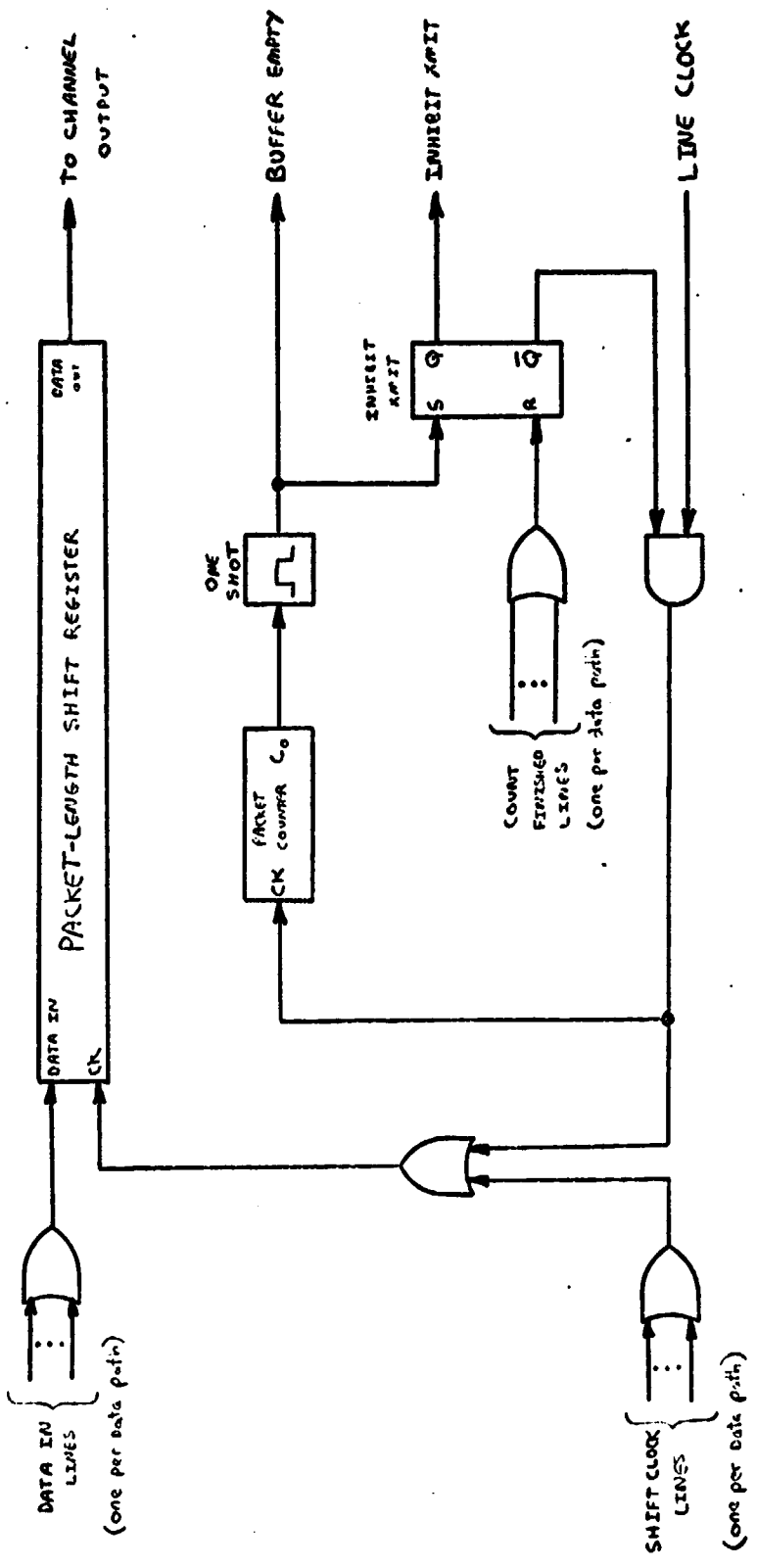


Fig. 3.13 One Output Buffer

INHIBIT XMIT flip-flop and generates the BUFFER EMPTY signal. The BUFFER EMPTY signal is supplied to the output buffer's Output Status Word (OSW). The function and operation of the OSW is presented as the next topic.

3.1.8 The Output Status Words

The Output Status Word (OSW) of an Output Buffer is hardware circuitry used to monitor and reflect the current status of the buffer. All OSW's are accessible to both the Routing Processor and the Output Processor. Each OSW is linked to an associated Output Queue List. Thus, just as in the case of the queue lists, only one processor may access a particular OSW at any given time. This scheme eliminates the possibility of one processor reading an OSW while the other processor is altering the same OSW.

Each OSW indicates one of the three states that its corresponding output buffer is in. The three output buffer states are Busy, Empty and Idle. An output buffer is in the Busy state whenever it is receiving a packet, transmitting a packet or receiving Output Processor service. Output buffers enter the Empty state when the packets that they were transmitting are completely transferred onto the channel lines. The Output Processor places an empty output buffer in the Idle state when the corresponding Output Queue List is also empty. The hardware implementation of one OSW and the Output Buffer Polling Circuit used to scan each OSW is illustrated in Figure 3.14.

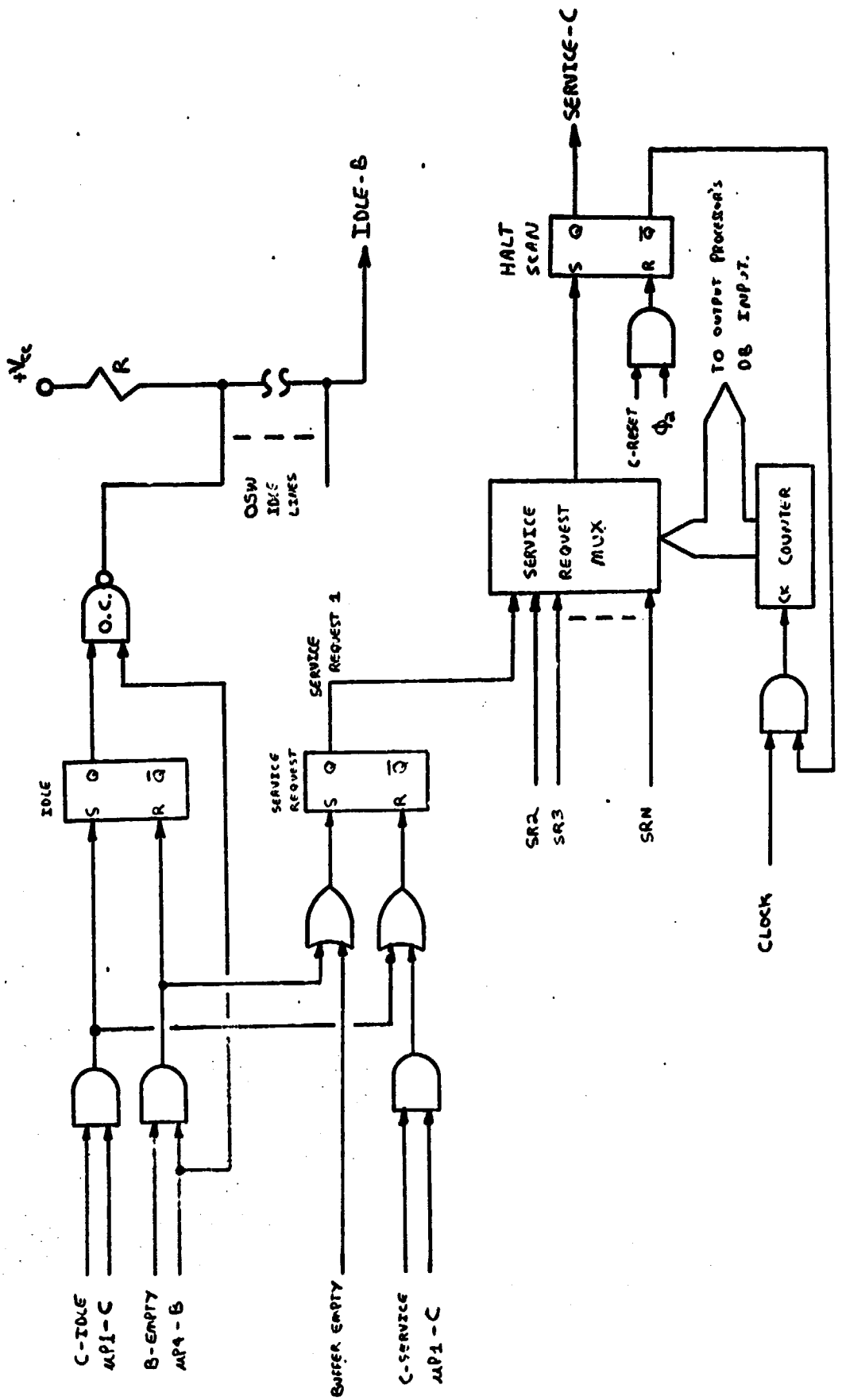


Fig. 3.14 One Output Status Word and the Output Buffer Polling Circuit

The Output Buffer Polling Circuit sequentially scans each OSW in search of an empty buffer. When a buffer empties, its support hardware generates a BUFFER EMPTY signal. This signal sets the OSW's SERVICE REQUEST flip-flop. The activated SERVICE REQUEST line is eventually found by the poller as it scans the CSW's. Finding an empty buffer, the poller signals the Output Processor and supplies the processor with the address of the empty buffer.

The Output Processor then accesses the Output Queue List associated with the empty buffer. If the list is empty, the Output Processor updates the OSW to indicate that the buffer is in the Idle state. This update is done when the Output Processor generates the C-IDLE signal. (The proper μP1-C select signal is still enabled from the queue list access.) The poller is restated by the C-RESET signal. If the list is not empty, the C-SERVICE signal is activated to clear the SERVICE REQUEST flip-flop. This updates the OSW to indicate that the buffer now is busy. The poller is restarted by the C-RESET signal.

Every time the Routing Processor updates an Output Queue List, it checks the corresponding OSW. If the OSW indicates that the buffer is not idle, the OSW is left unchanged. However, if the OSW indicates that the buffer is in the Idle state, the Routing Processor updates the OSW to indicate that the buffer is empty. This update is accomplished when the Routing Processor activates the B-EMPTY signal. (The proper μP4-B select line is still enabled from the queue list access.)

3.1.9 The Empty Shift Register List

The Empty Shift Register List (ELIST) contains the array addresses of every empty shift register in the array. This list is read by the Input Processor and written to by the Output Processor. Figure 3.15 shows the data structure used to maintain the list. The index pointer (EPTRØ) used by the Input Processor points to the next shift register address to be fetched. Once the address data is fetched, the Input Processor increments EPTRØ. The index pointer (EPTR1) used by the Output Processor points to the last location updated with the address of a freed shift register. The Output Processor must first increment EPTR1 and then perform the write operation.

This data structure is designed such that under normal operation, a Read and a Write operation will not take place at the same location. Thus, both processors can simultaneously access the list without interference.

Illustrated in Figure 3.16 is the hardware circuit required to implement the ELIST. As in the Output Queue List system, the pointers are implemented in hardware and the RAM is a 2-port RAM. Although this is not necessary, since neither processor requires access to the other's pointer, it does reduce software overhead. Since this increases throughput, this scheme is proposed over the previous scheme of storing the pointers in the register file. The use of hardware index pointers could also be used in a single processor

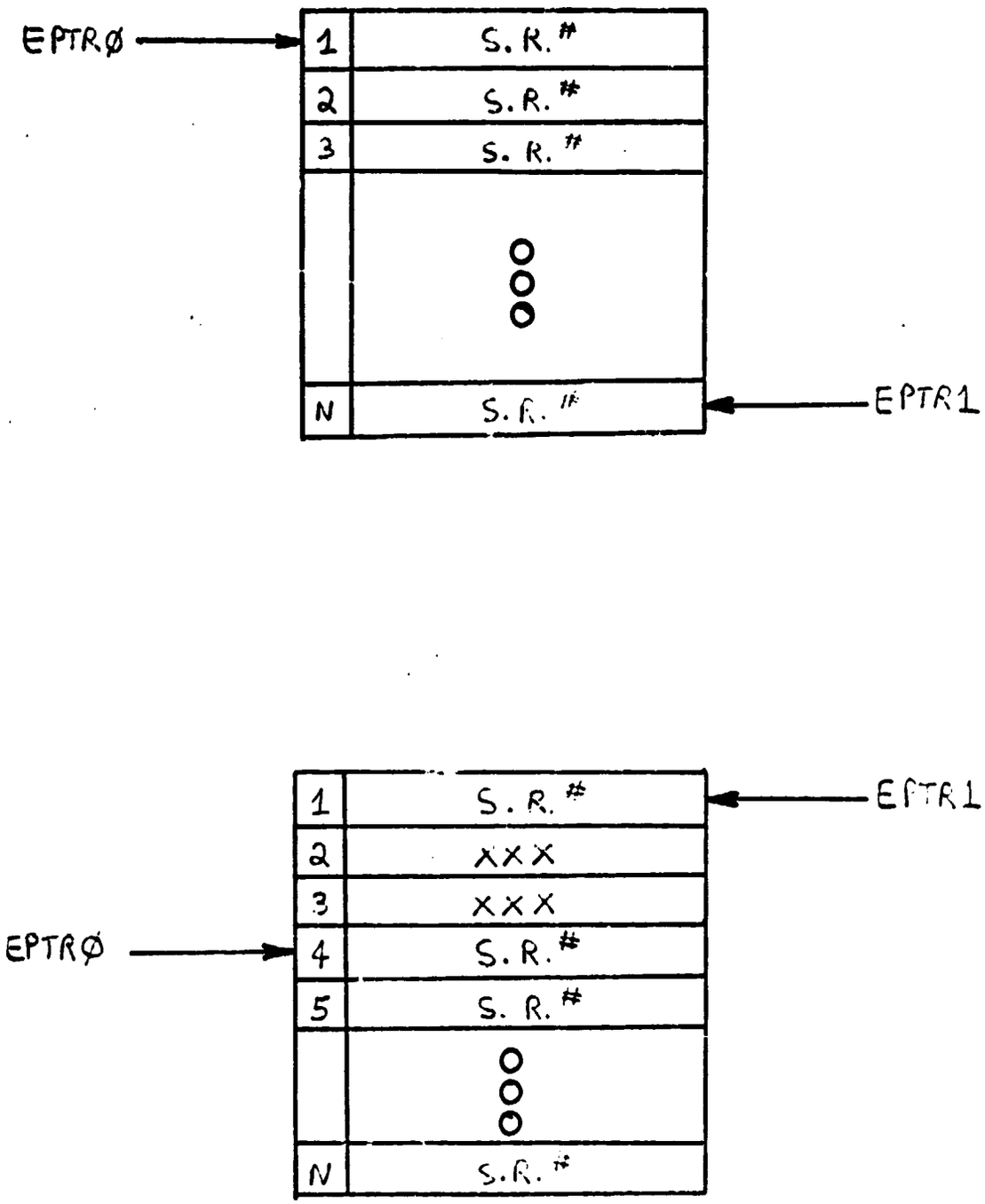


Fig. 3.15 The Empty Shift Register List Data Structure

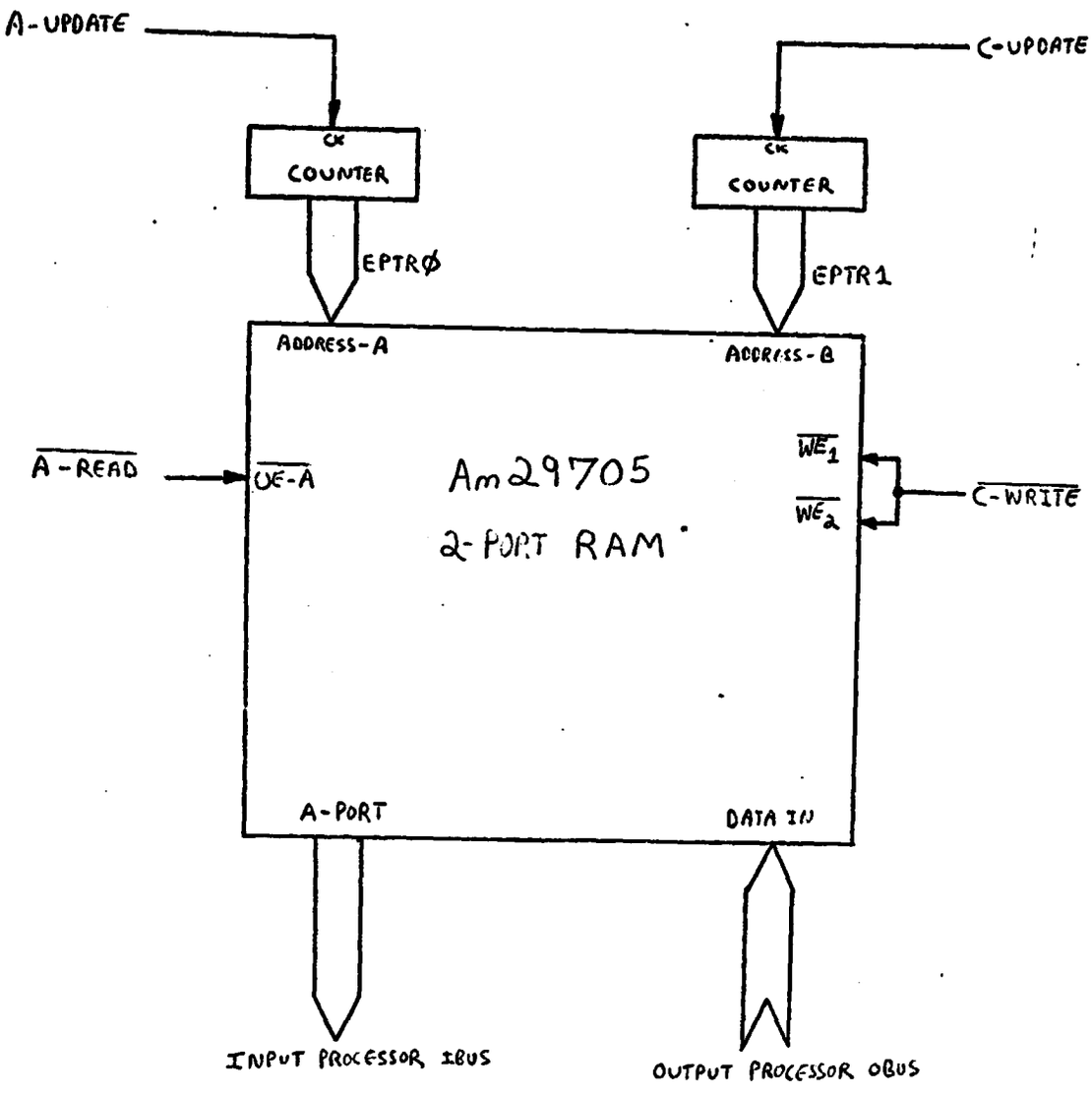


Fig. 3.16 The ELIST Hardware

system by using an up/down counter to hold a single index pointer. The ELIST data structure which supports the single index pointer is found in [2] and [3].

3.2 The Processors

As stated earlier, there exist three classes of processors in this implementation of a packet switch. Although each processor's function is quite different, the actual processor used in each class is constructed around a similar architecture. The custom software executed by each processor and the blocks of unique support hardware are the two elements which give each class of processor its distinct character. As in the single processor design, the processors are built using the Advanced Micro Devices (AMD) 2900 family of bit-sliced processing components. The design considerations which led to the selection of these components are discussed in [2] and [3].

3.2.1 General Processor Architecture

The architecture of all classes of processors is comprised of two functional blocks: The Microprogram Control Unit (MCU) and the Instruction Execution Unit (IEU). The Routing Processor contains one additional functional block: The Syndrome Read Only Memory (ROM) which contains the header error correction information in a lookup table format. Figure 3.17 contains the block diagram of the processor architecture.

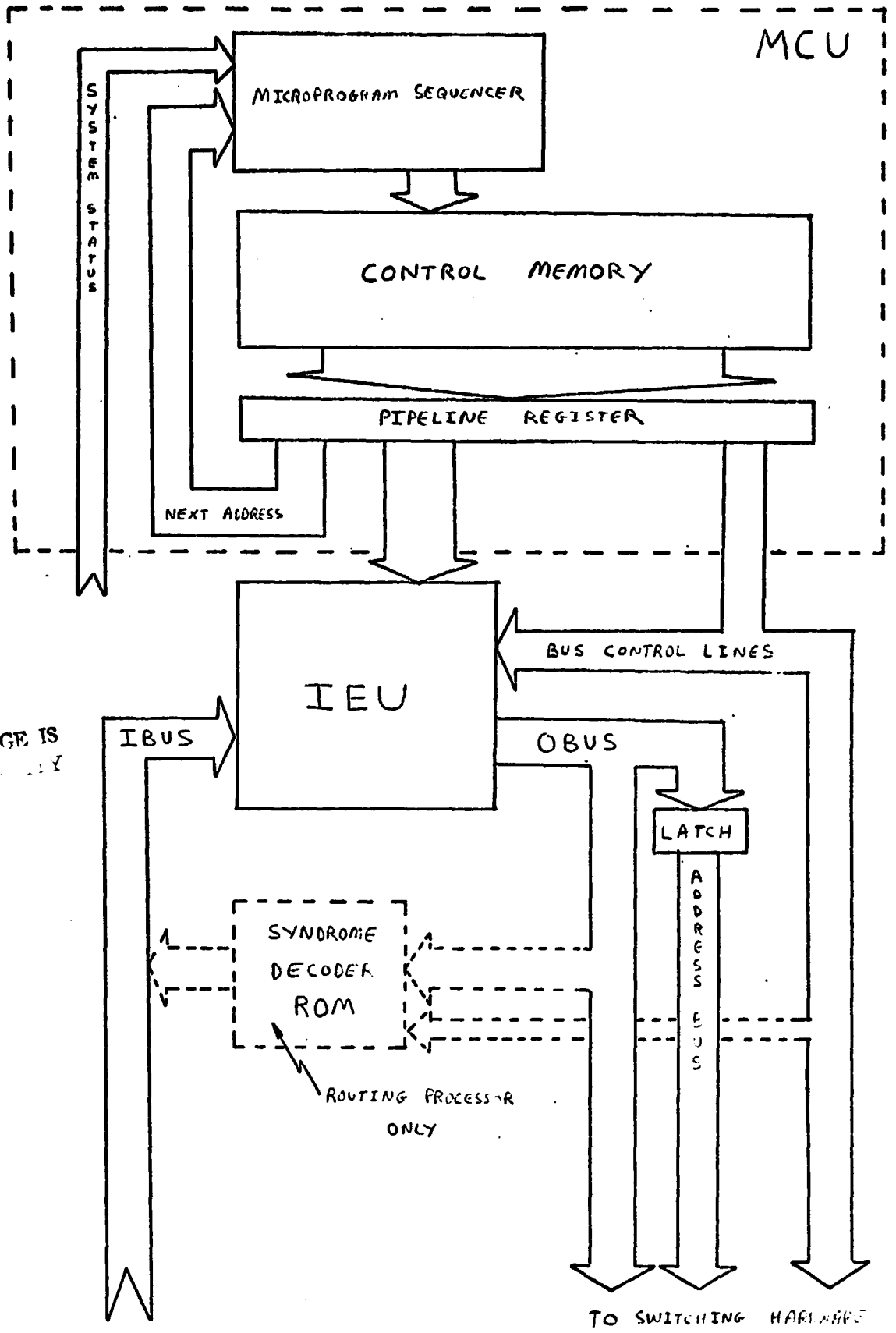


Fig. 3.17 The Processor Architecture

3.2.2 The Instruction Execution Unit

The Instruction Execution Unit (IEU) of the Input Processor and the Output Processor is presented in Figure 3.18. Figure 3.19 shows the IEU of the Routing Processor. Both versions of the IEU incorporate the AMD 2903 four-bit ALU slices. Shown in Figure 3.20 is the block diagram of the AMD 2903 ALU chip. Cascading these chips in parallel will provide the required width of the processor word. The AMD 2903 has been selected over the AMD 2901 ALU because the 2903 architecture supports two Direct Data Inputs. The use of the second data input allows the data from the polling circuits to be directly supplied to the ALU. This reduces software overhead since a typical two instruction read operation is no longer required. Instead, the data is sent directly to the ALU during the execution of a single instruction. Since this scheme is implemented for each class of processors, a total of three memory cycles has been saved, improving throughput.

All the arithmetic and logical operations required for address generation and data manipulation are carried out by the IEU. Inputs to the ALU are supplied by five different sources: The Input Bus (IBUS), the Microprogram Word (μW), Scratchpad 1, Scratchpad 2, and the polling circuit. The IBUS provides a data path from all external memory and data ports to the ALU. Immediate operands in the Control Memory are supplied to the ALU via the μW input. Scratchpads 1 and 2 are two file registers located in the ALU's internal RAM. Their addresses are supplied by an external circuit which can

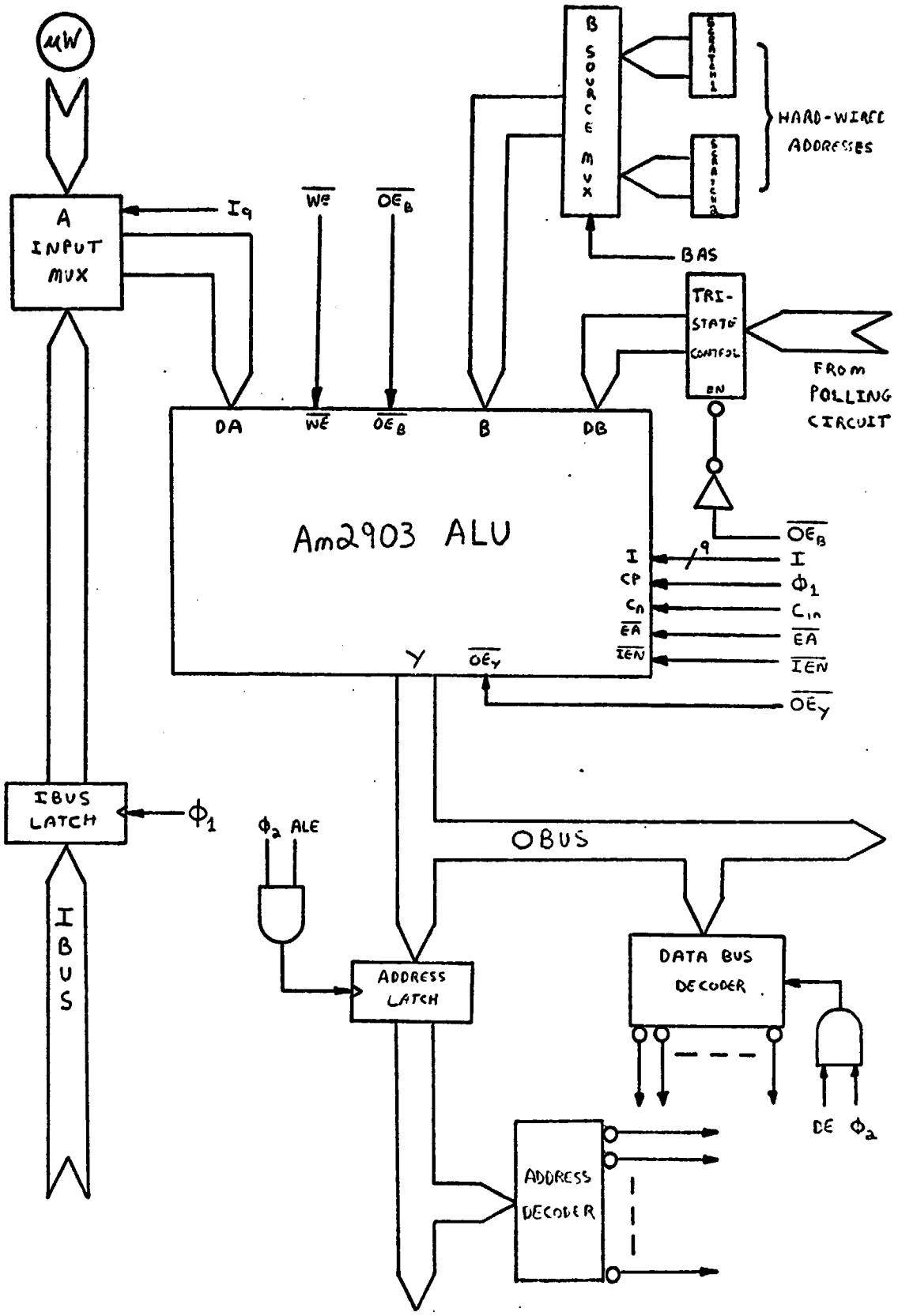


Fig. 3.18 The IEU for the Input and Output Processors

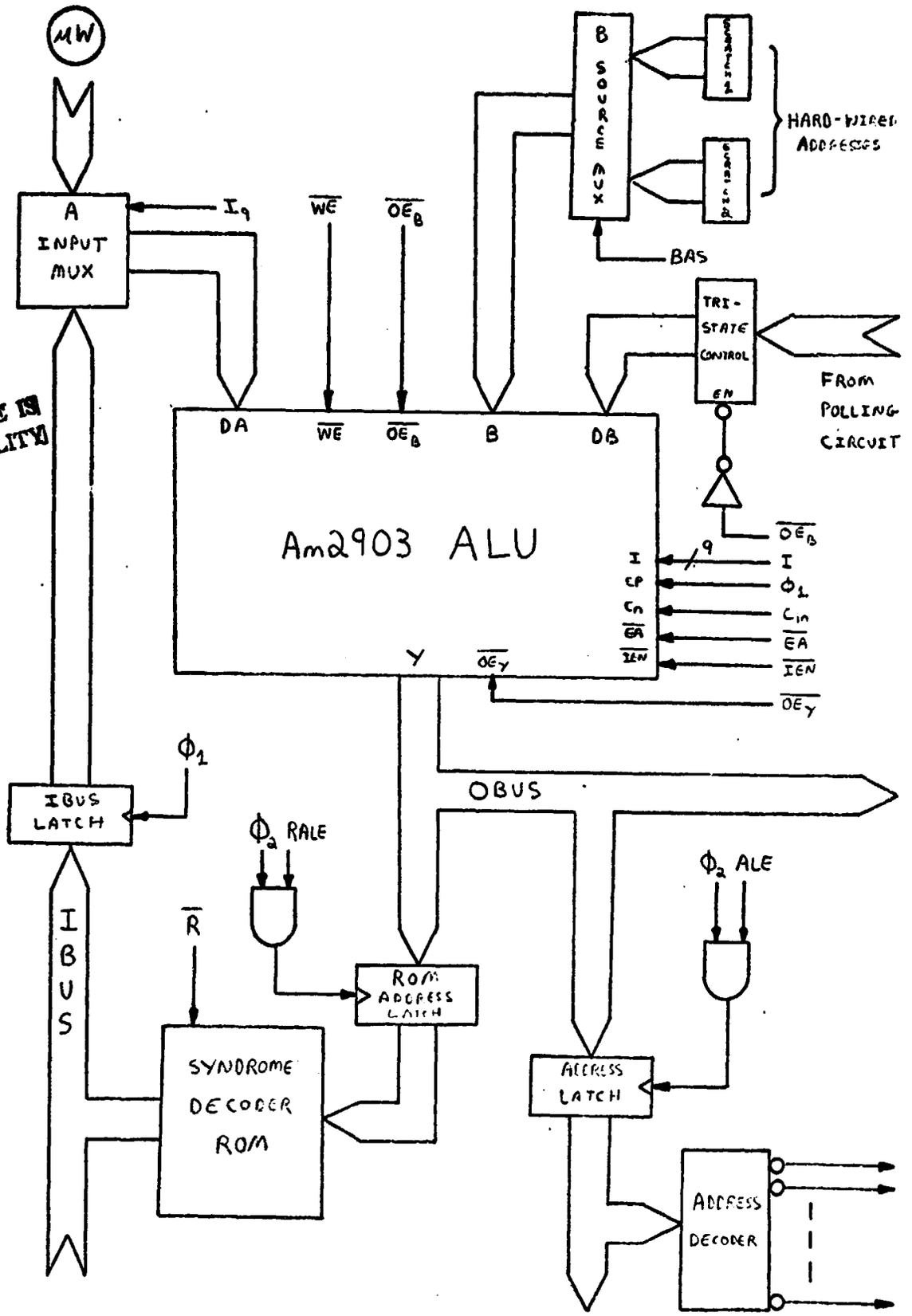


Fig. 3.19 The Routing Processor's IEU

BLOCK DIAGRAM

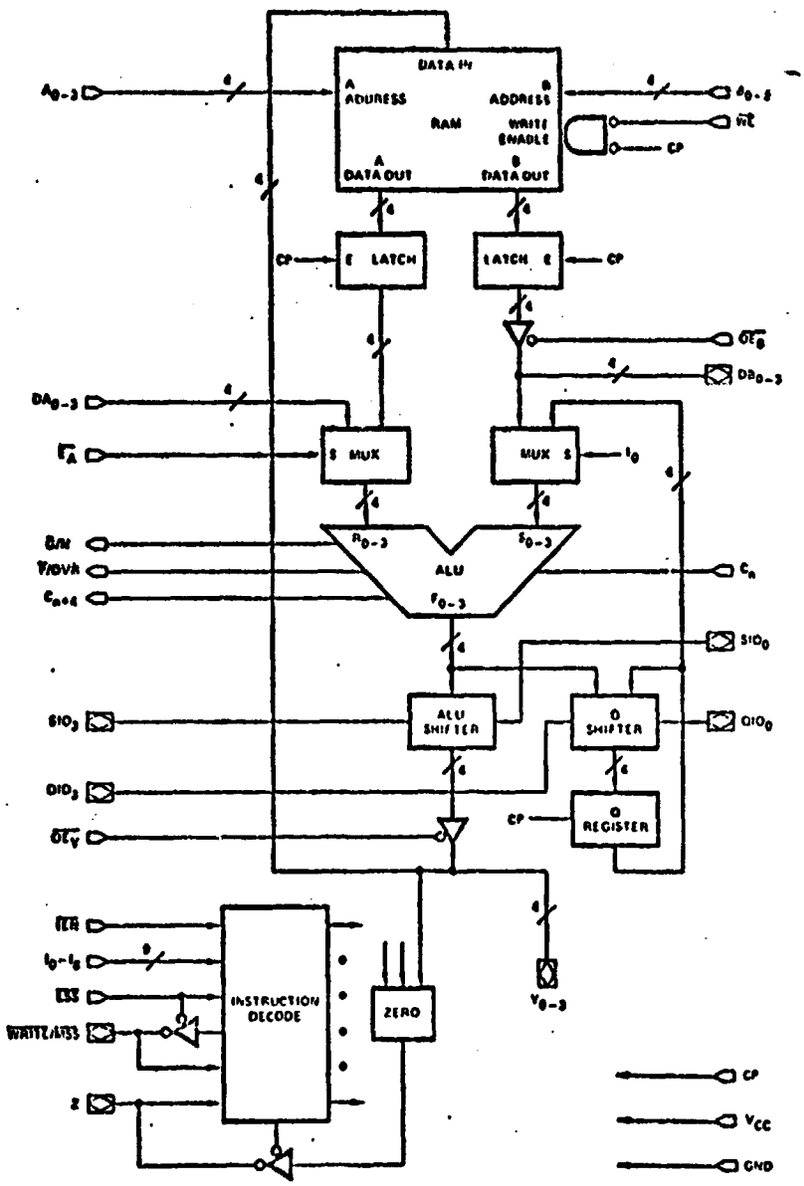


Fig. 3.20 AM 2903 Four-Bit ALU Slice

provide the hardwired address of the selected register, when needed. Data from the polling circuit provide the processor with the address of the device requesting service.

Once the ALU inputs are processed, they leave the ALU via the Output Bus (OBUS). This bus is sent to system hardware, the Address Latch, the Address Decoder and the Data Bus Decoder (Input Processor and Output Processor only). The Address Latch holds address data stable during read and write operations. The Address Decoder generates device select lines. When used in conjunction with the Data Bus Decoder, the Address Decoder forms an Addressing Matrix which can activate single bit control lines [3]. This matrix is illustrated in Figure 3.21.

3.2.3 Microprogram Word IEU and System Hardware Control Fields

IEU hardware and blocks of system hardware receive control signals from various fields within the Microprogram Word (μW). Along with control signals, the ALU can receive operands from the microprogram word. Control signals from the μW are also sent directly to systems hardware blocks. These signals do not require processing by the IEU. Therefore, while the processor performs one task, the μW control signals can activate components of the system hardware. This hardware can either assist the processor in completing its task or will independently perform a different task. This scheme permits concurrent operations to be carried on within the packet switch.

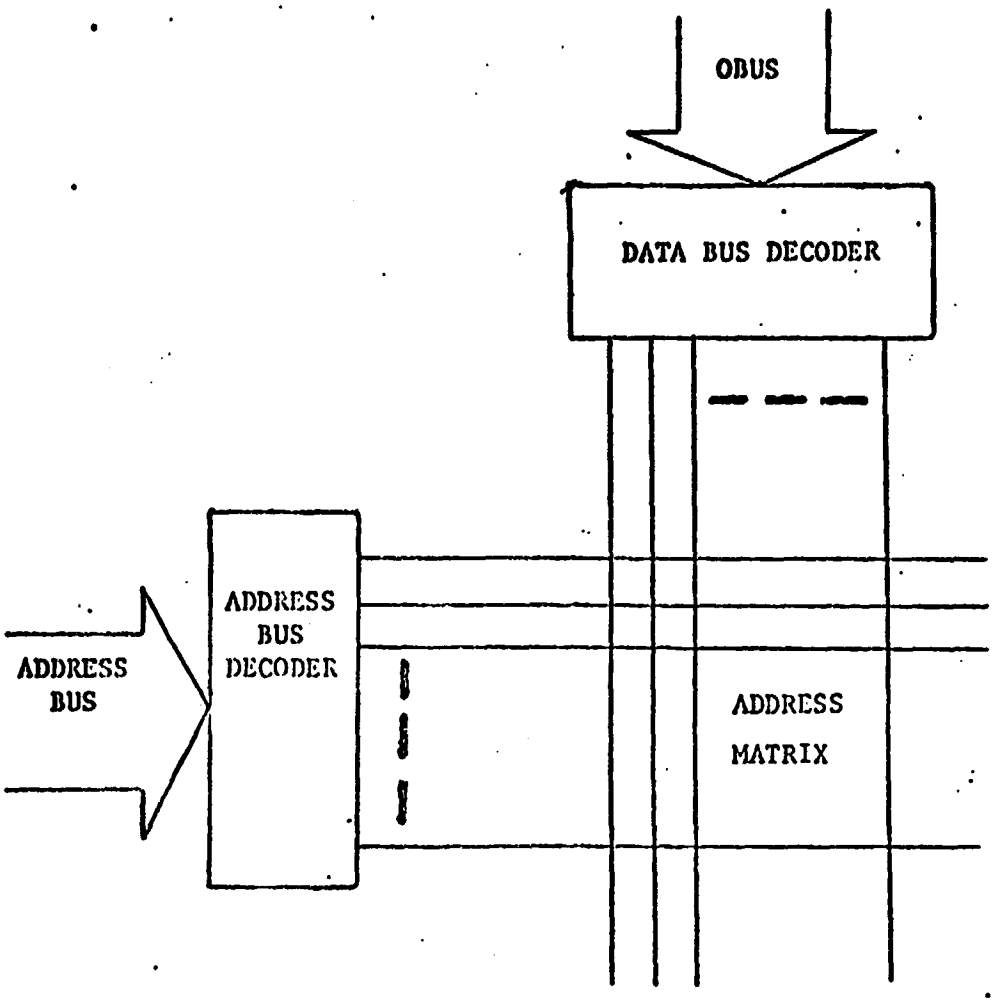


Fig. 3.21 Addressing Matrix
(Courtesy of James Burnell)

Presented in Figures 3.22, 3.23 and 3.24 are the segments of the μW which are required to control the IEU and the system hardware. In addition, these figures and Figure 3.25 contain the tables used to microprogram the packet switch.

3.2.3.1 ALU Source Fields

The AMD 2903 ALU chip provides the ALU with two operand inputs labeled R and S. A 2-1 MUX supplies the R operand input with data from either the A output from the internal register file or the external A-Direct-Data (DA) input. Since no class of processors utilizes the A register file, the DA input is permanently selected. External to the ALU, a 2-1 mux selects either the μW operand data or the data held in the IBUS Latch, and supplies the selected source to the DA input. This mux is controlled by the R SOURCE field in the μW.

The ALU's S input has three sources: The B output of the internal register file, the B-Direct-Data (DB) input and the internal Q register. Addresses for the B register file are supplied to the AM2903 via the external B SOURCE mux. This mux has the harwired addresses for each scratchpad register used as its inputs. The B ADDRESS field in μW controls this mux. Data supplied to the DB input arrives from the processor's polling circuit. Both the B register file output and the DB input are tristated. Tristate bus control is essential since both inputs share the same internal data bus. This data bus forms one of the two inputs to an internal 2-1 mux. The other mux input is the output from the Q register.

Ø-15	16-19		2Ø	21-24	25	26-3Ø	31	32	33	34	35	36	37
IMMEDIATE OPERAND	ALU SOURCE		B	ALU	CARRY	ALU	BUS	OUTPUT	DECODER	WRITE	ELIST	ELIST	POLLER
	R SOURCE	S SOURCE	ADDRESS	FUNCTION	In'	DESTINATION	LATCH	ENABLE	ENABLE	ENABLE	READ	UPDATE	RESET

MNEMONIC FIELDS

D_{15} — — — $D_Ø$	$\overline{EA}$, I_9, $\overline{OE_B}$, $I_Ø$	BAS	$I_4 \cdots I_1$	C_n	$\overline{IEN}$ $I_8 \cdots I_5$	ALE	$\overline{OE_Y}$	DE	$\overline{WE}$	$\overline{A\text{-READ}}$	A-UPDATE	A-RESET

CONTROL BITS

				ALU SOURCE	
$\overline{EA}$	I_9	$\overline{OE_B}$	$I_Ø$	R	S
1	Ø	Ø	Ø	µW	Reg @ B
1	Ø	X	1	µW	Q
1	Ø	1	Ø	µW	Poller Port
1	1	Ø	Ø	IBUS	Reg @ B
1	1	X	1	IBUS	Q
1	1	1	Ø	IBUS	Poller Port

X - DON'T CARE

BAS	B SOURCE
Ø	SCRATCH 1
1	SCRATCH 2

ALE	BUS LATCH
Ø	NONE
1	ADDRESS LATCH

Fig. 3.22 Input Processor IEU µW Control Fields

Ø – 15	16 – 19		2Ø	21-24	25	26-3Ø	31-32	33	34	35	36-38	39	4Ø	41
IMMEDIATE OPERAND	ALU SOURCE		B ADDRESS	ALU FUNCTION	CARRY IN	ALU DESTINATION	BUS LATCH	READ	OUTPUT ENABLE	WRITE ENABLE	OUTPUT QUEUE LIST CONTROL	POWER RESET	HEADER WRITE	OSW CONTROL
	R SOURCE	S SOURCE												

MNEMONIC FIELDS

D_{15} – – – $D_{\emptyset}$	$\overline{EA}$, I_9, $\overline{OE_B}$, $I_{\emptyset}$	BAS	$I_4 \cdots I_1$	C_n	$\overline{IEN}$, $I_8 \cdots I_5$	ALE RALE	$\overline{R}$	$\overline{OE_Y}$	$\overline{WE}$	B-LOAD B-REQUEST B-RELEASE	B-RESET	B-STORE	B-EMPTY

CONTROL BITS

				ALU SOURCE	
$\overline{EA}$	I_9	$\overline{OE_B}$	$I_{\emptyset}$	R	S
1	Ø	Ø	Ø	µW	Reg @ B
1	Ø	X	1	µW	Q
1	Ø	1	Ø	µW	Poller Port
1	1	Ø	Ø	I BUS	Reg @ B
1	1	X	1	I BUS	Q
1	1	1	Ø	I BUS	Poller Port

X - DON'T CARE

RALE	ALE	BUS LATCH
Ø	Ø	NONE
Ø	1	ADDRESS LATCH
1	Ø	ROM ADDRESS LATCH

BAS	B SOURCE
Ø	SCRATCH 1
1	SCRATCH 2

Fig. 3.23 Routing Processor IEU µW Control Fields

Ø - 15	16 - 19		2Ø	21 - 24	25	26 - 3Ø	31	32	33	34	35	36 - 37	38 - 39	4Ø - 41
IMMEDIATE OPERAND	ALU SOURCE		B ADDRESS	ALU FUNCTION	CARRY IN	ALU DESTINATION	BUS LATCH	DECODER ENABLE	WRITE ENABLE	OUTPUT ENABLE	POWER RESET	OSM CONTROL	OUTPUT QUEUE LIST CONTROL	E LIST CONTROL
	R SOURCE	S SOURCE												

MNEMONIC FIELDS

D_{15} - - - D_0	$\overline{EA}$, I_9, $\overline{OE_B}$, I_8	BAS	$I_4 \cdots I_2$	C_n	$\overline{IEN}$, $I_8 \cdots I_5$	ALE	DE	$\overline{WE}$	$\overline{OE_Y}$	C-RESET	C-SERVICE C-IDLE	C-REQUEST C-RELEASE	C-UPDATE C-WRITE

CONTROL BITS

99

				ALU SOURCE	
$\overline{EA}$	I_9	$\overline{OE_B}$	I_8	R	S
1	Ø	Ø	Ø	µW	Reg @ B
1	Ø	X	1	µW	Q
1	Ø	1	Ø	µW	Caller Port
1	1	Ø	Ø	IBUS	Reg @ B
1	1	X	1	IBUS	Q
1	1	1	Ø	IBUS	Caller Port

X - DON'T CARE

BAS	B SOURCE
Ø	SCRATCH 1
1	SCRATCH 2

ALU	BUS LATCH
Ø	PASS
1	LATCH

Fig. 3.24 Output Processor IEU µW Control Fields

TABLE 1. ALU OPERAND SOURCES

$\overline{E_A}$	I_0	$\overline{OE_B}$	ALU Operand R	ALU Operand S
L	L	L	RAM Output A	RAM Output B
L	L	H	RAM Output A	DB_{0-3}
L	H	X	RAM Output A	Q Register
H	L	L	DA_{0-3}	RAM Output B
H	L	H	DA_{0-3}	DB_{0-3}
H	H	X	DA_{0-3}	Q Register

L = LOW H = HIGH X = Don't Care

TABLE 2. ALU FUNCTIONS

I_4	I_3	I_2	I_1	Hex Code	ALU Functions	
L	L	L	L	0	I_0 = L	Special Functions
					I_0 = H	F_i = HIGH
L	L	L	H	1	F = S Minus R Minus 1 Plus C_n	
L	L	H	L	2	F = R Minus S Minus 1 Plus C_n	
L	L	H	H	3	F = R Plus S Plus C_n	
L	H	L	L	4	F = S Plus C_n	
L	H	L	H	5	F = S Plus C_n	
L	H	H	L	6	F = R Plus C_n	
L	H	H	H	7	F = R Plus C_n	
H	L	L	L	8	F_i = LOW	
H	L	L	H	9	F_i = R_i AND S_i	
H	L	H	L	A	F_i = R_i EXCLUSIVE NOR S_i	
H	L	H	H	B	F_i = R_i EXCLUSIVE OR S_i	
H	H	L	L	C	F_i = R_i AND S_i	
H	H	L	H	D	F_i = R_i NOR S_i	
H	H	H	L	E	F_i = R_i NAND S_i	
H	H	H	H	F	F_i = R_i OR S_i	

L = LOW H = HIGH i = 0 to 3

TABLE 3. ALU DESTINATION CONTROL FOR I_0 OR I_1 OR I_2 OR I_3 OR I_4 = HIGH, $\overline{IEN}$ = LOW.

I_0	I_1	I_2	I_3	I_4	Hex Code	ALU Shifter Function	SIO_3 Most Sig. Slice	SIO_3 Other Slices	Y_3 Most Sig. Slice	Y_3 Other Slices	Y_2 Most Sig. Slice	Y_2 Other Slices	Y_1	Y_0	SIO_0	$\overline{Write}$	Q Reg. & Shifter Function	CIO_3	CIO_0
L	L	L	L	L	0	$A_i \oplus F_{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	L	Input	H-Z	H-Z
L	L	L	L	H	1	$Log\ 2^{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	L	Log 2i-0	Input	O_0
L	L	H	L	L	2	$Arith\ 2^{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	L	Log 2i-0	Input	O_0
L	L	H	L	H	3	$Log\ 2^{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	L	Input	H-Z	H-Z
L	H	L	L	L	4	F_{i-1}	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	L	Input	H-Z	H-Z
L	H	L	L	H	5	F_{i-1}	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	L	Log 2i-0	Input	O_0
L	H	H	L	L	6	$Arith\ 2^{i-1}$	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	L	Log 2i-0	Input	O_0
L	H	H	L	H	7	$Log\ 2^{i-1}$	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	L	Input	H-Z	H-Z
H	L	L	L	L	8	$A_i \oplus F_{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	H	Input	H-Z	H-Z
H	L	L	L	H	9	$Log\ 2^{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	H	Log 2i-0	Input	O_0
H	L	H	L	L	A	$Arith\ 2^{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	H	Log 2i-0	Input	O_0
H	L	H	L	H	B	$Log\ 2^{i-1}$	Input	Input	$S0_i$	$S0_i$	$S0_i$	$S0_i$	F_i	F_i	F_i	H	Input	H-Z	H-Z
H	H	L	L	L	C	F_{i-1}	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	H	Input	H-Z	H-Z
H	H	L	L	H	D	F_{i-1}	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	H	Log 2i-0	Input	O_0
H	H	H	L	L	E	$Arith\ 2^{i-1}$	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	H	Log 2i-0	Input	O_0
H	H	H	L	H	F	$Log\ 2^{i-1}$	Input	Input	F_i	F_i	F_i	F_i	F_i	F_i	F_i	H	Input	H-Z	H-Z

Fig. 3.25 ALU Control Fields

TABLE 1. ALU OPERAND SOURCES

$\overline{E_A}$	I_0	$\overline{OE_B}$	ALU Operand R	ALU Operand S
L	L	L	RAM Output A	RAM Output B
L	L	H	RAM Output A	DB_{0-3}
L	H	X	RAM Output A	Q Register
H	L	L	DA_{0-3}	RAM Output B
H	L	H	DA_{0-3}	DB_{0-3}
H	H	X	DA_{0-3}	Q Register

L = LOW H = HIGH X = Don't Care

TABLE 2. ALU FUNCTIONS

I_4	I_3	I_2	I_1	Hex Code	ALU Functions	
L	L	L	L	0	I_0 = L	Special Functions
					I_0 = H	F_i = HIGH
L	L	L	H	1	F = S Minus R Minus 1 Plus C_n	
L	L	H	L	2	F = R Minus S Minus 1 Plus C_n	
L	L	H	H	3	F = R Plus S Plus C_n	
L	H	L	L	4	F = S Plus C_n	
L	H	L	H	5	F = S Plus C_n	
L	H	H	L	6	F = R Plus C_n	
L	H	H	H	7	F = R Plus C_n	
H	L	L	L	8	F_i = LOW	
H	L	L	H	9	F_i = R, AND S_i	
H	L	H	L	A	F_i = R, EXCLUSIVE NOR S_i	
H	L	H	H	B	F_i = R, EXCLUSIVE OR S_i	
H	H	L	L	C	F_i = R, AND S_i	
H	H	L	H	D	F_i = R, NOR S_i	
H	H	H	L	E	F_i = R, NAND S_i	
H	H	H	H	F	F_i = R, OR S_i	

L = LOW H = HIGH i = 0 to 3

TABLE 3. ALU DESTINATION CONTROL FOR I_0 OR I_1 OR I_2 OR I_3 OR I_4 = HIGH, $\overline{IEN}$ = LOW.

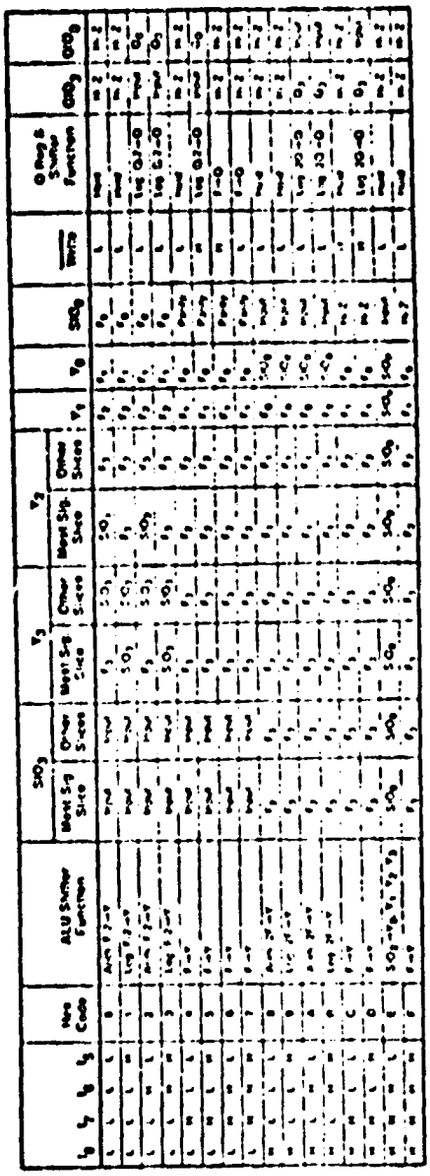


Fig. 3.25 ALU Control Fields

Selection of the S input is made by the μW S SOURCE control field. This μW field controls the 2-1 mux and the tristate logic.

3.2.3.2 ALU Function Fields

The selection of an ALU arithmetic function or logical operation is determined by the μW ALU Function field.

3.2.3.3 ALU Destination Fields

Internally, the 2903's ALU output is sent to both the register file's DATA IN input and the Q register (via the Q shifter). The ALU's output is also available to the Output Bus (OBUS) via an internal tri-state buffer. The ALU Destination field can direct the ALU output to any or all of these locations.

3.2.3.4 Bus Control Fields

In order to hold address data stable, the OBUS is supplied to two address latches: The Address Latch and the ROM Address Latch (Routing Processor Only). These latches are enabled by the Bus Latch μW field in conjunction with the Phase 2 clock (see 3.2.5).

The various μW Read and Write fields control data transfers between the processors and external hardware.

3.2.3.5 System Hardware Control Fields

The System Hardware Control Fields consist of various control bits used to activate system hardware operations.

These signals are sent directly to the hardware since they do not control IEU operations. However, they usually act in conjunction with the processor, often helping to speed up processor tasks. They also may direct hardware operations which carry out independent tasks. Thus, use of these special control bits has improved system throughput.

3.2.4 The Microprogram Control Unit

The function of the Microprogram Control Unit is twofold: It must control the execution of the processor's software and it must supply the microprogram's control signals to the IEU and the system hardware. A diagram of this unit is given in Figure 3.26. The MCU consists of an AMD 2911 microprogram sequencer, jump control logic (implemented by a Programmable Logic Array (PLA)) [3], a pipeline register and the microprogram memory. A block diagram of the AMD 2911 chip is presented in Figure 3.27. This device generates the μprogram counter value used to control the execution sequence of the processor's microprogram. Next address selection provides the MCU with one of the two possible next addresses. Either the μprogram counter or the address in Jump Address field of the μW is supplied to the address lines of the microprogram memory. The PLA Jump Control Logic determines this selection. Inputs to the PLA Jump Control Logic come from various system status signals and the Next Address Select μW field. Figure 3.28 contains the Next Address Select field and the Jump Address field. The Jump Control Logic Function for each class of processor is given in Figure 3.29.

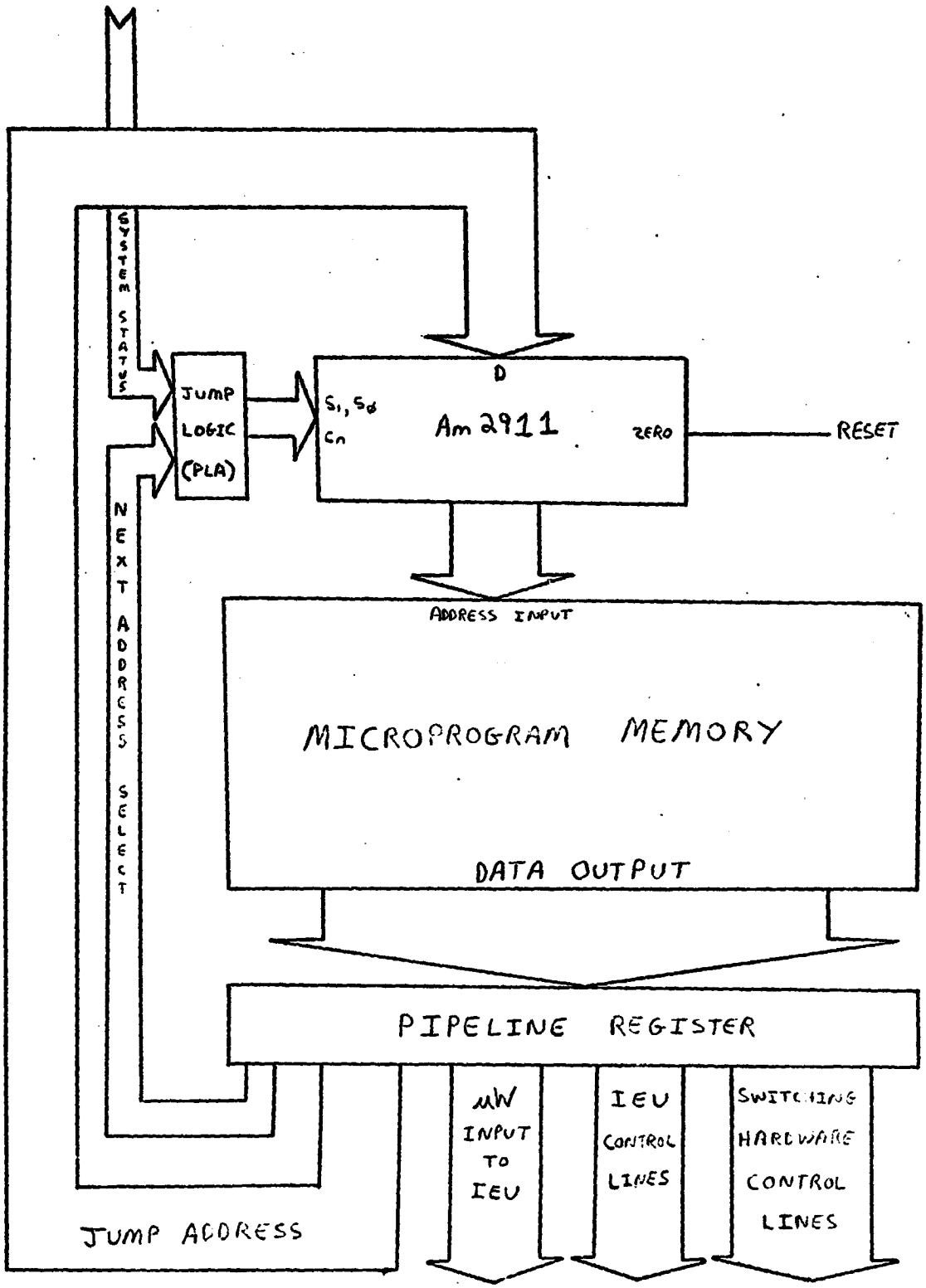


Fig. 3.26 Microprogram Control Unit

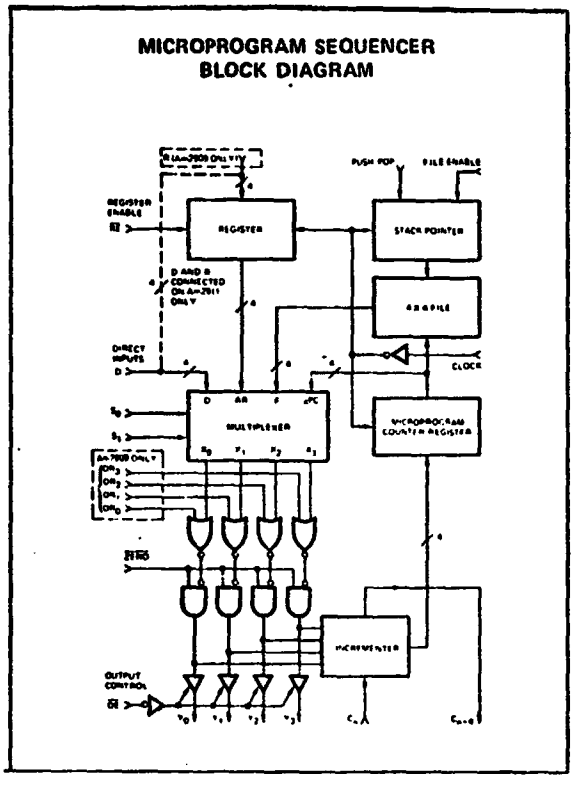


Fig. 3.27 Am 2911 Microprogram Sequencer

Mnemonic Fields	Next Address Select	Jump Address

Control Bits

38-39

N_1	$N_\emptyset$	Next Address
Ø	Ø	μPC + 1
Ø	1	unconditional Jump
1	X	Jump on IBSR-A =Ø

INPUT PROCESSOR

40-43

Jump Address
$JA_3, JA_2, JA_1, JA_\emptyset$

42-44

N_1	$N_\emptyset$	J	Next Address
Ø	Ø	Ø	μPC + 1
Ø	Ø	1	unconditional Jump
Ø	1	X	Jump on NEW-B =Ø
1	Ø	X	Jump on STATUS -B=1
1	1	X	Jump on IDLE-B=1

ROUTING PROCESSOR

45-48

Jump Address
$JA_3, JA_2, JA, JA_\emptyset$

42-44

N_1	$N_\emptyset$	J	Next Address
Ø	Ø	Ø	μPC + 1
Ø	Ø	1	unconditional Jump
Ø	1	X	Jump on SERVICE-C=Ø
1	Ø	X	Jump on STATUS -C=1
1	1	X	Jump on EMPTY-C=Ø

OUTPUT PROCESSOR

45-49

Jump Address
JA_4, JA_3, JA_2, JA_1 $JA_\emptyset$

Fig. 3.28 MCU μW Control Fields

INPUTS			OUTPUTS				
IBSR-A	N_1	$N_\emptyset$	$\overline{FE}$	C_n	S_1	$S_\emptyset$	ADDRESS SOURCE
X	Ø	Ø	1	1	Ø	Ø	μPC + 1
X	Ø	1	1	Ø	1	1	Jump Address
Ø	1	X	1	Ø	1	1	Jump Address
1	1	X	1	1	Ø	Ø	μPC + 1

Input Processor Jump Control Logic Function

INPUTS						OUTPUTS				
NEW -B	STATUS -B	IDLE -B	N_1	$N_\emptyset$	J	$\overline{FE}$	C_n	S_1	$S_\emptyset$	ADDRESS SOURCE
X	X	X	Ø	Ø	Ø	1	1	Ø	Ø	μPC + 1
X	X	X	Ø	Ø	1	1	Ø	1	1	Jump Address
Ø	X	X	Ø	1	X	1	Ø	1	1	Jump Address
1	X	X	Ø	1	X	1	1	Ø	Ø	μPC + 1
X	Ø	X	1	Ø	X	1	1	Ø	Ø	μPC + 1
X	1	X	1	Ø	X	1	Ø	1	1	Jump Address
X	X	Ø	1	1	X	1	1	Ø	Ø	μPC + 1
X	X	1	1	1	X	1	Ø	1	1	Jump Address

Background Processor Jump Control Logic Function

INPUTS						OUTPUTS				
SER-VICE-C	STATUS -C	EMPTY -C	N_1	$N_\emptyset$	J	$\overline{FE}$	C_n	S_1	$S_\emptyset$	ADDRESS SOURCE
X	X	X	Ø	Ø	Ø	1	1	Ø	Ø	μPC + L
X	X	X	Ø	Ø	1	1	Ø	1	1	Jump Address
Ø	X	X	Ø	1	X	1	Ø	1	1	Jump Address
1	X	X	Ø	1	X	1	1	Ø	Ø	μPC + 1
X	Ø	X	1	Ø	X	1	1	Ø	Ø	μPC + 1
X	1	X	1	Ø	X	1	Ø	1	1	Jump Address
X	X	Ø	1	1	X	1	Ø	1	1	Jump Address
X	X	1	1	1	X	1	1	Ø	Ø	μPC + 1

Output Processor Jump Control Logic Function

Fig. 3.29 Jump Control Logic Functions

The output of the AMD 2911 can be unconditionally reset to zero. This allows the system to initialize program execution whenever required. Table 3.2 contains the μW widths for each class of processors. Since some bits in the μW remain a constant logic value, they can be hardwired. This reduces the actual Microprogram Memory (ROM) widths.

3.2.5 Processor Timing

A two-phase clock drives the processors. This clock controls the timing of internal and external data transfers. Figure 3.30 presents the waveforms and significant timing events. Phase 1 latches the internal data of the 2903 ALU and the 2911 microprogram sequences. Phase 2 is required to stabilize data in the IEU hardware that is external to the ALU. In addition, this clock phase is used to latch data and address information required for external data transfers to I/O ports and memory. Each clock cycle has a period of 120 nanoseconds which yields a maximum clock frequency of 8.33 mHz [3].

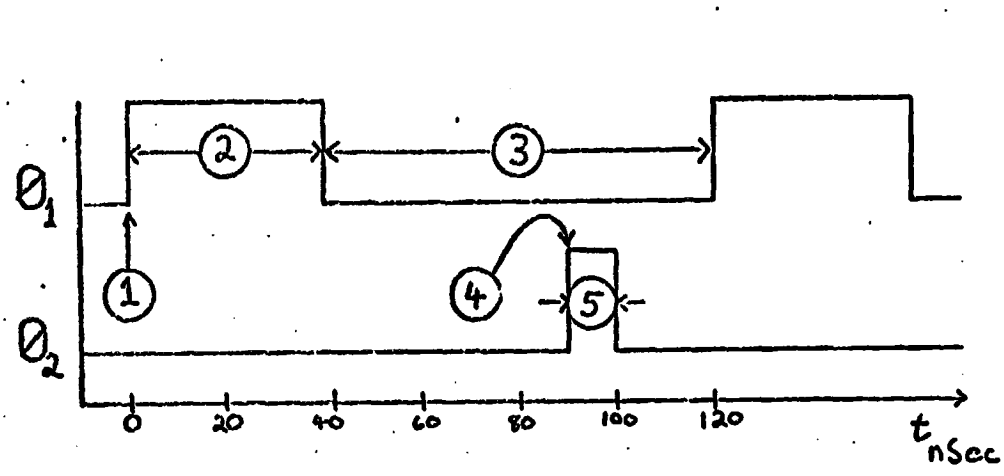
3.3 The System Software

Each class of processor executes a unique software routine. The three different routines are: The Input Service Routine, the Routing Service Routine and the Output Service Routine. A detailed explanation of each routine's function is presented next.

Microprogram Word Control Fields	Bit Format		
	Input Processor	Routing Processor	Output Processor
Immediate Operand	Ø-15	Ø-15	Ø-15
Source Control	16-2Ø	16-2Ø	16-2Ø
Function Control	21-25	21-25	21-25
Destination Control	26-30	26-30	26-30
Bus Latch Control	31	31-32	31
IEU Hardware Control	32-34	33-35	32-34
System Hardware Control	35-37	36-41	35-41
Next Address select	38-39	42-44	42-44
Jump Address	40-43	45-48	45-49
TOTAL uW WIDTH	44 Bits	49 Bits	50 Bits

Processor	Hardwired uW Control Signal		uW ROM WIDTH
	Logic Ø	Logic 1	
Input	$\overline{IEN}$, C_n, I_8, I_5, I_4 JA_3-$JA_Ø$	I_7, $\overline{EA}$	33 Bits
Routing	$\overline{IEN}$, C_n, I_8, I_5	I_7, $\overline{EA}$	43 Bits
Output	$\overline{IEN}$, C_n, I_8, I_5, I_4 JA_3, JA_2	I_7, $\overline{EA}$	41 Bits

Table 3.2 Microprogram Word Bit Divisions



① a) Current instruction is latched into pipeline register.

b) Data is clocked into Q register.

② a) A and B latches internal to 2903 are open.

b) IBUS latch is open.

c) $\overline{\text{READ}}$ line is low during this time in a read operation.

③ a) A and B latches, IBUS latch are closed.

b) ALU output is stable.

c) $\overline{\text{WE}}$ is low if storing into register file.

④ Address is latched on this edge during an address generation operation.

⑤ If Write microprogram word bit is high, $\overline{\text{WRITE}}$ goes low during this pulse.

Fig. 3.30 Processor Clock Waveforms
(Courtesy of James Burnell)

3.3.1 The Input Service Routine

The Input Service Routine is executed by the Input Processor. Shown in Figure 3.31 is the flowchart of this software routine. This routine is sense-loop driven. The Input Processor loops on a status bit which is controlled by the Input Polling Circuit. When the poller finds a full input buffer, it updates the sense-loop status bit. Once the processor leaves the loop, it fetches the address of the full input buffer. This address is supplied by the polling circuit. Next, the processor clears the buffer's DAV flag and restarts the poller. Restarting the poller, before service is complete allows the poller to find the next full buffer before the processor returns to the sense-loop. This scheme reduces processor idleness due to poller scan time.

The Input Processor then fetches the address of a free data path in the Input Switching Network. This address is supplied by the Input Data Path Status Port. The ELIST is accessed next. Using this list, the Input Processor fetches the address of a free shift register in the array. After obtaining the address stored in the location selected by the EPTRØ index pointer, the Input Processor increments the EPTRØ. EPTRØ now points to the next empty shift register address stored in ELIST. Using the three addresses mentioned above, the Input Processor links the full input buffer to the empty shift register via the free data path. Upon completion of the link, the Input Processor initiates the packets transfer into the array. The Input Processor then returns to the sense loop.

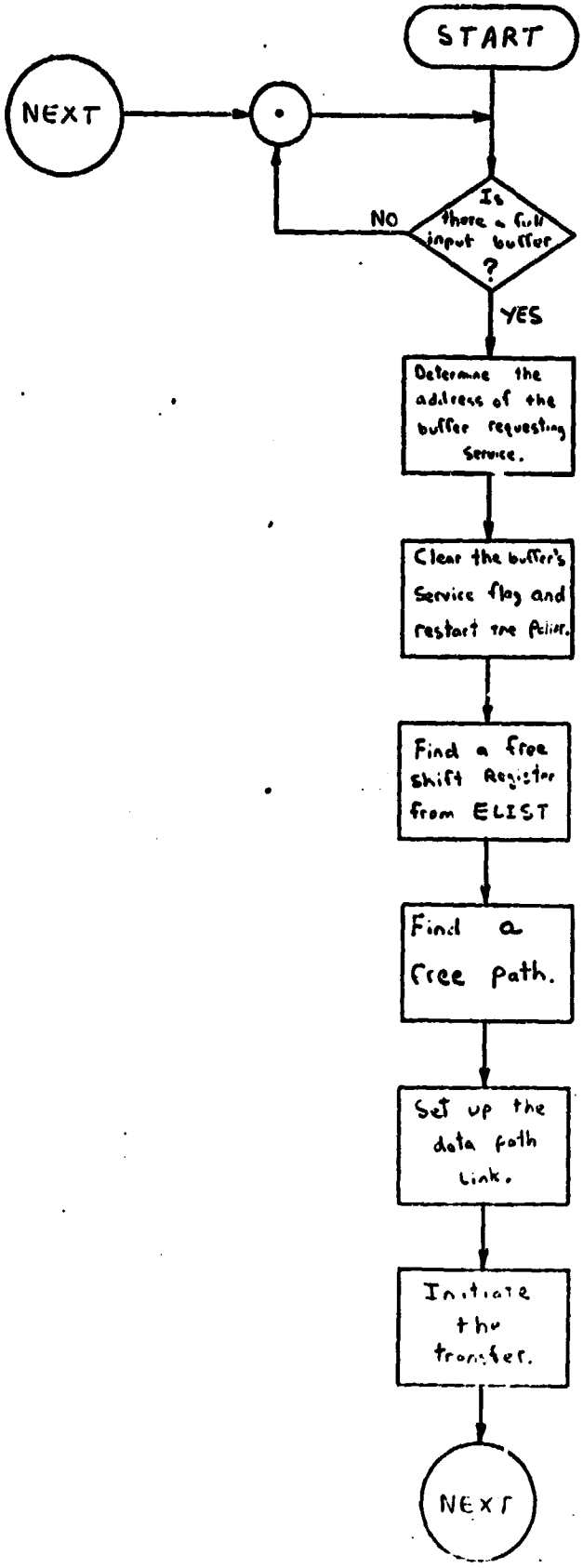


Fig. 3.31 Input Service Routine Flowchart

A listing of this routine is given in Figure 3.32. Since the instruction set of each processor is custom tailored, no standard computer language exists to describe the packet switch's software. In order to document the software, a simple format is used to code each line of software:

<1st operand><operation><2nd operand> → <destination>.

In the listing, instructions performing a single task are grouped together, followed by a comment explaining their function. Concurrent task execution is noted by ";". In addition, the μP Address Code is listed next to the instruction which generates that particular control signal. This is done to help explain how the software interfaces with the system hardware. Each line of code listed requires 120 nanoseconds of execution time.

3.3.2 The Routing Service Routine

Execution of the Routing Service Routine is carried out by the Routing Processor. The flowchart of this software routine is illustrated in Figure 3.33. This routine is sense loop driven. The processor loops waiting for the Shift Register Polling Circuit to indicate that a newly arrived packet has been found in the array. When a new packet is found, the Routing Processor leaves the loop and fetches the packet's array address from the poller. Using this address, the Routing Processor fetches the packet's syndrome from the Syndrome Generator. This syndrome is latched into the address input of the Syndrome Decoder Rom. Simultaneously, the shift

```
INPUT: If IBSR-A = Ø, JMP TO INPT
                                        *Is there an input buffer
                                         requesting Service?
                                         NO: Loop @ INPUT.
       Input Polling Port → Q
                                        *YES: Input the address
                                         of the buffer requesting
                                         service.
       [ELIST]@EPTRØ → Scratch 1; Reset Poller
                                        *Find a free shift regis-
                                         ter, clear IBSR-A and
                                         restart poller.
       Input Data Path Status Port Address → Address Latch (μP1-A)
       Data Path Busy Status Port → Q; Update EPTRØ
                                        *Find a free data path
                                         and increment EPTRØ.
       Scratch 2+Data Path Latch A Base Address→Address Latch (μP2-A)
       Q → Data Path mux select Latch A(D)
                                        *Link the input buffer
                                         to the data path.
       Scratch 2+Data Path Latch B Base Address→Address Latch (μP3-A)
       Scratch 1 → Data Path Demux select Latch B(D)
                                        *Link the empty shift
                                         register to the data
                                         path.
       Data Path Transmit Control Address → Address Latch
       Scratch 2 → Data Bus Decoder (M1-A); Jump to INPUT
                                        *Start data transfer
                                         and return to the
                                         sense loop.
```

Fig. 3.32 Input Service Routine

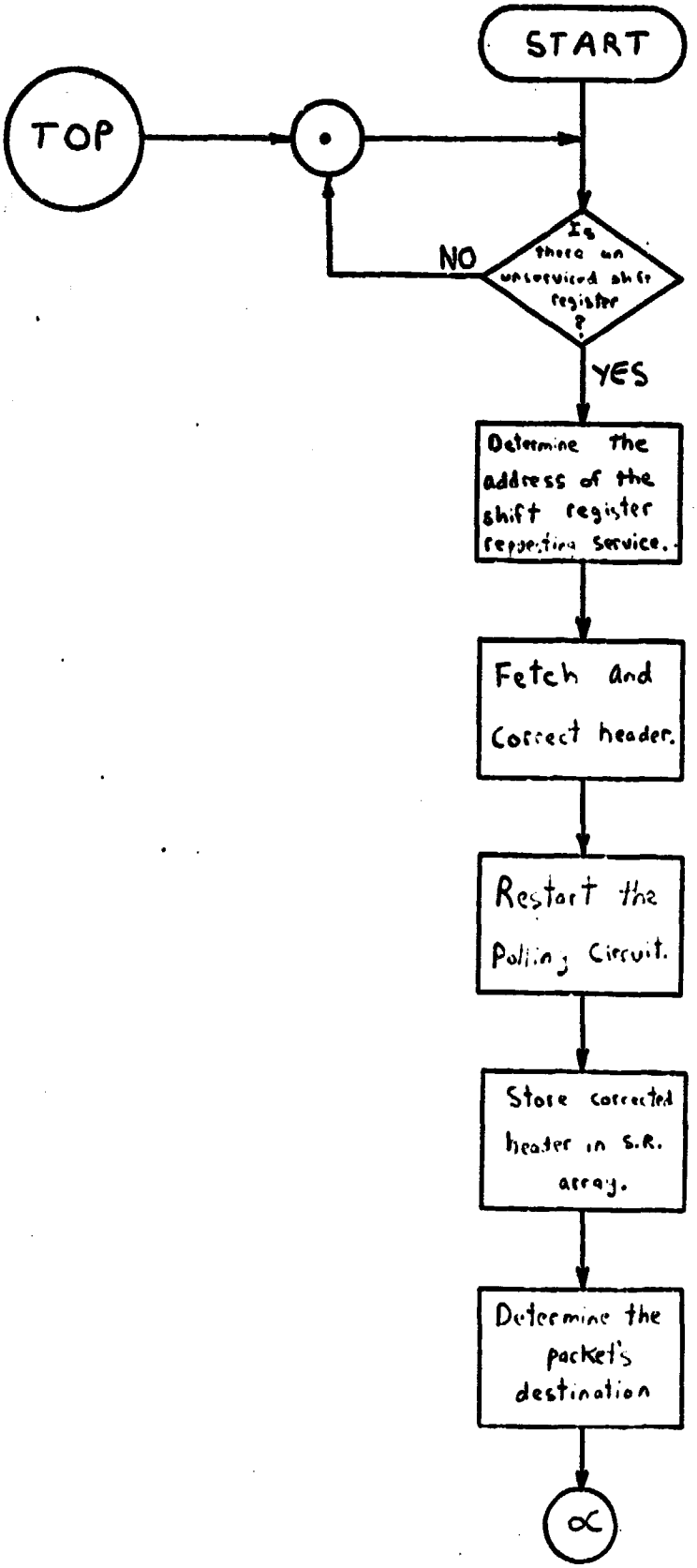


Fig. 3.33 Packet Routing Service Routine Flowchart

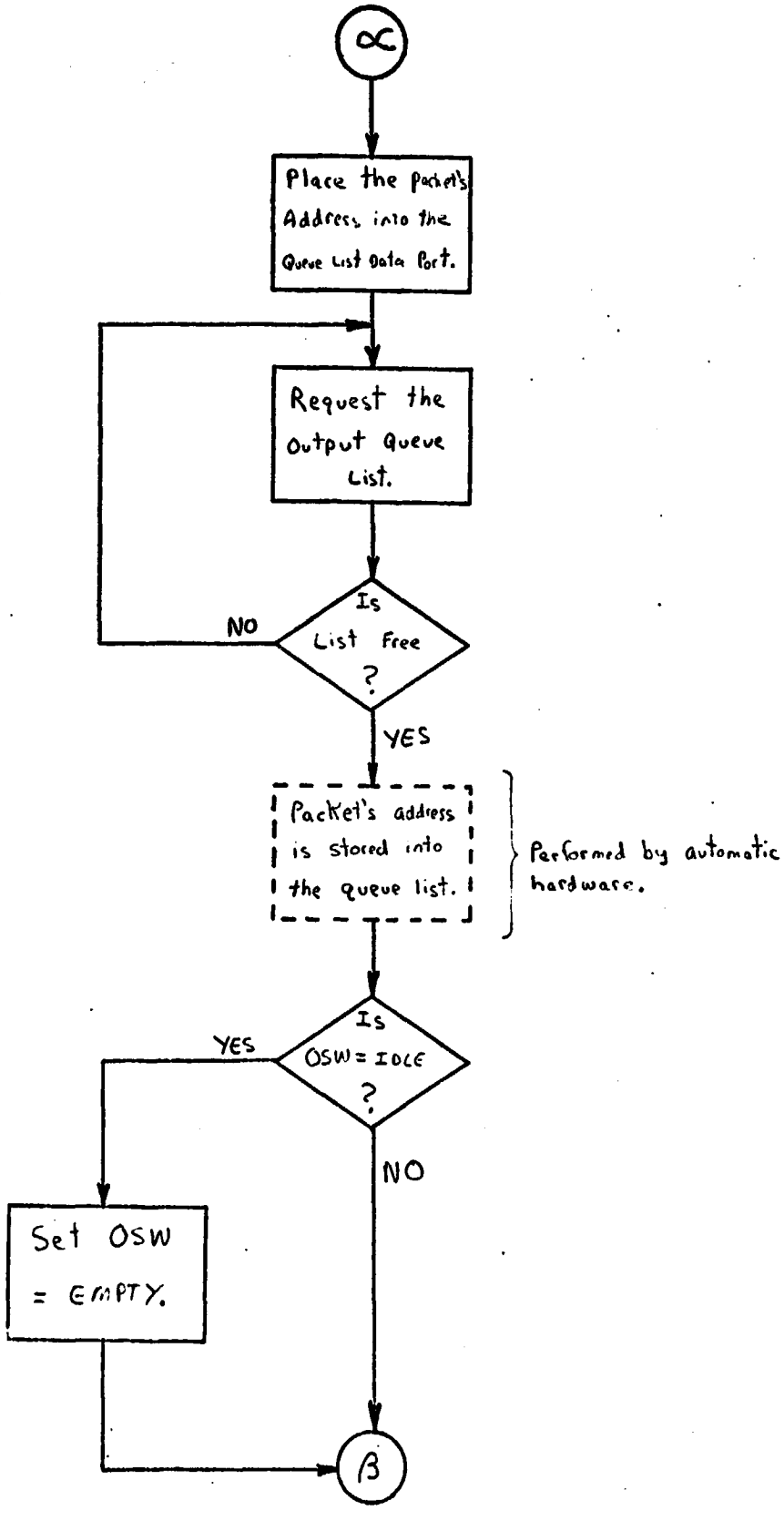


Fig. 3.33 Packet Routing Service Routine Flowchart, continued

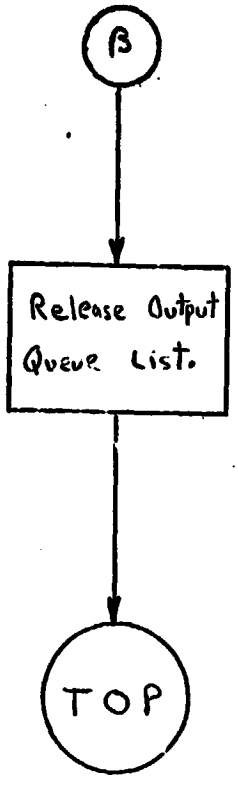


Fig. 3.33 Packet Routing Service Routine Flowchart, continued

register status flag is cleared and the poller is restarted. The output from the Syndrome Decoder Rom is exclusive-ORed with the packet's header. This operation yields the corrected header, which is stored back into the shift register.

Using the corrected header, the Routing Processor determines the packet's destination. In order to route the packet, the Routing Processor must place the packet's array address into the proper Output Queue List. As stated earlier, these lists are shared resources which have contention problems. Thus, they are regulated by hardware locks which permit access to only one processor at a time. Therefore, before accessing any list, the Routing Processor must request access. In order to minimize the time spent accessing these resources, the shift register address data is first placed into the Output Queue List Data Port. The Routing Processor then requests access to the selected queue list. If access is granted, the data in the data port is automatically strobed into the queue list at the location specified by the IPTR. This scheme permits the Routing Processor to move on to the next task rather than writing to the list.

Once the shift register address is placed into the proper queue list, the Routing Processor checks the associated OSW. If the OSW indicates that the corresponding buffer is not in the Idle state, the Routing Processor releases the Output Queue List. The signal generated to release the queue list also activates the IPTR update circuitry, which automatically increments the IPTR counter. After releasing the queue list, the Routing Processor returns to the sense loop.

Should the OSW indicate that the output buffer is in the Idle state, the Routing Processor updates the OSW. After this update, the OSW indicates that the buffer is now in the Empty state, waiting for Output Processor service. The Routing Processor then releases the queue list and returns to the sense loop.

If the selected Output Queue List is not available, the Routing Processor loops request service. This loop is called a SPIN LOCK since the processor spins on the hardware lock while waiting for the busy resource to be freed [5]. There exists an alternative locking scheme called the SUSPEND LOCK. This alternative scheme requires the processor to suspend the current task which needs the busy resource [5]. This task is temporarily put aside as the processor moves on to a new task. Implementation of this scheme was considered, but was abandoned. Several reasons led to the abandonment of the Suspend Lock:

1) The additional hardware and software required to suspend and resume jobs.
2) The next task selected may also require the busy resource.
3) The time wasted idling in the spin lock is far shorter than the time required to suspend and resume the execution of a job.
4) The possibility that no new task existed, resulting in wasted time as the processor suspended the only job available.

Thus, the Spin Lock is used in both the Routing Service Routine and the Output Service Routine. A listing of the Routing Service Routine is contained in Figure 3.34.

3.3.3 The Output Service Routine

The Output Service Routine is executed by the Output Processor. Figure 3.35 contains the flowchart of this routine. This routine is sense loop driven. The Output Processor remains in the loop until the Output Buffer Polling circuit locates an empty output buffer. When the poller finds an empty buffer, it notifies the Output Processor by changing the sense loop status bit. Once the processor leaves the loop, it fetches the buffer's address from the poller. Using this address, the Output Processor selects the corresponding Output Queue List. Access to the queue list is then requested. As in the Routing Service Routine, a spin lock is implemented for queue list accesses. The Output Processor must spin on any activated queue list lock. Once access is granted, the Output Processor checks to see if the selected queue list is empty. If the queue list is empty, the Output Processor updates the buffer's OSW to indicate that the buffer is now in the Idle state. Then the Output Processor releases the queue list, restarts the poller and returns to the sense loop.

If the selected queue list is not empty, the Output Processor fetches the oldest packet address in the list. The associated OSW is changed to indicate that the output buffer is in the Busy state. After updating the OSW, the Output Processor releases the queue list and restarts the poller.

```
START: If NEW-B = Ø, Jmp to START

                                        *Is there a shift register
                                         requesting service?
                                         NO: Loop @ START.

SRS Polling Port → Scratch 1

                                        *YES: Input the address of
                                         the shift register.

Syndrome Generator Base Address+Scratch 1→Address Latch (μP1-B
Syndrome (R) → Decoder ROM Address Latch; Reset Poller

                                        *Fetch header Syndrome and
                                         send it to the Decoder ROM.
                                         Clear NEW-B and restart the
                                         poller.

Decoder ROM Address → Address Latch (μP2-B)
[Decoder ROM]@Syndrom(R) → Q

                                        *Fetch error word from ROM.

Header Base Address+Scratch 1→Address Latch (μP3-A)
ALU EXOR Q → Scratch 2, Header Port(R)

                                        *Correct the header.  Store it
                                         back into the S.R. Array and
                                         into Scratch 2.

Scratch AND Destination Mask → Q

                                        *Determine packet destination.

Q+Output Queue List Base Address→Address Latch (μP4-B)

                                        *Select the queue list and the
                                         OSW of the destination out-
                                         put buffer.

Scratch 1 → Output Queue List Data Port

                                        *Place the packet's S.R. Array
                                         address into Queue List Data
                                         Port.
```

Fig. 3.34 Packet Routing Service Routine

```
REQUEST: Request access to Queue List (N)

                                        *Request access to the Output
                                         Queue List selected.  If
                                         access is granted, the data
                                         in the port is automatically
                                         written into the queue list.

        If STATUS-B = 1, Jmp to REQUEST

                                        *If access is not granted,
                                         loop @ REQUEST.  Proceed
                                         otherwise.

        If OSW(N) = NOT IDLE, Jmp to END

                                        *Is output buffer idle?

        Set OSW(N)=EMPTY; Release Output Queue List; Jmp to START

                                        *YES: update OSW, release
                                         queue list and return to
                                         the sense loop.

    END: Release Output Queue List; Jmp to START

                                        *NO: Release queue list and
                                         return to the sense loop.
```

Fig. 3.34 Packet Routing Service Routine, continued

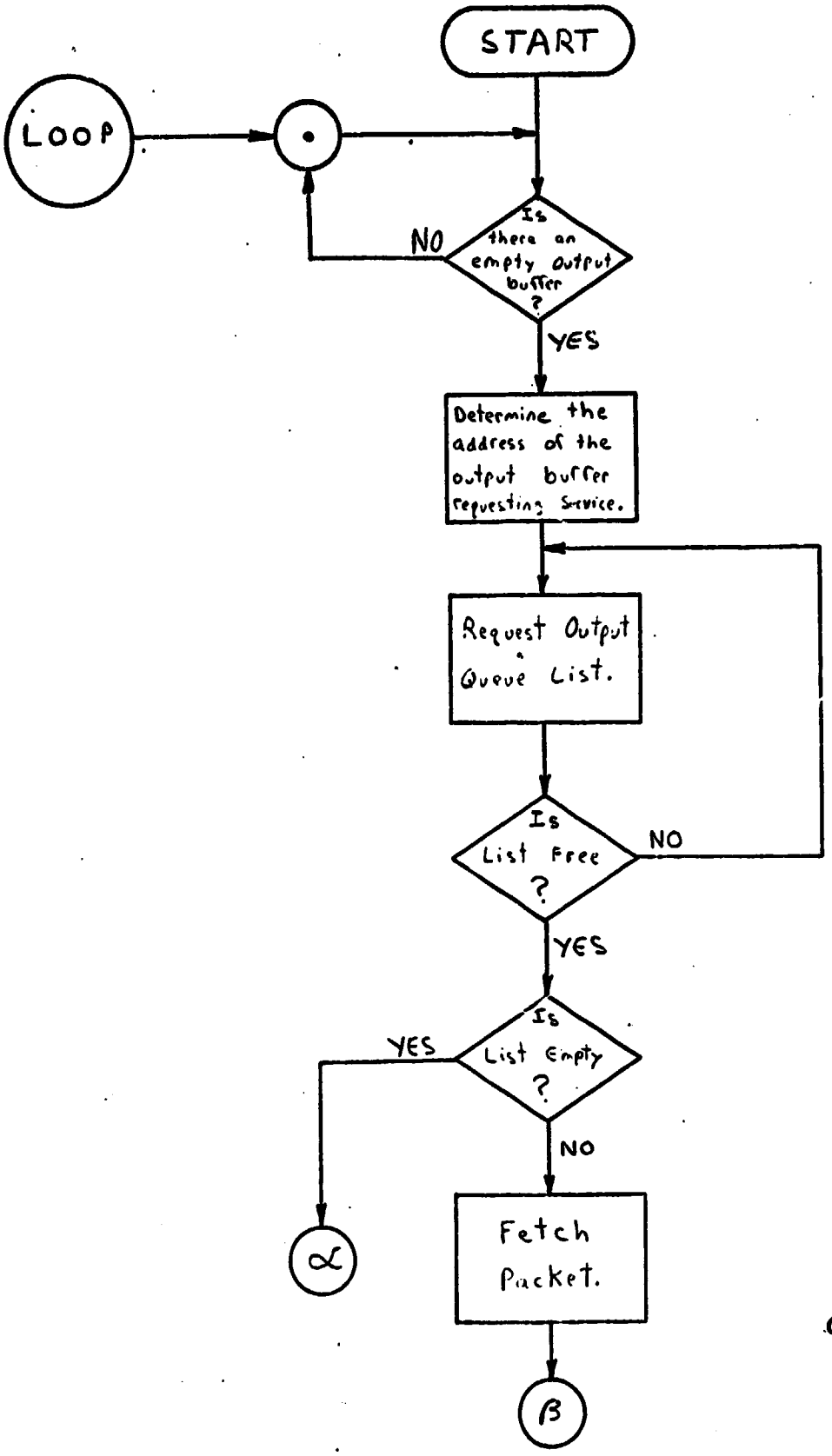


Fig. 3.35 Output Service Routine Flowchart

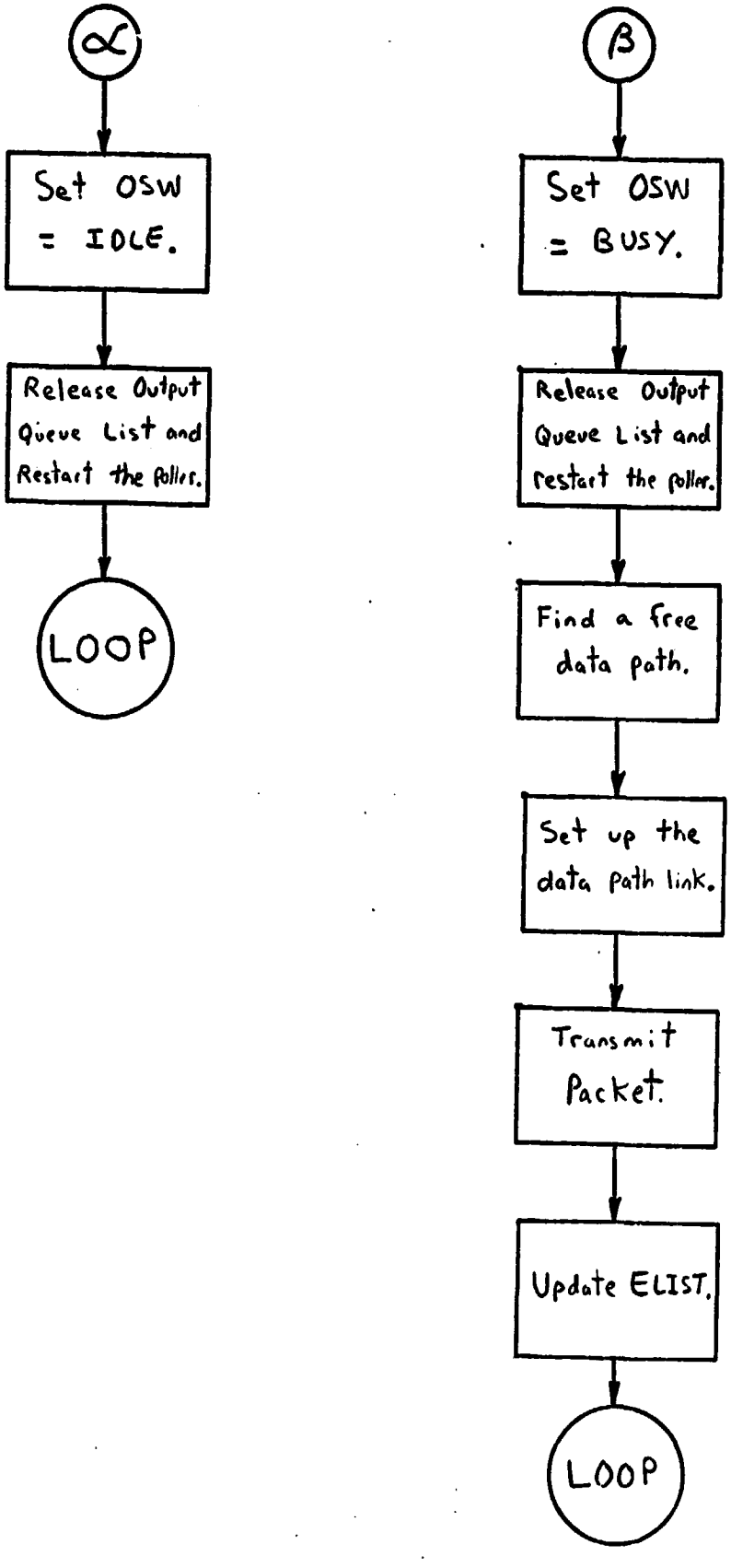


Fig. 3.35 Output Service Routine Flowchart, continued.

Next, the address of a free data path in the Output Switching Network is fetched from the Data Path Busy Port. The Output Processor links the shift register containing the packet awaiting transmission to the empty output buffer via the free data path. Once the data link is established, the Output Processor initiates the packet transfer and increments the ELIST index pointer EPTR1. EPTR1 now points to an unfilled location in ELIST. After this update, the address of the shift register containing the packet being transmitted is placed into ELIST at the location specified by EPTR1. The Output Processor then returns to the sense loop.

A listing of this routine is given in Figure 3.36.

```
OUTPUT: If SERVICE-C = Ø, Jmp to OUTPUT

                                        *Is there an output buffer
                                         requesting service?
                                         NO: Loop @ OUTPUT.

        Output Polling Port → Q

                                        *YES: Input the address of the
                                         buffer requesting service.

        Q+Output Queue List Base Address→Address Latch (μP1-C)

REQUEST: Request Queue List (N)

                                        *Select the buffer's output
                                         Queue List and OSW.  Then
                                         request access.

        If STATUS-C = 1, Jmp to REQUEST

                                        *Was access granted?
                                         NO: Request access again.

        If EMPTY-C = Ø, Jmp to IDLE

                                        *YES: Determine if the list
                                         is empty.  List Empty: Jump
                                         to IDLE.

        [Output Queue List(N)]@OPTR(N)→Scratch 1; Set OSW=BUSY;
        Release Output Queue List; Reset Poller

                                        *List Not Empty: Input the
                                         S.R.# which contains the
                                         packet to be transmitted.
                                         Then update the OSW, restart
                                         the poller and release the
                                         queue list.

        Output Data Path Status Port Address→Address Latch (μP2-C)
        Data Path Busy Status Port → Scratch 2

                                        *Find a free data path.

        Scratch 2+Data Path Latch A Base Address→Address Latch (μP3-C)
        Scratch 1 → Data Path MUX Select Latch A(D)

                                        *Link the shift register to
                                         the data path.
```

Fig. 3.36 Output Service Routine

```
        Scratch 2+Data Path Latch B Base Address→Address Latch (μP4-C)
        Q → Data Path Demux Select Latch B(D)

                                        *Link the output buffer to
                                         the data path.

        Data Path Transmit Control Address → Address Latch
        Scratch 2 → Data Bus Decoder (M1-C); Update EPTR1

                                        *Start Packet transfer and
                                         increment EPTR1.

        Scratch 1 → [ELIST]@EPTR1; Jmp to OUTPUT

                                        *Place S.R.# in the Empty
                                         S.R. List and return to the
                                         top of the program.

IDLE:   Set OSW=IDLE: Release Output Queue List; Reset Poller;
        Jmp to OUTPUT

                                        *Update OSW, release queue
                                         list, restart poller and re-
                                         turn to the top of the pro-
                                         gram.
```

Fig. 3.36 Output Service Routine, continued

4.0 THE MULTIPLE PROCESSOR DESIGN

With the three processor design complete, the next logical step in the expansion of the system is to include multiple processors in each processor class. The major incentive behind this idea is to increase the system throughput through the use of a multiprocessor architecture. However, two major problems must be overcome before this goal can be achieved. The two problems are contention and throughput-limiting functions. The solutions to these problems are presented as topics in this chapter since they shape the final system architecture. Also included in this chapter is the system architecture, the processors, hardware and software required for implementation, and the design trade-offs made. Many hardware components used in this design are exactly the same as those used in the three processor design and, therefore, are not presented in much detail. This chapter begins with an overview of the system architecture and its operation.

4.1 The System Architecture

The system architecture is shown in Figure 4.1. This new architecture is controlled by four classes of processors. The new class of processors and the system requirements that caused the additional workload division are discussed in 4.4. In order to examine the duties of each class of processors, a packet's transfer through the packet switch is traced.

The first function of the switch is to receive and to store each incoming packet. When a packet arrives, it is temporarily stored in an input buffer. An input buffer

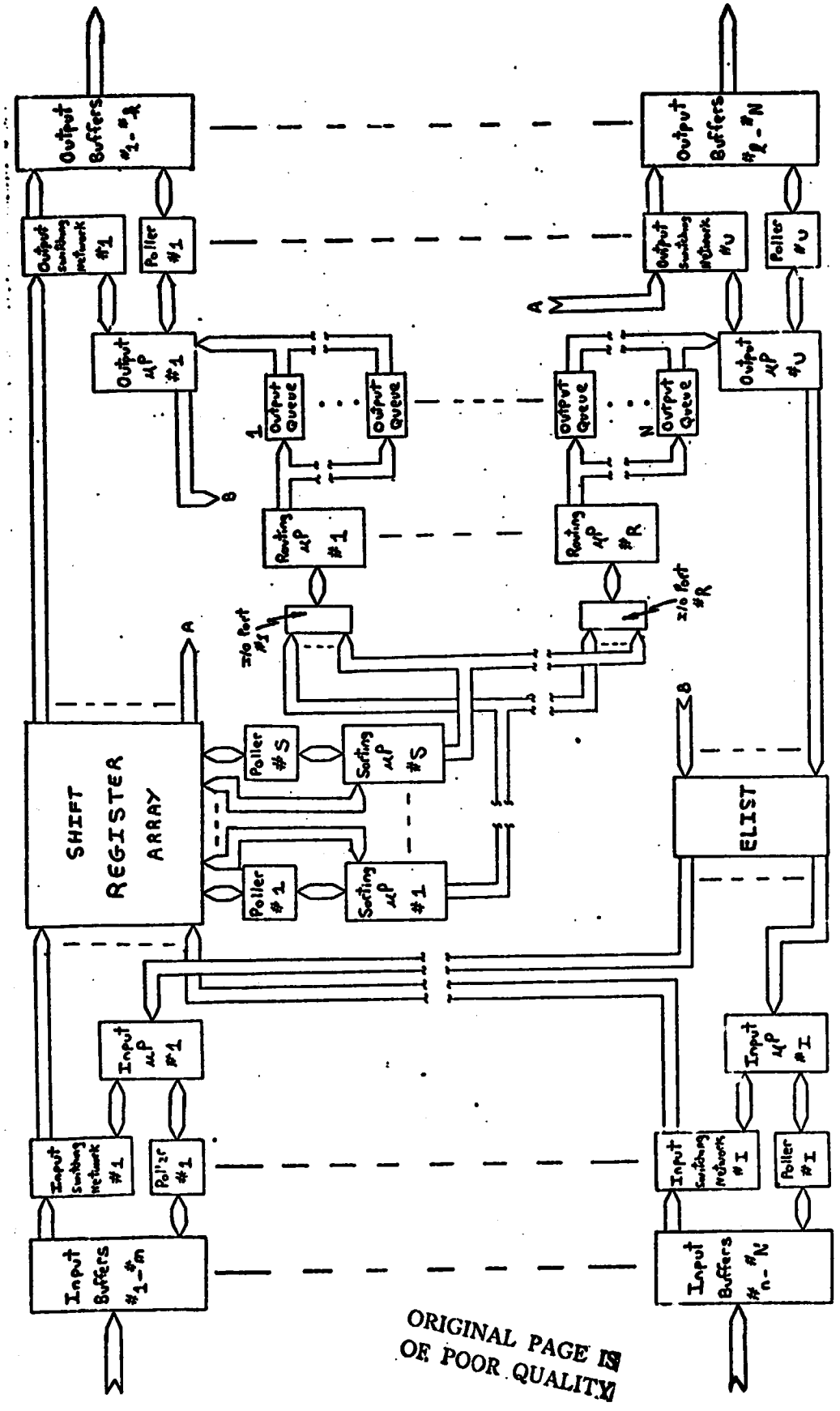


Figure 4.1 The Multiple Processor System Architecture

containing a newly received packet requests processor service. Dedicated hardware pollers sequentially scan their assigned group of input buffers searching for full buffers. One group of input buffers is assigned to one Input Processor. Upon finding a full buffer, a polling circuit signals the Input Processor it is serving. Immediately, this processor establishes a data link between the full buffer and the Shift Register Array. In order to set up this link, the processor must first find an available data path in the processor's dedicated Input Switching Network. Next, the processor must find an empty location in the Shift Register Array. Once the address of an empty location is fetched from the Empty Shift Register List (ELIST), the processor completes the data link. The processor then initiates the packet's serial transfer into the array. As in the previous systems, this transfer is hardware monitored and terminated, allowing the processor to move on to a new task.

The second function of the switch is to sort each packet in the array into groups of packets that are destined for the same group of ground stations. Each unique group of stations is serviced by one unique Routing Processor. Shift registers containing newly arrived packets signal for Packet Sorting Processor service. Dedicated hardware pollers scan their assigned group of shift registers for new packets. Once a polling circuit locates a new packet, the Sorting Processor it is serving is notified. This processor fetches the packet's header and corrects it. As in all previous systems, the header

is protected by the BCH error-correcting code. The packet's destination is then read from the header. Using this information, the Sorting Processor sends the packet's destination information and array address to an input/output port associated with the packet's destination. Each different I/O port belongs to one Unique Packet Routing Processor. Any Sorting Processor may access any I/O port.

The Packet Routing Processors carry out the switch's third function, which is the updating of the Output Queue Lists with the addresses of sorted packets. Once an I/O port is found to contain valid packet routing data, the I/O port polling circuit signals the Routing Processor it serves. The Routing Processor responds by fetching the packet's destination information. Using this information, the processor determines to which ground station the packet is destined. Packets leave for a ground station via an output buffer which corresponds to that ground station. Each output buffer is assigned to only a single ground station. In order to route a packet to a particular ground station, the Routing Processor must assign the packet to the software output queue list which corresponds to the proper output buffer. This assignment is made by fetching the packet's array address from the I/O port and placing it into the proper queue list. Each Routing Processor controls a unique group of output queue lists. A packet is considered routed once its array address is placed into one of the N queue lists.

The fourth and final function of the switch is to transmit the routed packets to their final destinations. This job belongs to the Output Processors. When an output buffer empties due to a completed packet transmission, the buffer requests processor service. Dedicated hardware pollers sequentially scan their own group of output buffers in search of empty buffers. When an empty buffer is found by a polling circuit, the Output Processor served by this poller is informed. The processor then accesses the output queue list belonging to the empty buffer. The address of the oldest packet waiting for transmission to this destination is fetched from the queue list. Next, the processor finds a free data path in its dedicated Output Switching Network. A link is established between the shift register containing the packet to be transferred and the empty buffer via the free data path. Once this link is complete, the packet transfer is initiated by the processor. Automatic hardware controls this serial packet transfer. As soon as an output buffer is loaded, the packet is automatically transmitted to the ground station by hardware external to the packet switch. While the internal hardware transfer takes place, the Output Processor updates ELIST by placing the packet's array address into ELIST.

If an output queue list is empty when its associated output buffer becomes empty the Output Processor must place the buffer in the "idle" state. An idle buffer will remain idle until a new packet arrives for that buffer. The Routing Processor will assign the new packet to the empty queue list.

Next, the Routing Processor must change the buffer's status to indicate that the buffer is empty and requires service from the Output Processor servicing that particular buffer.

4.2 Shared Resources

In the three processor design, contention problems between the different classes of processors are discussed in depth. A workable solution is found and implemented for each shared resource. In this multiple processor design, new contention problems arise. Since there can be more than one processor in each processor class, contention may occur between processors of the same class. The contention problems of these resources can be solved with design changes within the subsystem they serve. These design changes may affect the architecture of that subsystem, but they do not affect the other packet switch functions. Thus, the resource allocation schemes required by these shared resources are discussed in the sections which describe each subsystem of the switch.

However, there are several resources which are shared by two or more classes of processors. The design of these "Multi-Access Resources" and the formation of their allocation schemes may affect the architecture of two or more packet switch subsystems. Thus, these resources must be considered before the entire architecture of the packet switch can be designed. A review of the three processor design reveals that there are three resources which will become Multi-Access Resources in the multiple processor system. These resources are:

1. The Shift Register Array
2. The Output Queue Lists
3. ELIST

Now identified, each of these resources must be investigated and redesigned if necessary.

4.2.1 The Shift Register Array

Each Input (Output) Processor's switching network may be linked to any location in the Shift Register Array. However, no two Input (Output) Switching Networks will ever access the same location simultaneously. This is due to the fact that two or more Input (Output) Processors can never fetch the same address for a particular array location from ELIST (an Output Queue List) simultaneously as they service packets. As mentioned earlier, the array is capable of receiving a new packet while concurrently transmitting the older packet from the same location. Thus, no contention problems will arise between the Input and Output processors even if they access the same location concurrently. However, unless only one Sorting Processor is allowed to access a single location at one time, contention problems will arise. These problems can be eliminated by the assignment of groups of locations to one Sorting Processor. Since packets may be stored in the array with an uneven distribution, the locations assigned to each Sorting Processor should be interleaved. This ensures against the Sorting Processors being forced to carry unproportional workloads due to uneven packet storage.

4.2.2 The Output Queue Lists

In the three processor design, the Output Queue Lists are not completely free from contention problems. They are shared between the Routing Processor and the Output Processor. In the multiple processor system, the lists are needed by the multiple processors in both the Routing and the Output classes of processors. This requirement adds new contention problems for these already contention-plagued resources. In order to keep the amount of processor contention from increasing, a restriction regarding processor access to these lists must be made. Only one Routing Processor and only one Output Processor will be allowed to share a list. This requirement changes the workload of the Routing Processor used in the multiple processor packet switch.

In the three processor design, the Routing Processor services the entire Shift Rigester Array and all of the N output queue lists. A packet in any Shift Register Array location can require routing to any output buffer. A packet is considered routed only after its array address is placed into the proper queue list.

As described earlier, the Shift Register Array is now divided into groups of locations, each of which is serviced by a unique processor. This architecture, using the previous Routing Processor structure, would require that all the Routing Processors be allowed to access any of the N queue lists. Since this requirement is in conflict with the previous design decision that limited one Routing Processor to a list, a new architecture is needed.

The new architecture will force a division of the Routing Processor's workload. This workload division requires the implementation of a new class of processors which is needed to carry out some of the tasks formally assigned to the Routing Processor. The new class of processor is the Sorting Processor. Each Sorting Processor is assigned to a group of shift register array locations. They are allowed to send routing data to any Routing Processor. Each Routing Processor is assigned to a unique group of Output Queue Lists. These two classes of processors are linked by a contention-free hardware interface. Details concerning the actual implementation of this interface and the new processors are presented in section 4.4.

4.2.3 ELIST

The Empty Shift Register List (ELIST) is accessed by every Input Processor and every Output Processor as well. The previous ELIST structure cannot handle this requirement. Since only one Input Processor and only one Output Processor can access ELIST without interference, a new ELIST allocation scheme is needed to provide the multiple processors with contention-free access.

The first scheme considered is the division of ELIST into smaller lists. Each list would then be assigned to one Input Processor and to one Output Processor. However, in order for this scheme to work properly, the workload must be distributed evenly among the Input Processors and also among

the Output Processors. An example of how an uneven packet distribution can cause this scheme to fail is easily illustrated.

Assume each user is transmitting packets at his maximum allowable rate. Assume even further that most of the packets sent are destined to only one or two users that are serviced by the same Output Processor. After a short time, all but one of the Input Processors will have depleted their supply of array addresses. Only the Input Processor that shares the same ELIST with the busy Output Processor will continue to receive new array addresses. This case illustrates the need to supply ELIST data to each Input Processor through the use of a data distribution scheme. In addition, this case example clearly demonstrates that ELIST must remain as a single resource that is shared through the use of an allocation scheme. The idea of an ELIST data distribution system is the foundation on which two ELIST implementations are based. One design is based around an Elist Support Processor while the other design uses only automatic hardware. These two designs are discussed in detail below.

4.2.3.1 Processor-Controlled ELIST

Since there are no constraints regulating the use of support processors, the use of a processor to coordinate the operation of the ELIST data distribution system is a logical choice. The processor controlled ELIST system architecture is presented in Figure 4.2.

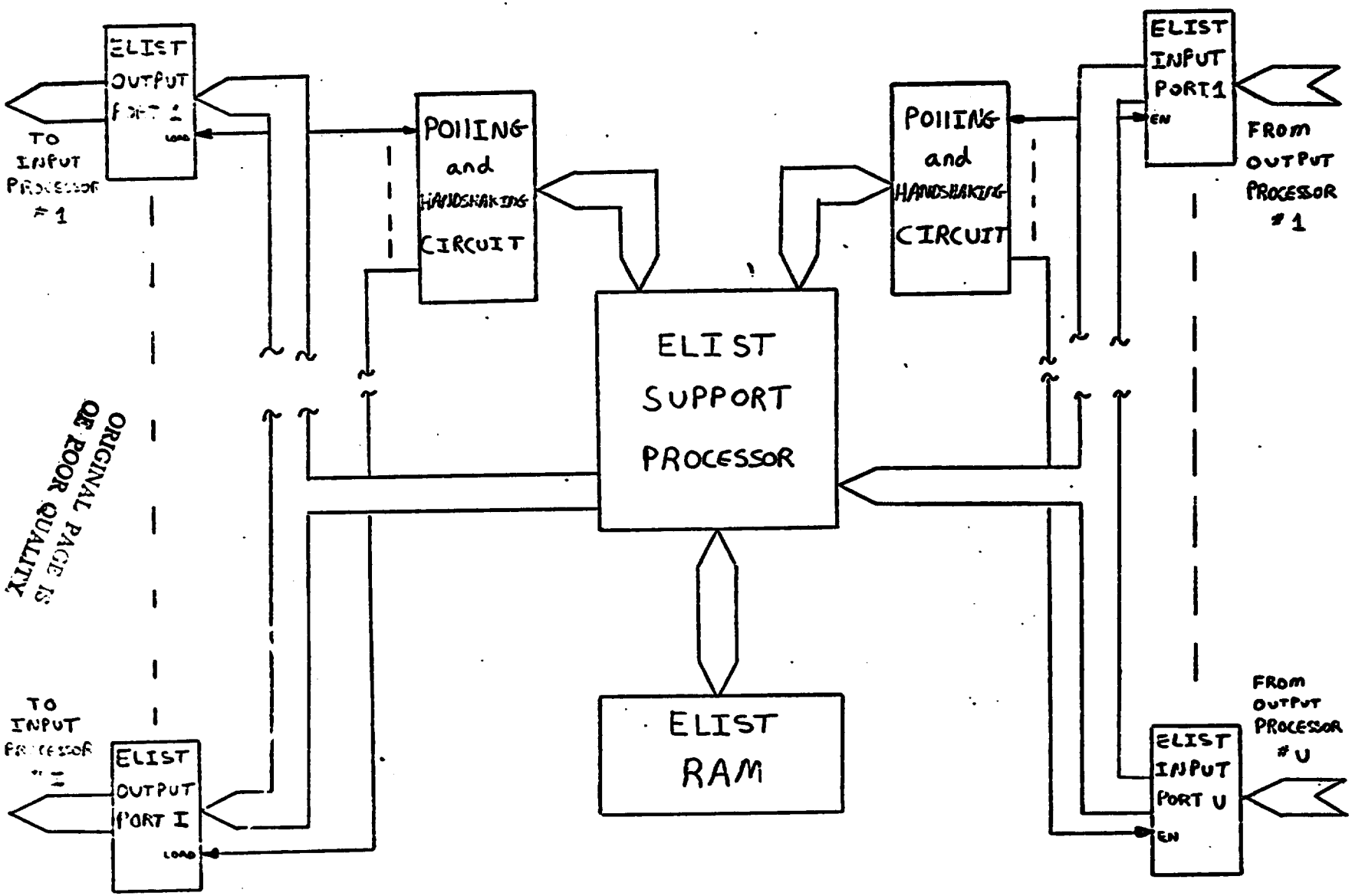


Fig. 4.2 Processor-Controlled ELIST Architecture

The operation of this ELIST data distribution system is straightforward. Each Output Processor sends its ELIST data to a dedicated I/O port. These ports support the common DAV/DAC handshaking protocol. A dedicated poller scans these ports in search of a full port. When the poller finds a full port (identified as full by its activated DAV flag), it signals the Elist Support Processor. The support processor then fetches the data and sets the DAC flag. The Elist Support Processor then checks to see if any Input Processor-linked I/O port requires data. Each of these I/O ports is assigned to one Input Processor. Again, the DAV/DAC flag handshaking is used and a dedicated hardware poller is also used to scan these ports. If the poller had located an empty port (signalled by an activated DAC flag), the Elist Support Processor sends the ELIST data directly to the empty port. If no I/O port is empty, the data is stored into the ELIST RAM. If an Input Processor's I/O port empties before the Elist Support Processor has received data from an Output Processor, the support fetches the data from the RAM and then sends it to the empty port.

Since this ELIST data distribution system is controlled by a processor, it can serve the Input Processors and the Output Processors only as fast as the Elist Support Processor executes its task. The Elist Support Processor can support any packet switch throughput up to 3 Mega-packets per second (see Appendix). This ELIST structure is a throughput-limiting function. Therefore, adding additional processors to the

other four classes will never increase the system throughput beyond the upper bound of 3 Mega-packets per second. Thus, this system is replaced by a hardware-controlled data distribution system, which is the next topic of discussion.

Although the processor-controlled system is not used in this particular architecture, the processor architecture, the interface hardware and the software required for implementation are located in the Appendix. This material is presented because the processor-controlled ELIST scheme is less complex than the hardware-controlled ELIST and it can offer the user some degree of flexibility in that the processor software can be custom tailored. Thus, the processor-based ELIST is the recommended implementation for packet switches operating below 3 Mega-packets per second.

4.2.3.2 Hardware-Controlled ELIST

Since hardware is relatively faster than software, a completely hardware-controlled ELIST will serve the packet switch at the fastest rate possible. This design removes the previous throughput limitations encountered in the ELIST Support processor-based design.

ELIST interfaces to the Input Processors and the Output Processors through input/output ports. A dedicated port is assigned to each processor accessing the list. In order to explain how the system services the two classes of processors (Input and Output), the operation of the data storage function is described first.

Figure 4.3 contains the ELIST Data Input Port architecture. When an Output Processor has a new array address for ELIST, it checks the port's Data Accepted (DAC) flag. If the previous data was fetched by the ELIST hardware, this flag is set. A set DAC flag allows the Output Processor to load its port with the new data. Once the data is loaded into the port, the processor sets the Data Available (DAV) flag. If the DAC flag is not set, the Output Processor must wait until this flag gets set.

The ELIST storage hardware is controlled by a polling circuit which is driven by a counter. All the ELIST Data Input Port DAV flags are sent to the ELIST DAV MUX. The Output of this MUX generates the FULL PORT signal which reflects the status of the DAV flag selected. Selection of the DAV flags is controlled by the value of the counter. The value of this counter is also supplied to the ELIST Input Enable Demux. The activated Demux output enables the handshaking logic and the tri-stated port output of the addressed I/O port.

If the addressed I/O port's DAV flag is set, the FULL PORT signal becomes activated. This activated signal sets the STORE DATA Flip-Flop. Once set, this flip-flop halts the poller's counter. Simultaneously, the flip-flop activates the one-shot that generates the active-low WRITE signal. While the WRITE signal is activated, the two-port ELIST RAM is enabled in the write mode. The data from the enabled I/O port is then strobed into the RAM.

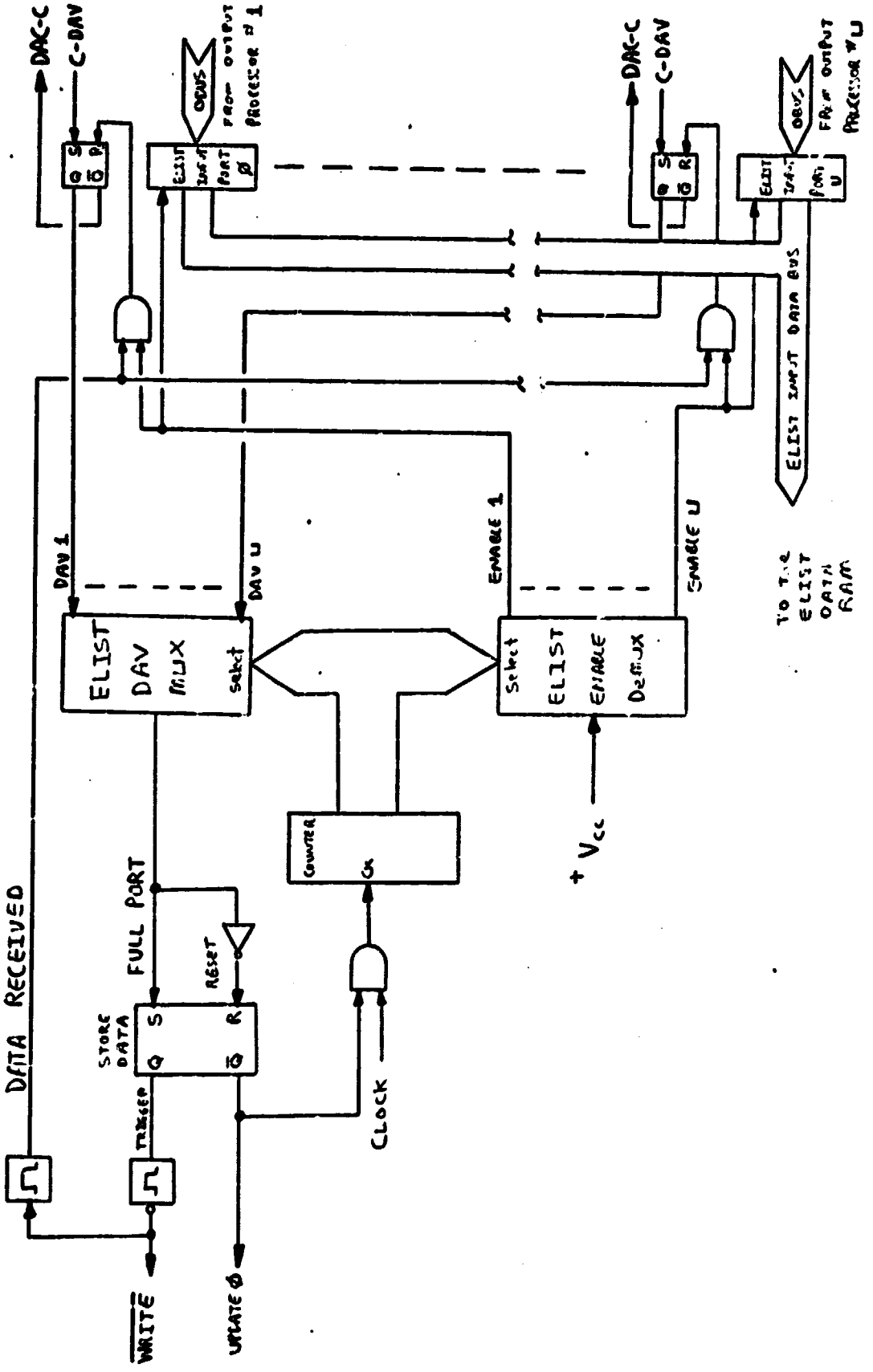


Fig. 4.3 ELIST Data Input Port

The ELIST RAM Structure is shown in Figure 4.4. The ELIST Data Structure is presented in Figure 4.5. In this data structure, both the read and write operations are performed before the index pointers are updated. Both pointers are updated by being incremented. Once the bottom of the list is encountered, they roll over and return to the top of the list.

Once the write operation is complete, the low-active WRITE signal goes high. The leading edge of this low-to-high transition fires the one shot which activates the DATA RECEIVED signal. This activated signal clears the DAV flag and sets the DAC flag belonging to the enabled I/O port. The clearing of the DAV flag clears the STORE DATA flip-flop. The reset flip-flop activates the UPDATEØ signal which increments the counter which serves as the write index pointer (EPTRØ). In addition, the reset flip-flop enables the poller to restart. Figure 4.6 contains the timing diagram and the significant events for this entire operation.

The second function of the ELIST data distribution system is to supply each Input Processor with the address data stored in the ELIST when required. Figure 4.7 contains the ELIST Data Output Port system which carries out this task. The primary function of this system is to keep each ELIST Data Output Port filled with valid data. If the ports are kept full, no Input Processor will be forced to wait for data.

As with the ELIST Data Input Ports, each Data Output Port is assigned to one processor. Each port has its own handshaking flags. When an Input Processor needs data from the

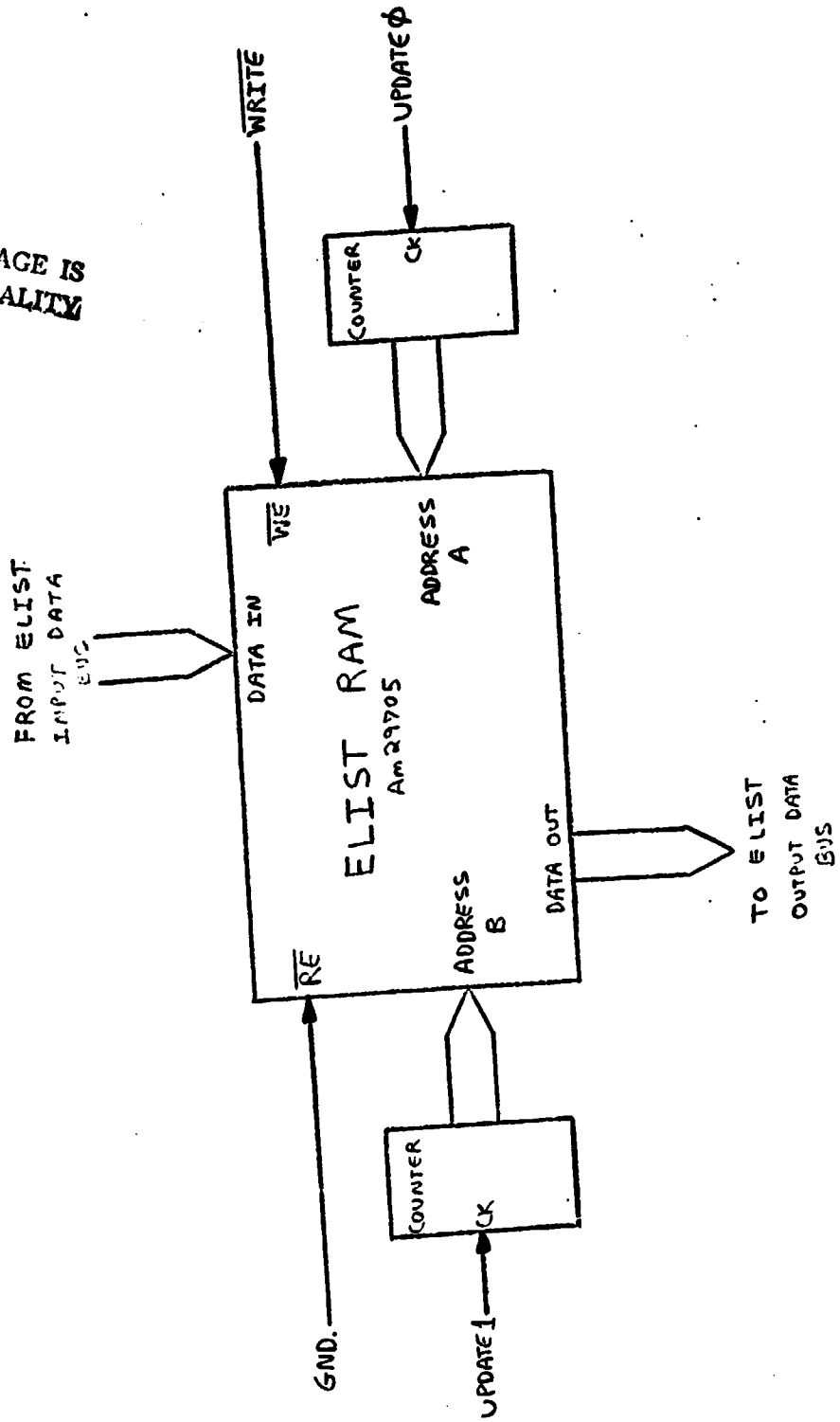


Fig. 4.4 ELIST RAM Structure

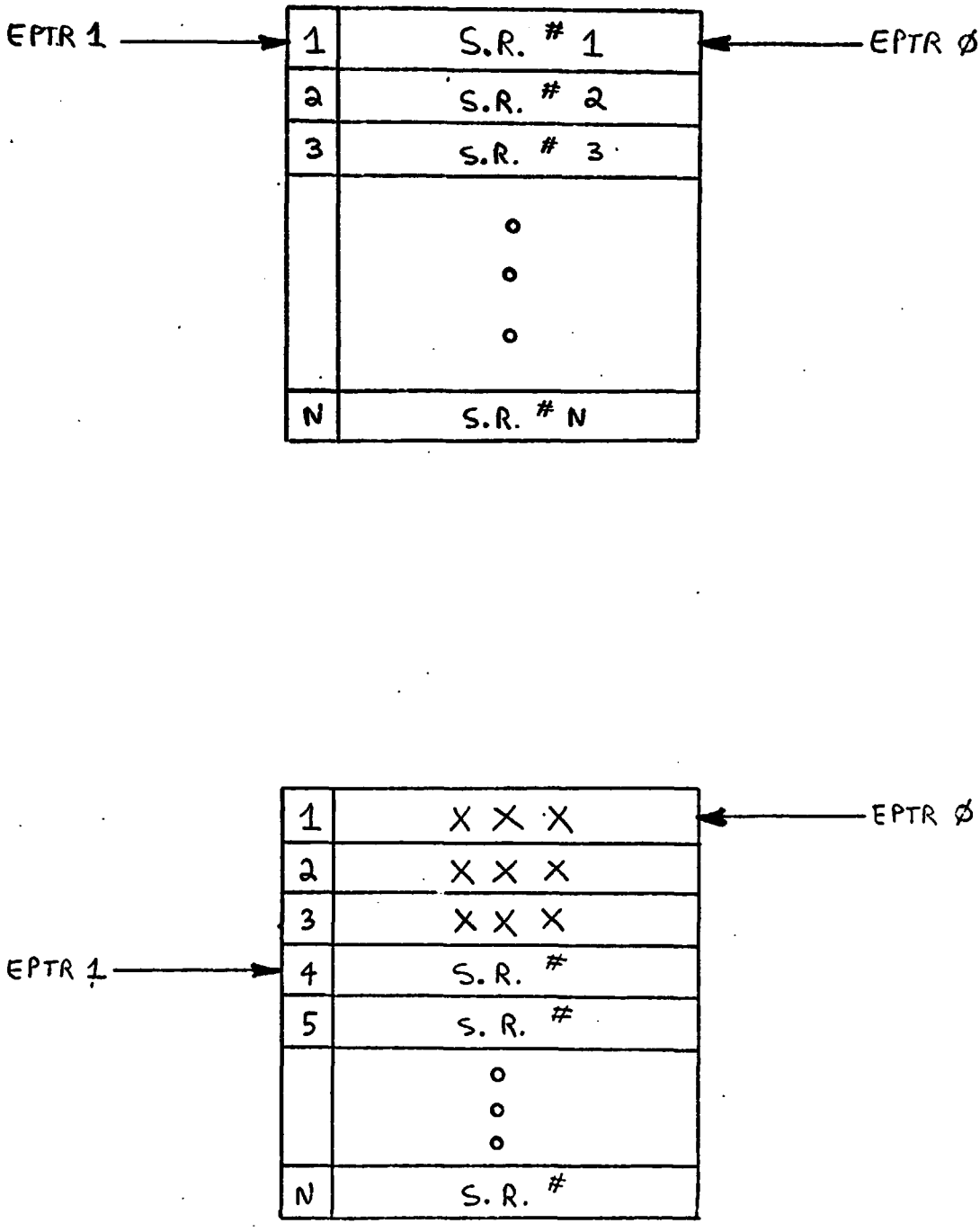
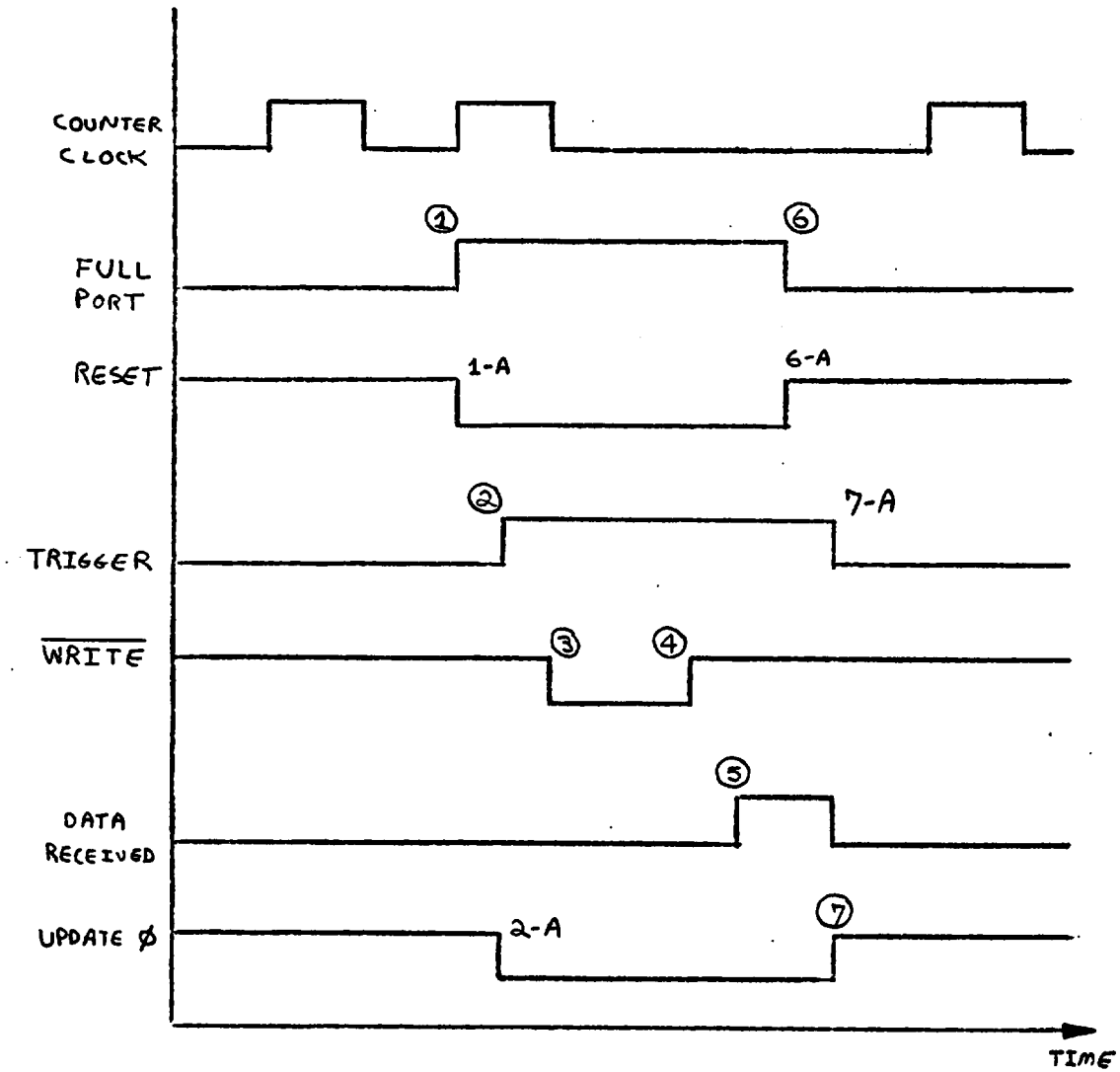


Fig. 4.5 ELIST Data Structure



① A full input port has been found by the poller. The activated FULL PORT signal sets the STORE DATA flip-flop.

② The set flip-flop activates the TRIGGER signal. The TRIGGER signal fires the one shot.

③ The one shot generates the $\overline{\text{WRITE}}$ signal enabling the data RAM.

④ The data has been stored and the $\overline{\text{WRITE}}$ signal is deactivated. The DATA RECEIVED one shot is triggered.

⑤ The one shot fires a pulse to the selected handshaking flip-flop.

⑥ The DAV signal is cleared. This clears the STORE DATA flip-flop.

⑦ The RAM's index pointer is incremented by the activated UPDATE Ø line.

Fig. 4.6 ELIST Input Port Hardware Timing Diagram

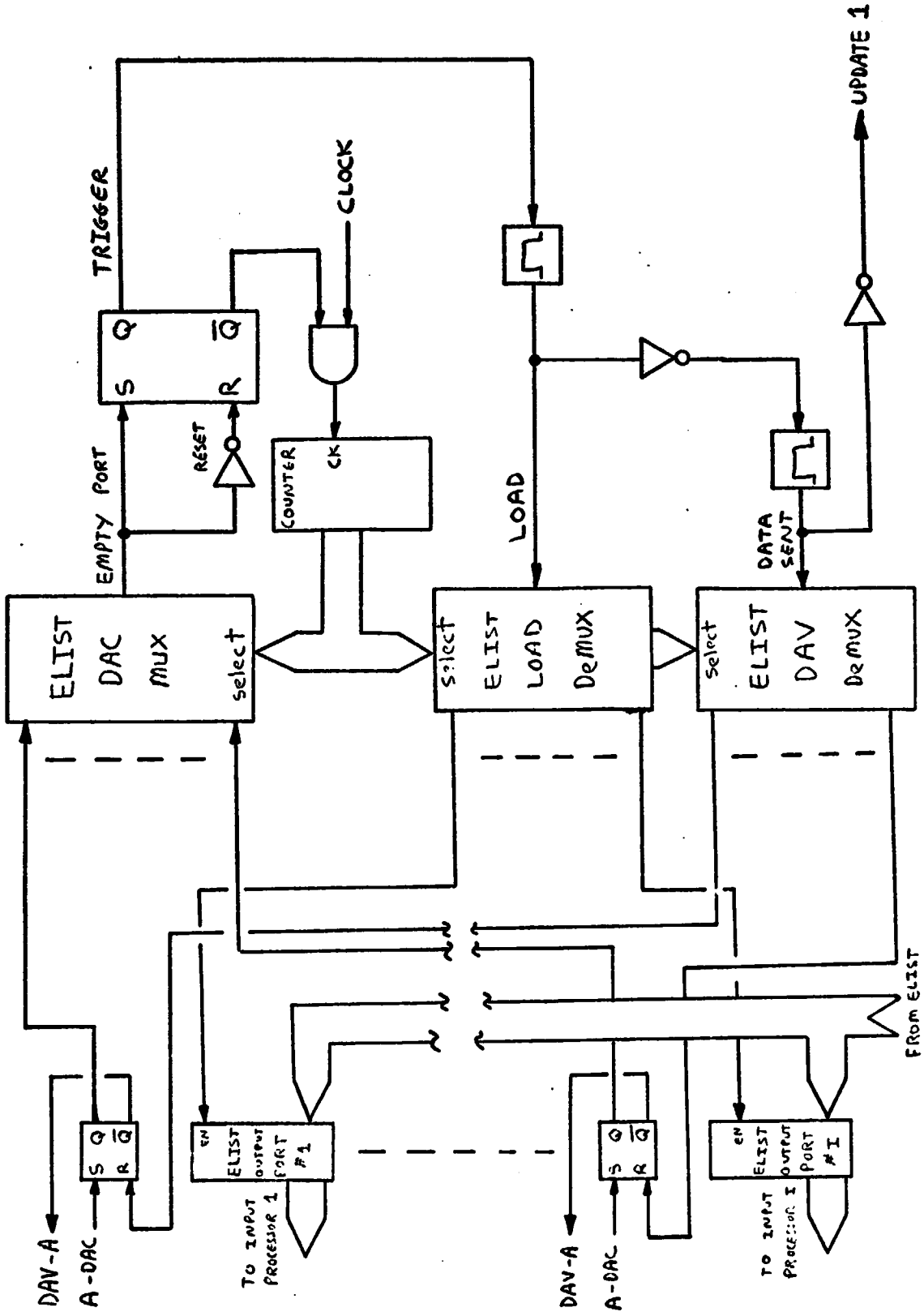


Fig. 4.7 ELIST Data Output Port

ELIST, it checks the I/O port's DAV flag. If this flag is set, valid data is held in the port and the Input Processor fetches this data immediately. If the DAV flag is not set, the processor must wait for service.

Once data is fetched from an I/O port, the Input Processor accessing this port sets the DAC flag. Every I/O port's DAC flag is sent to the ELIST DAC MUX. The output of this MUX generates the EMPTY PORT signal. Selection of the DAC flag is controlled by the counter which drives the mux. The value of this counter is also sent to the LOAD Demux and to the DATA SENT DEMUX. The activated Demux outputs enable the handshaking logic and the data loading circuitry of the addressed I/O port.

If the selected I/O port's DAC flag is set, the EMPTY PORT signal is activated. This signal then sets the SEND DATA flip-flop. Once set, this flip-flop halts the poller and activates the one-shot that produces the active-low LOAD signal. While the LOAD signal is active, the data on the ELIST Output Data Bus is strobed into the enabled I/O port. The data on the ELIST Output Data Bus is supplied from the RAM location selected by the read index pointer.

Once the data transfer has been finished, the active-low LOAD signal goes high. The leading edge of this low-to-high transition activates the one shot that generates the DATA SENT signal. The DATA SENT signal then clears the enabled port's DAC flag and sets its DAV flag. The reset DAC flag clears the SEND DATA flip-flop, enabling the poller to restart its

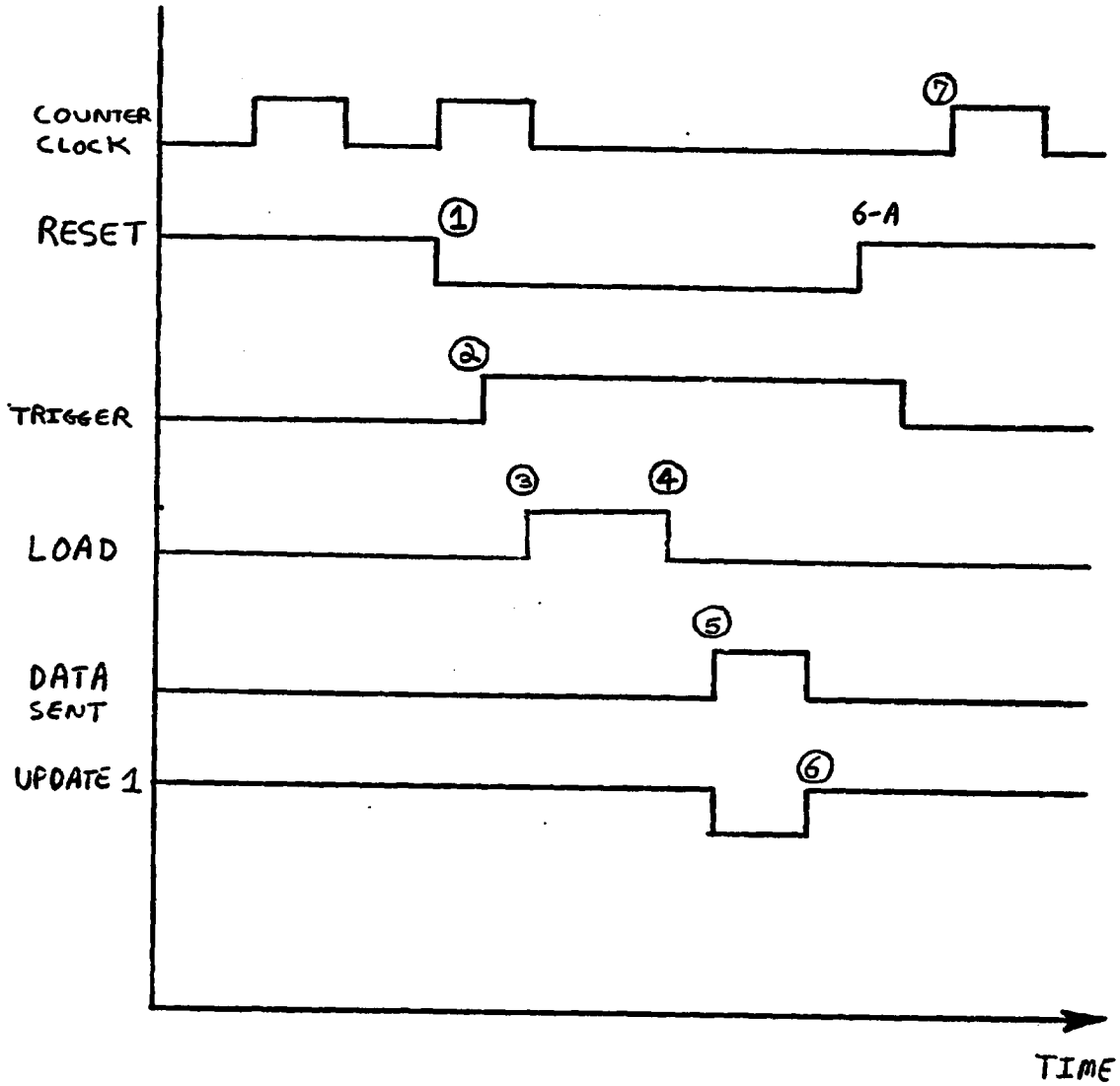
scanning. Simultaneously, the DATA SENT signal goes low resulting in the updating of the read index pointer. This update increments the hardware counter which serves as the read index counter. A timing diagram of the complete ELIST output function, along with the significant timing events, is presented in Figure 4.8.

4.3 The Input System

The Input System consists of the Input Buffers, the Input Processors, the Input Switching Networks and the Input Polling Circuits. This sytem interfaces to the Shift Register Array and the ELIST Data Distribution System. The architectural organization of this system is presented below.

However, before the architecture can be designed and explained, the contention problem related to the Input Switching Network must be solved. As in the previous designs, the Input Switching Network provides programmable data paths from the input buffers to the Shift Register Array. In order to provide the address of an available data path in the network, the status of each path is monitored by the hardware in the Data Path Busy Port. This port is accessed by the Input Processor.

If a single Input Switching Network is used in the multiple processor system, access to the status port must be granted to only one Input Processor at a time. Since several Input Processors will require access to this resource, a resource allocation scheduling scheme is needed. This scheme



① An empty output port has been found by the Poller. The poller halts and the LOAD DATA flip-flop is set.

② The set flip-flop activates the TRIGGER signal. This signal fires the LOAD one shot.

③ The one shot generates the LOAD signal enabling the selected port.

④ The data has been placed into the port. The LOAD signal is deactivated.

⑤ The DATA SENT signal is activated, setting the port's DAV flag.

⑥ The ELIST RAM index pointer is updated.

⑦ The poller is restarted.

Fig. 4.8 ELIST Output Port Hardware Timing Diagram

will require new hardware and additional software. The additional software will reduce the throughput of the Input Processors. Throughput may be reduced even further if the Input Processors are forced to wait for the resource whenever it is busy. This contention problem needed to be solved. The solution implemented in this design eliminates contention completely by allocating a dedicated Input Switching Network to each input processor.

4.3.1 Architectural Workload Division

In the three processor design, the workload is divided into three relatively independent tasks. This scheme works quite well, in that each processor can carry out its assigned task without interference from the other processors. However, in the multiple processor design, the workload of the packet switch must be sub-divided within the three functions. Processors in the same class must share the workload within the function assigned to that processor class. Therefore, if the proper architecture is not implemented, a processor may be faced with interference from the other processors in its own class.

The processors controlling the Input System can be organized using one of two techniques: Master/Slave Scheduling or Separate Systems [5]. The Master/Slave Scheduling scheme is organized such that one processor maintains the status of all the "Slave Processors" and the uncompleted tasks. This "Master Processor" schedules the work for each of the Slave Processors.

The Separate Systems scheme is organized such that each processor carries out its assigned tasks in parallel with the other processors. The assignment of tasks for each processor is fixed by the system architecture. There is no dynamic allocation of processors to tasks, as in the Master/Slave Scheduling System. In addition, each processor is assigned dedicated memory and dedicated I/O devices.

These two schemes are the foundation on which two architectures for the input system are based. Each of the two architectures are presented below. Also included are the design considerations which led to the selection of the Separate System scheme.

4.3.1.1 Master/Slave Scheduling

One possible implementation of the Input System using the Master/Slave Scheduling scheme is presented in Figure 4.9. This figure contains a block diagram of the Input System Architecture A.

Input System Architecture A uses a hardware poller to locate full input buffers. Once the poller finds a full buffer, which is indicated via an Input Status Word (ISW), it stops and signals the Job Scheduling Processor (Master Processor). The Job Scheduling Processor inputs the address of the full buffer from the poller. The Job Scheduling Processor then updates the ISW to indicate "partial service" and restarts the poller. Next, the Job Scheduling Processor fetches the address generated by the priority encoder. This encoder

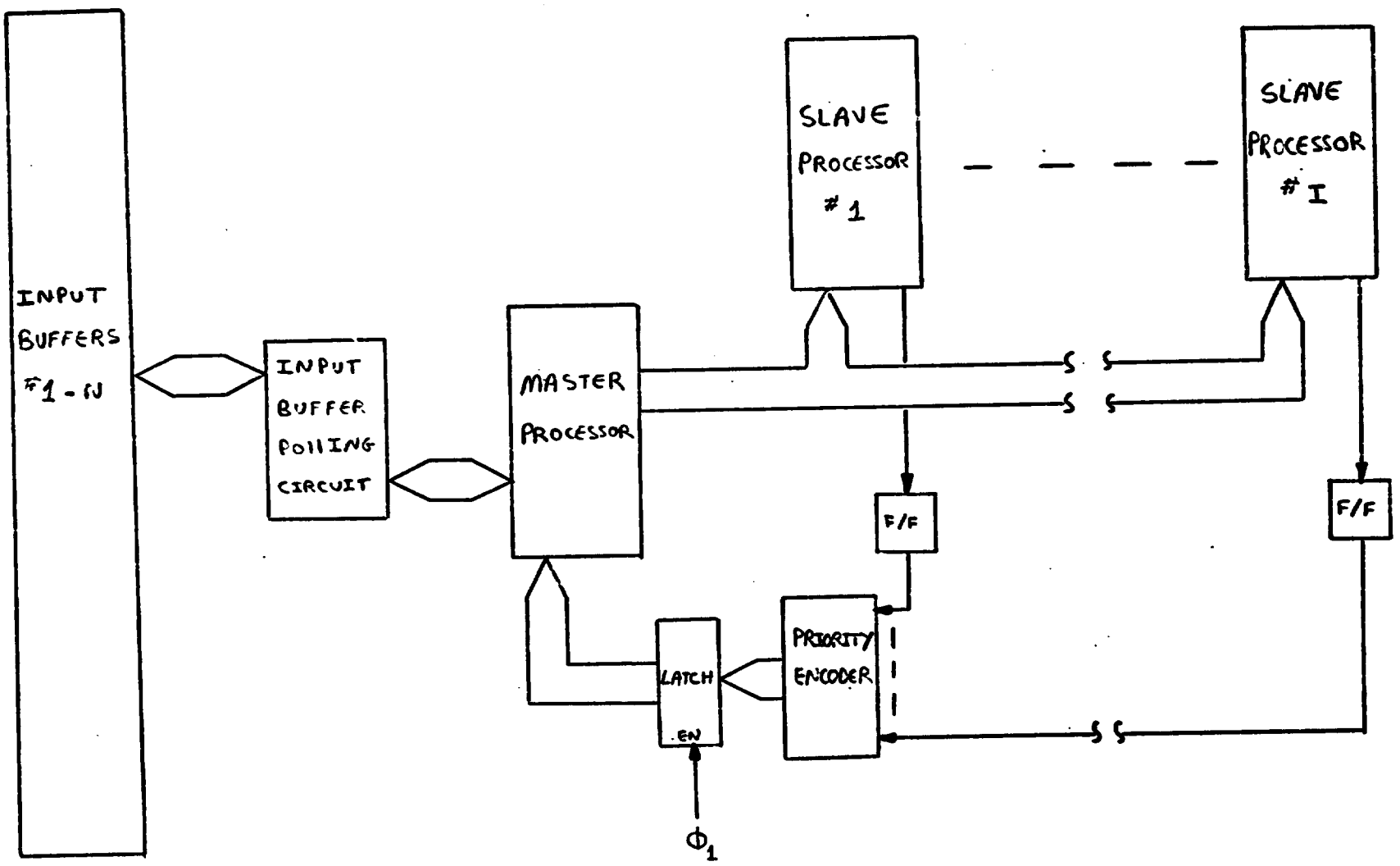


Fig. 4.9 Input System Architecture "A"

is driven by the Busy flip-flops that indicate the current status of each Slave Processor. The encoder supplies the address of the free Slave Processor which has the highest assigned priority. Using this address, the Job Scheduling Processor assigns the task of servicing the full buffer to the free Slave Processor. This Slave Processor sets its Busy flip-flop and begins the task of inputting the packet to the Shift Register Array.

The main advantage of this scheme is that the workload is shared by all the available Slave Processors, regardless of the distribution of the incoming packets. Since the Slave Processors are assigned to tasks (incoming packets) and not to the input buffers, all the processors will be utilized even if only one or two channels are heavily loaded. An additional advantage of this system is that under lightly loaded conditions, the low priority Slave Processors will be free. These low priority processors could be programmed to execute background functions. Service for the input buffers could then be interrupt driven.

There exist two disadvantages in Architecture A. The first disadvantage is a reliability problem. If the Job Scheduling Processor fails, the entire packet switch becomes inoperative. One possible solution to this problem is the implementation of additional Job Scheduling Processors that are assigned their own dedicated Slave Processors. Another possible solution is to have a Slave Processor replace the Job Scheduling Processor in the event of a failure. Both of

these schemes will add complexity to the hardware and/or software.

The second disadvantage is the amount of hardware required to allow any Slave Processor to serve any input buffer. This architecture could require thousands of control lines for just one control signal. An example of this problem is given below for a typical system:

N = 100 Users (100 input buffers required)

# of Slave Processors = 4 processors

# of Input Switching Network Data Paths

= 10 paths/processor

# of DATA IN lines = 1 line/user/data path

(100 users)·(4 processors)·(10 paths/processor)·1 line/user/data path = 4000 DATA IN lines.

As a result of this finding, a new multiprocessor architecture for the Input System is proposed. The new architecture is discussed below.

4.3.1.2 Separate Systems

The Input System Architecture is presented in Figure 4.10. Each Input Processor controls a complete input system. Each of these systems operate independently of one another. The size of these systems is determined by the number of input buffers assigned to each system. Once a group of buffers are assigned to an Input Processor, they remain fixed to that processor. Therefore, in this scheme,

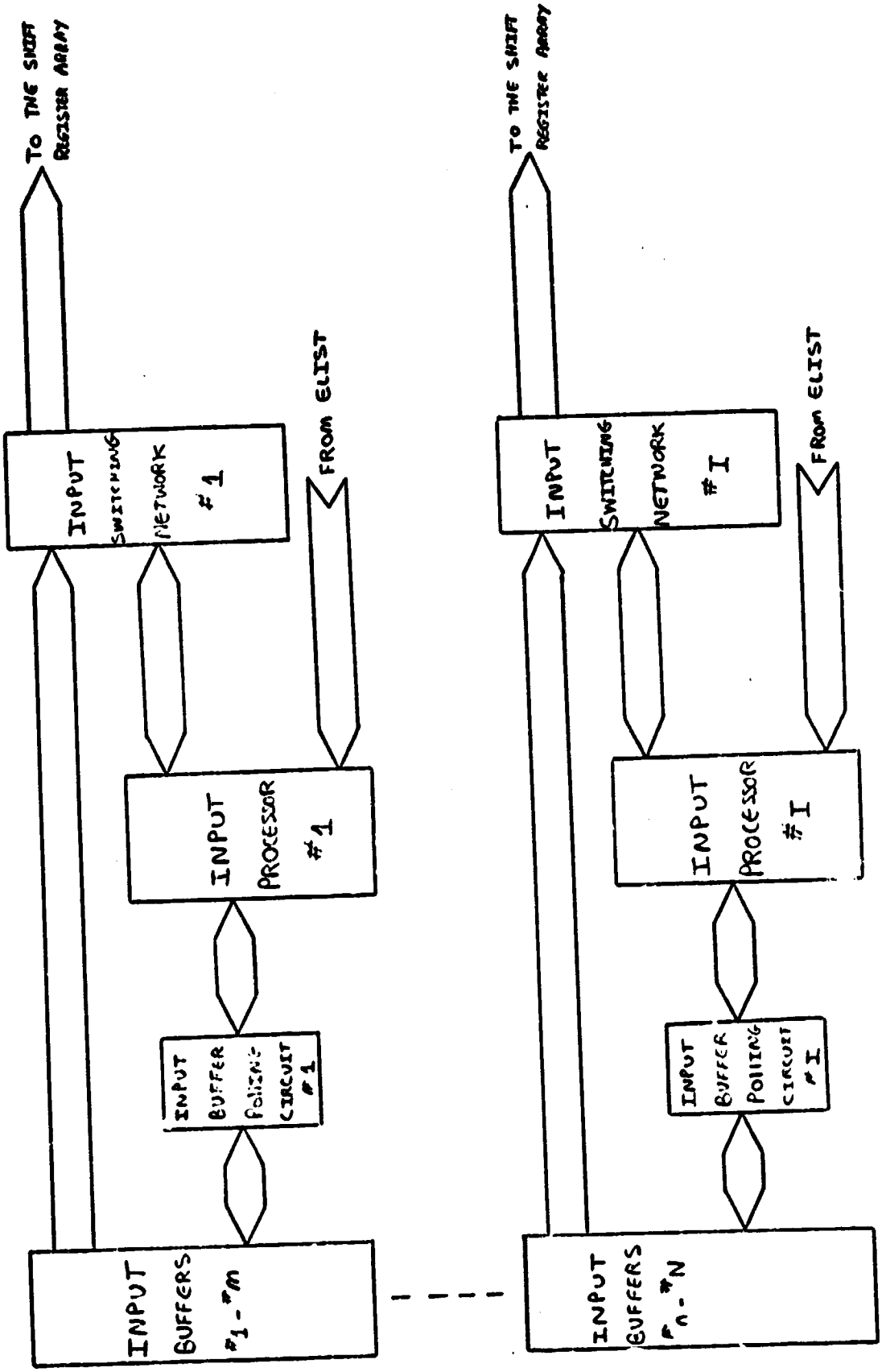


Fig. 4.10 Input System Architecture

tasks are assigned to processors, while the reverse is never true.

In order to compare the hardware complexity of this scheme to the complexity of the Master/Slave Scheduling scheme, the previous example using the DATA IN lines will be continued:

N = 100 users (100 input buffers required)

# of Input Processors = 4 processors

# of Input Switching Network Data Paths

= 10 paths/processor

# of DATA IN lines = 1 line/1 user/1 data path

(25 users/separate system/processor)·(4 separate systems)·(10 data paths/processor)·(1 DATA IN line/1 user/1 data path) = 1000 DATA IN lines

The Master/Slave Scheduling scheme required 4000 DATA IN lines. Since the DATA IN line is only one of two Input Switching Network signals that requires 1 line per 1 user per 1 data path, the Separate Systems scheme is clearly less complex.

The one major drawback with this architecture is that idle or lightly loaded processors cannot be assigned to heavily loaded channels if those channels are under the control of another processor. Thus, some Input Processors may become heavily loaded while the other Input Processors remain idle or under-utilized. However, this architecture is considered to be the best compromise since it is not as complex

as Architecture A. Therefore, this is the architecture that is chosen for the actual implementation.

Since this architecture merely divides the workload by means of buffer assignments, no major hardware changes are required. The Input Buffers, the Input Switching Networks and the polling circuits are identical to those used in the three processor design. Therefore, these system components are not presented in this chapter (See 3.1 for a review of these components).

4.3.2 The Input Processors

The Instruction Execution Units and the Microprogram Control Units for the Input Processors are the same as those used in the Three Processor Designs (see section 3.2 for a review). However, since ELIST has been redesigned to meet new requirements, the Input Processors' Microprogram word is different. Figure 4.11 contains the IEU Control Fields in the microprogram word. Figure 4.12 contains the MCU Control Fields and the Jump Control logic function for the Input Processor. Again, their functions are similar to those in the three processor design as discussed in 3.2.4.

4.3.3 The Input Software Routine

The Input Service Routine is sense loop driven. The Input Processor remains in the loop until the Input Polling Circuit locates a full input buffer. Once a full buffer is found, the processor leaves the loop and fetches the address of the buffer from the poller. Next, the Input Processor

Ø-15	16-19		2Ø	21-24	25	26-30	31	32	33	34	35	36
IMMEDIATE OPERAND	ALU SOURCE		B	ALU	CARRY	ALU	BUS	OUTPUT	DECODER	WRITE	POWER	ELIST
	R SOURCE	S SOURCE	ADDRESS	FUNCTION	IN	DESTINATION	LATCH	ENABLE	ENABLE	ENABLE	RESET	DAC

MNEMONIC FIELDS

D_{15} — — — $D_Ø$	$\overline{EA}$, I_9, $\overline{OE_B}$, $I_Ø$	BAS	$I_4 \cdots I_1$	C_n	$\overline{IEN}$, I_8 I_5	ALE	$\overline{OE_Y}$	DE	$\overline{WE}$	A-RESET	A-DAC

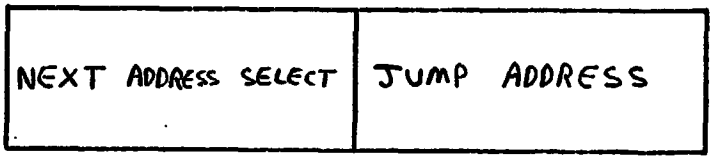
CONTROL BITS

				ALU SOURCE	
$\overline{EA}$	I_9	$\overline{OE_B}$	$I_Ø$	R	S
1	Ø	Ø	Ø	µW	Reg @ B
1	Ø	X	1	µW	Q
1	Ø	1	Ø	µW	Poller Port
1	1	Ø	Ø	IBUS	Reg @ B
1	1	X	1	IBUS	Q
1	1	1	Ø	IBUS	Poller Port

BAS	B SOURCE
Ø	SCRATCH 1
1	SCRATCH 2

ALE	BUS LATCH
Ø	NONE
1	ADDRESS LATCH

Fig. 4.11 Input Processor IEU Microprogram Control Fields



37-38

N_1	$N_\emptyset$	NEXT ADDRESS
Ø	Ø	μPC + 1
Ø	1	UNCONDITIONAL JUMP
1	Ø	JUMP ON IBSR-A = Ø
1	1	JUMP ON DAV-A = Ø

39-42

JUMP ADDRESS
JA_3, JA_2, JA_1, $JA_\emptyset$

IBSR-A	ELIST DAV	N_1	$N_\emptyset$	FE	C_n	S_1	$S_\emptyset$	ADDRESS SOURCE
X	X	Ø	Ø	1	1	Ø	Ø	μPC + 1
X	X	Ø	1	1	Ø	1	1	Jump Address
Ø	X	1	Ø	1	Ø	1	1	Jump Address
1	X	1	Ø	1	1	Ø	Ø	μPC + 1
X	Ø	1	1	1	Ø	1	1	Jump Address
X	1	1	1	1	1	Ø	Ø	μPC + 1

Fig. 4.12 Input Processor MCU Control Fields and Jump Control Logic Function

fetches the address of a free data path in its dedicated Input Switching Network. Simultaneously, the processor clears the buffer's service request flag and restarts the poller. The Input Processor then checks the DAV flag at its ELIST Data Port. If this flag is not set, the processor loops until the flag becomes set. When the flag is set, the Input Processor fetches the address of an empty shift register from the port and sets the DAC flag.

Using these three addresses, the Input Processor links the full input buffer to the empty shift register via the free data path. Once this link is established, the processor initiates the data transfer and returns to the loop. A flow chart of this software routine is presented in Figure 4.13. A listing of this program is given in Figure 4.14.

4.4 The Routing System

The Routing System consists of the Shift Register Array, the Sorting Processors, the Shift Register Array polling circuit, the Routing Processors, and the Packet Routing Data I/O ports. The Routing System interfaces to the Input System, the Output Queues Lists and the Output System. The architectural organization of this system is presented below.

4.4.1 Architectural Workload Division

As discussed in section 4.2, the Routing function as defined in the three processor architecture can no longer meet the requirements of the multiple processor design. The

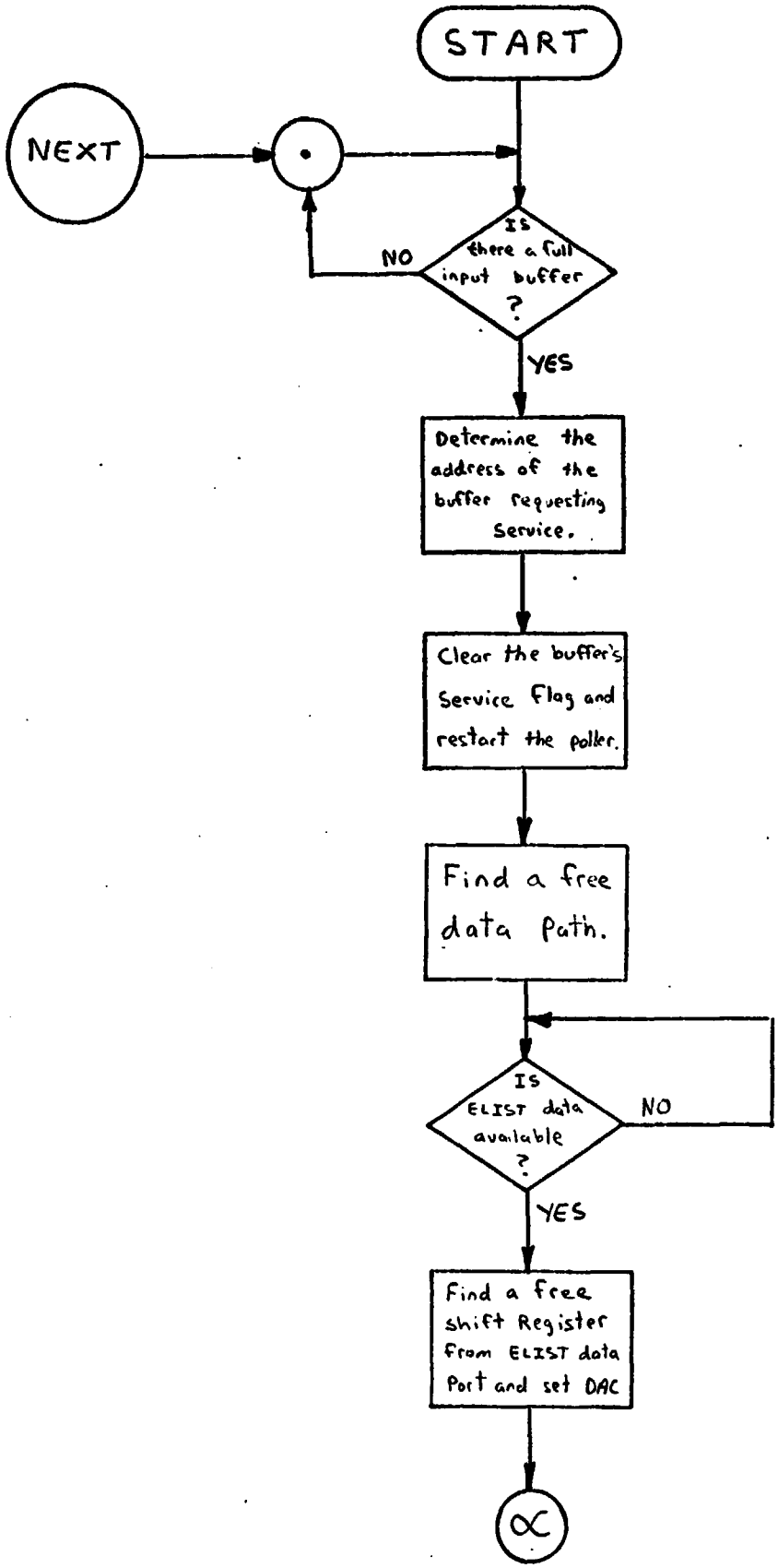


Fig. 4.13 Input Service Routine Flowchart

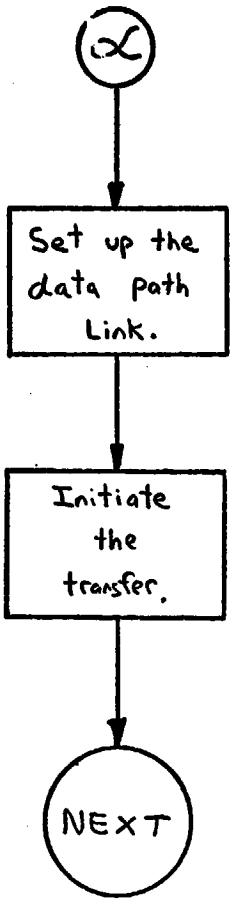


Fig. 4.13 Input Service Routine Flowchart, continued.

```
Input:  If IBSR-A=Ø, JMP to INPUT

                                        *Is there an input buffer
                                         requesting service?
                                         NO:  Loop @ INPUT.

        Input Polling Port → Q

                                        *YES:  Input the buffer's
                                               address.

        Input Data Path Status Port Address→Address Latch (μp1-A);
        Reset Poller Data Path Busy Status Port→Scratch 1

                                        *Find a Free data path,
                                         clear IBSR-A and restart
                                         the poller.

        ELIST Data Port Address→Address Latch (μp2-A)

WAIT:   If ELIST DAV = Ø, JMP to WAIT

        ELIST Data Port→Scratch 2; Sent a DAC

                                        *When the data becomes
                                         available, input the shift
                                         register number from the
                                         ELIST port.

        Scratch 1+Data Path Latch A Base Address→Address Latch (μP3-A)
        Q → Data Path MUX select Latch A(D)

                                        *Link the buffer to the
                                         data path.

        Scratch 1+Data Path Latch B Base Address→Address Latch (μP4-A)
        Scratch 2 → Data Path DeMUX select Latch B(D)

                                        *Link to empty shift
                                         register to the data path.

        Data Transmit Control Address→Address Latch
        Scratch 1→Data Bus Decoder (M1-A); JMP to INPUT

                                        *Start data transfer and
                                         return to sense loop.
```

Figure 4.14 Input Service Routine

principle requirement is that each Output Queue List be accessed by only one Routing Processor. This constraint is satisfied by dividing the Routing function into two smaller tasks. Each task, the sorting of packets and the routing of packets, is assigned to one class of processors. The Packet Sorting Processor is assigned the task of sorting each packet in the array. The sorting function requires that a packet's destination and its location in the array be sent to the proper Packet Routing Processor. The Packet Routing Processor then uses this information to route the packet by placing the packet's array address into the proper output queue list. The system architecture for a single Packet Sorting Processor is presented in Figure 4.15. Figure 4.16 illustrates the system architecture for a single Routing Processor.

Implementation of this scheme does not require the redesign of the Shift Register Array, the Shift Register Array Polling Polling Circuit, or the Output Queue Lists. Therefore, these components are not discussed in this chapter (see section 3.1 for a review of this hardware). However, the processors and their software routines are different from those in the previous design. In addition, the new component, the Packet Routing Data Port, is implemented in this architecture. Therefore, these topics are discussed. The Packet Routing Data Ports are presented below as the first topic.

4.4.2 Packet Routing Data Ports

The Packet Routing Data Ports provide the necessary interface between the Packet Sorting Processor and the Packet Routing Processor. Each Packet Routing Processor is assigned

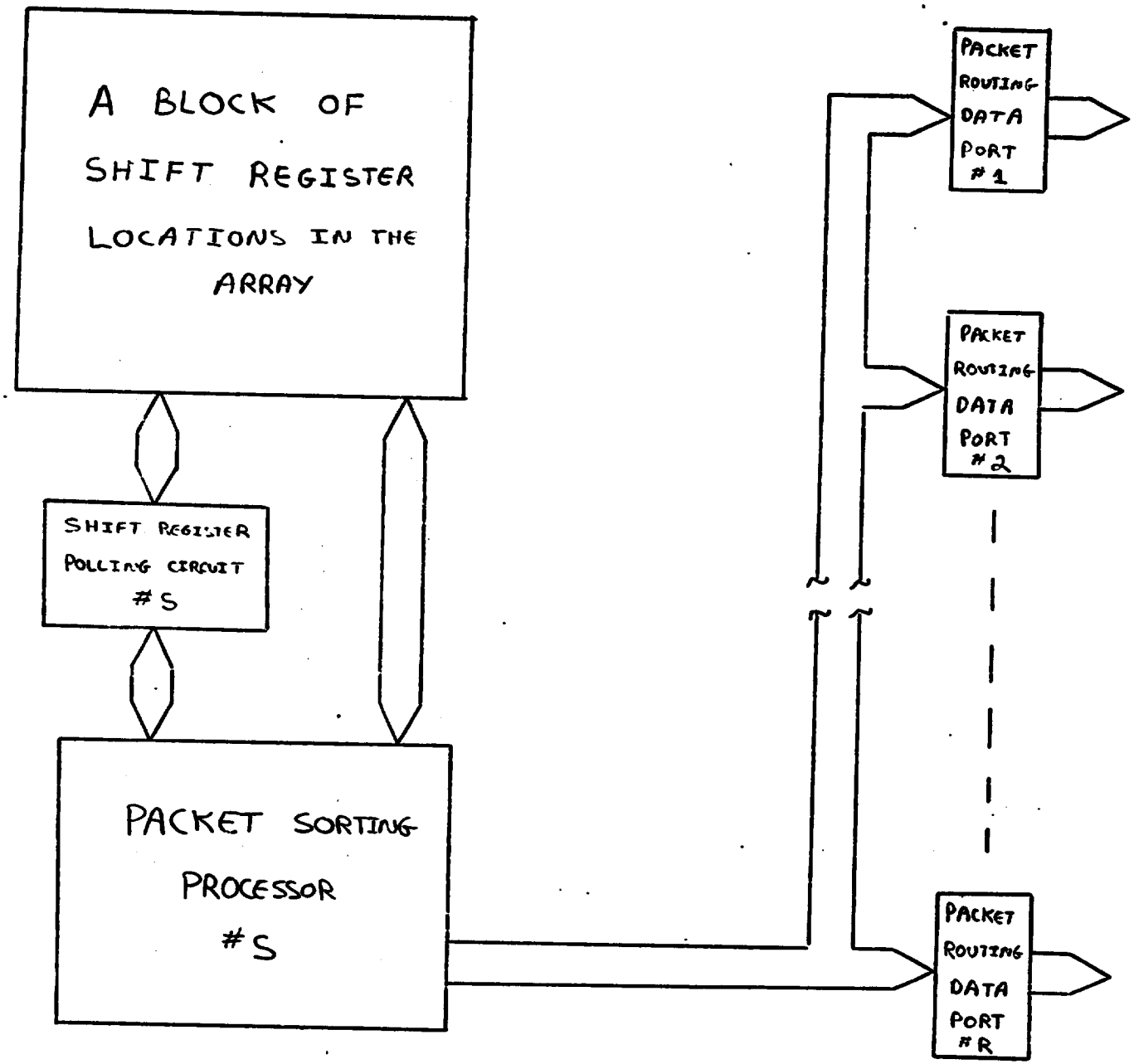


Fig. 4.15 System Architecture for a Single Packet Sorting Processor

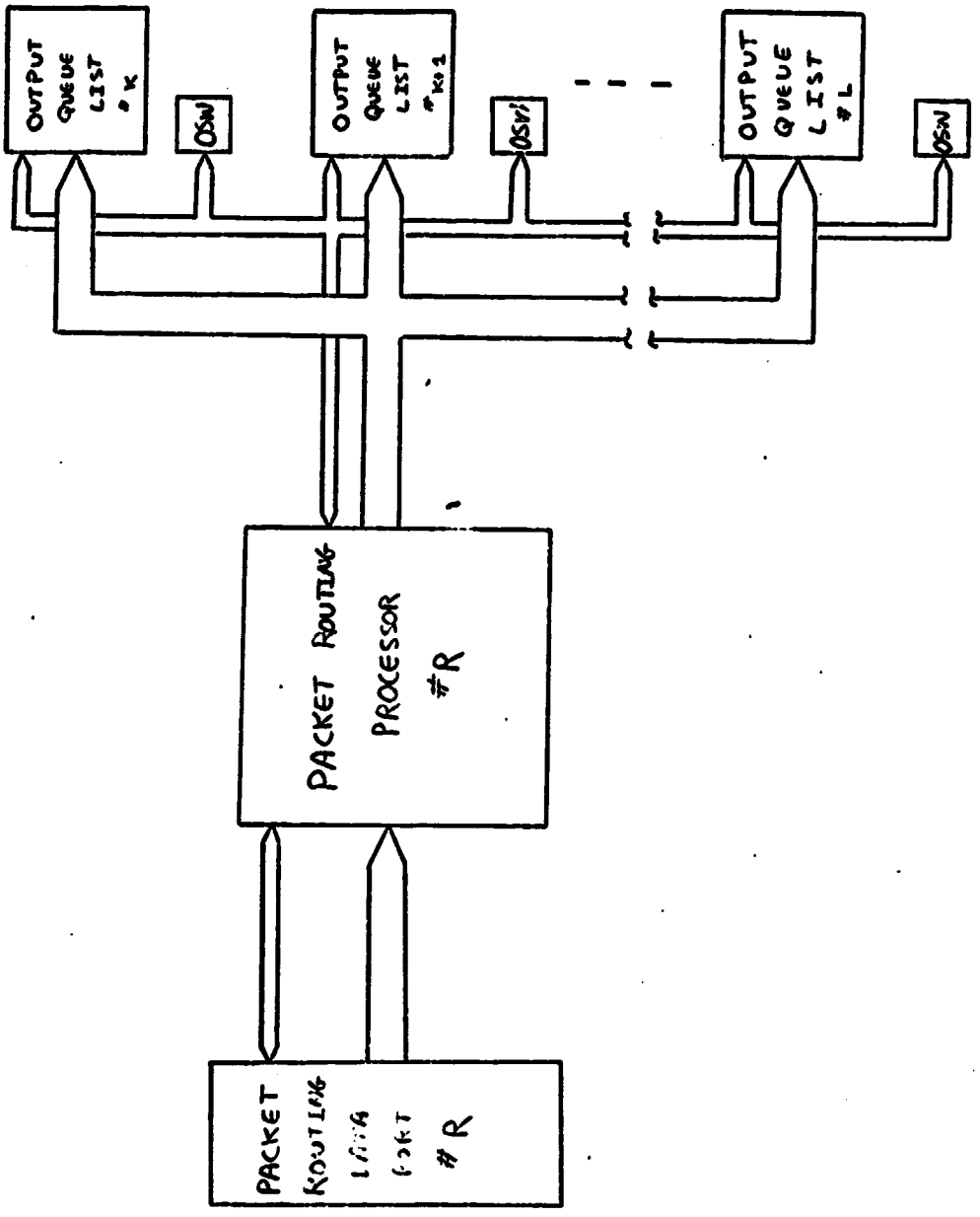


Fig. 4.16 System Architecture for a Single Packet Routing Processor

its own Packet Routing Data Port. Any Packet Sorting Processor can send data to any Packet Routing Data Port. Associated with every Packet Routing Data Port is a dedicated RAM which is external to the Routing Processor it serves. The function of these ports is to accept the routing information from the Packet Sorting Processors, to store the routing information in the external RAM and to provide this data to the Packet Routing Processor when needed.

There exists an alternative to using the external RAM for storage, but it is considered too costly to implement. The alternate scheme requires that whenever a Sorting Processor places data into a Routing Processor's data port, the Routing Processor is to be notified by an interrupt. This interrupt signal is activated by the Sorting Processor. The Routing Processor responds to the interrupt by suspending the Packet Routing Routine in order to fetch the data from the full port. Once fetched, this data is stored in an internal software queue. This scheme is considered too costly because:

1) Additional processor hardware will be required to handle the interrupts.
2) The additional required software overhead will increase execution times and reduce throughput.

These are the reasons the hardware stack scheme is implemented.

The operation of a Packet Routing Data Port can be best explained by tracing the procedure that a Packet Sorting Processor follows to send data to a port. Once a Packet Sorting Processor determines which port is to receive the data, it

checks the associated DAC flag. If this flag is not set, the processor waits for it to be set. When the flag is set, the Packet Sorting Processor sends the packet's destination information to the Packet Destination Data Latch. Next, the Packet Sorting Processor sends the packet's shift register array address to the Packet Array Address Data Latch. Once both latches are loaded, the Packet Sorting Processor sets the DAV flag which automatically clears the DAC flag. A single Packet Routing Data Port is illustrated in Figure 4.17.

Every DAV flag is scanned by a hardware polling circuit. This polling circuit is presented in Figure 4.18. When an activated DAV flag is found by the poller, the STORE DATA flip-flop is set. The set flip-flop halts the poller and activates the one-shot that generates the low-active WRITE signal. The outputs of the two data latches associated with active DAV flags are enabled. The enabled output of the Packet Destination Data Latch is sent to the Packet Destination Data RAM and the output of the Packet Array Address Data Latch is sent to the Packet Array Address Data RAM. Both of these RAM's are enabled in the write mode by the activated WRITE signal. The WRITE signal is held activated until the data is strobed into the RAM's. This data is stored into the two RAM's at locations which have the same address since both RAM's share a single index pointer. The architecture of the Packet Routing Data RAM's is presented in Figure 4.19.

Once the write operation is complete, the active-low WRITE signal goes high. The low-to-high transition of this

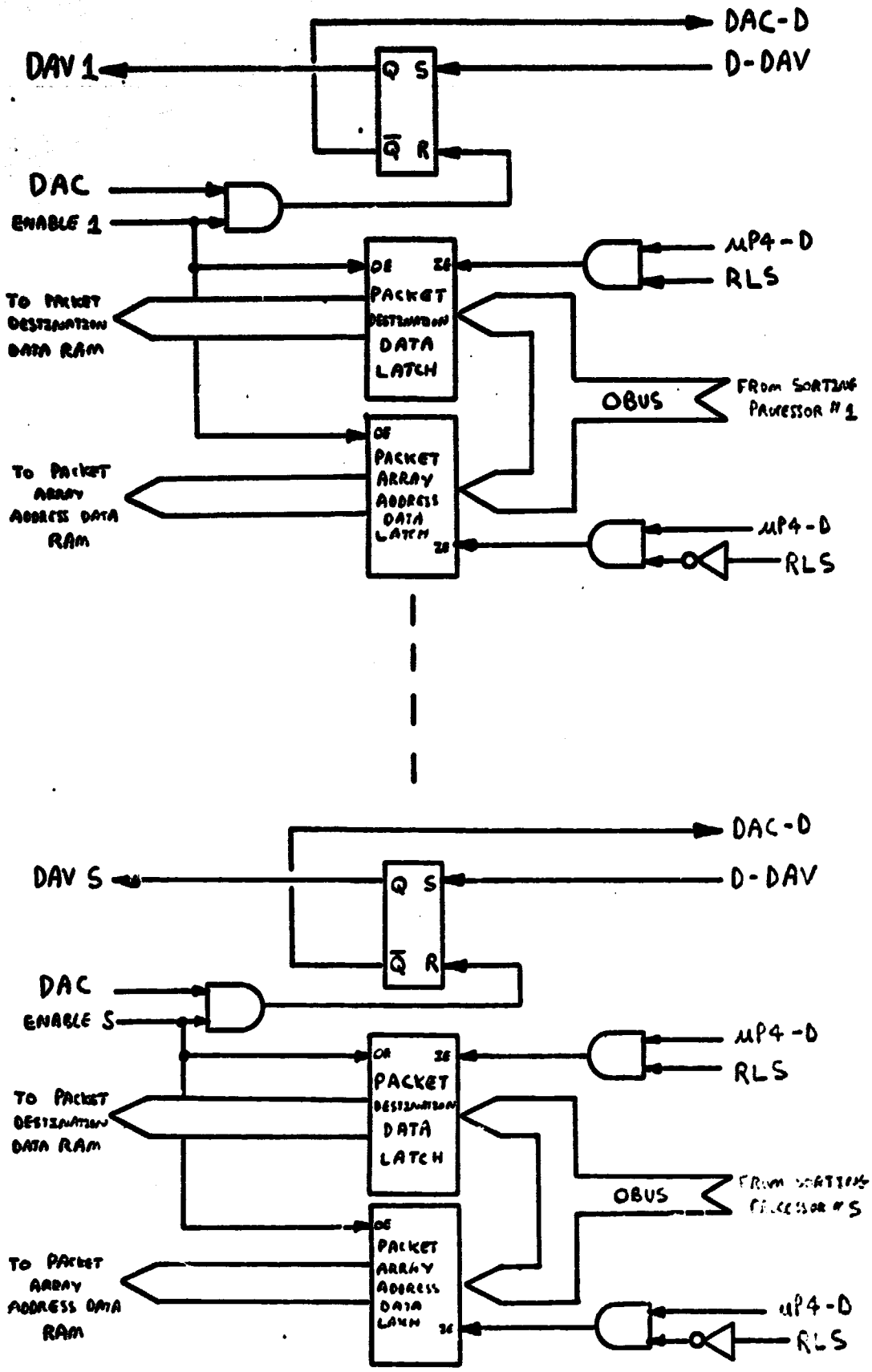


Fig. 4.17 A Single Packet Routing Data Port

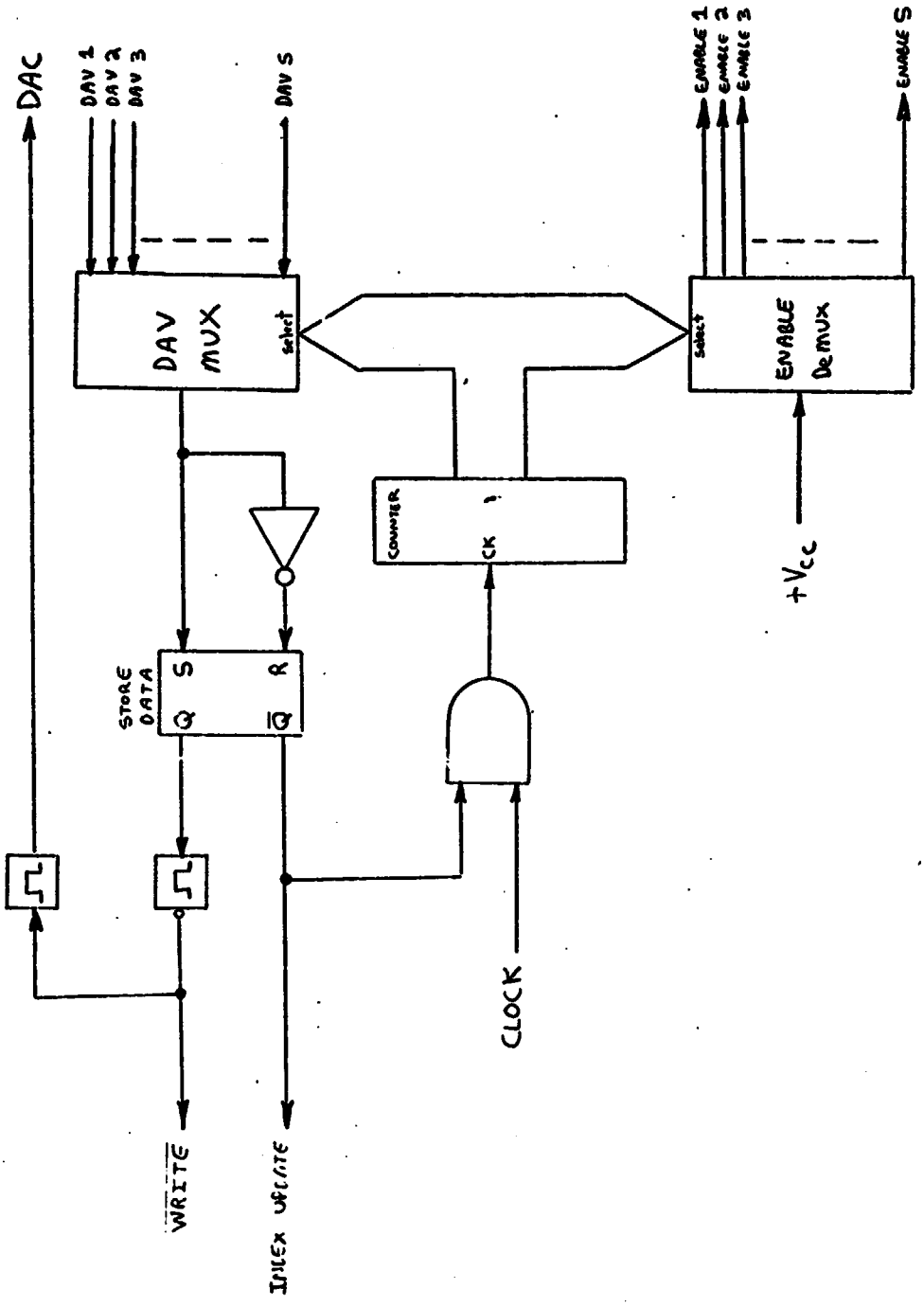


Fig. 4.18 Packet Routing Data Port Polling Circuit

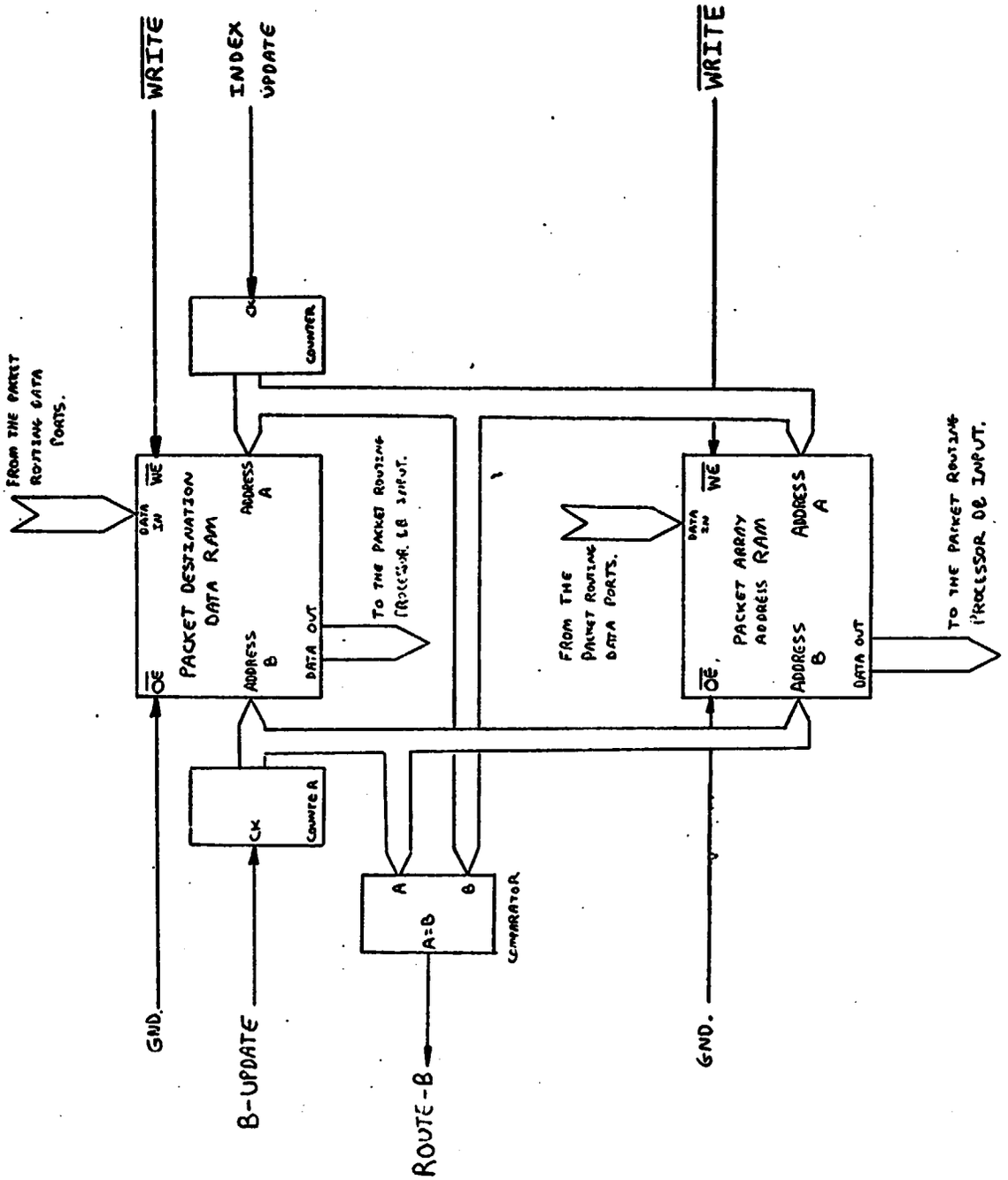


Fig. 4.19 Packet Routing Data RAMs

signal activates the one-shot that generates the Flag UPDATE signal. The Flag UPDATE signal clears the DAV flag and sets the DAC flag. The clearing of the DAV flag resets the STORE DATA flip-flop. The reset flip-flop activates the INDEX UPDATE signal which increments the hardware counter that serves as an index pointer. The data structure for the Packet Routing Data List is given in Figure 4.20. Both the read and the write operations take place before the index pointers are incremented. When the two index pointers are equal, the list is assumed to be empty.

When the Packet Routing Processor needs to fetch data from the RAM's, it first selects the Packet Destination Data RAM and then fetches the data. Next, the Packet Routing Processor selects the Packet Array Address RAM and fetches the data. The processor increments the index pointer once both read operations are finished. Two-port RAM's are used to allow a simultaneous read by the processor and write by the hardware. Since the list is assumed to be empty when the index pointers are equal, a read and a write operation will never occur at the same location.

4.4.3 The Packet Sorting Processors

The Instruction Execution Units and the Microprogram Control Units of the Packet Sorting Processors are similar to those used in the three processor design for the Routing Processor (see section 3.2). The IEU control fields in the Microprogram Word for the Sorting Processors are presented in

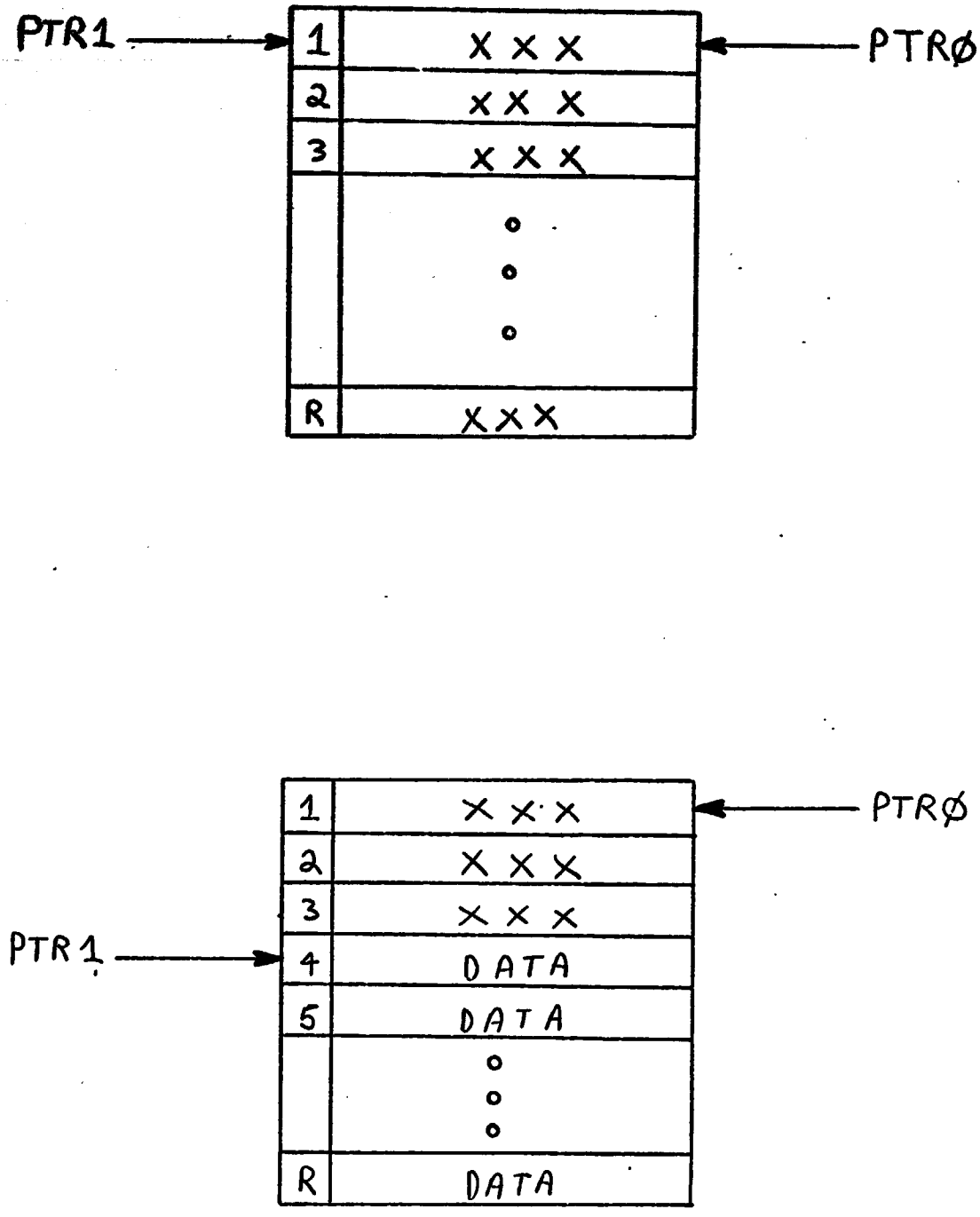


Fig. 4.20 Packet Routing Data List Data Structure

Figure 4.21. Figure 4.22 contains the MCU control fields and the Jump Control Logic Function for the Packet Sorting Processors.

4.4.4 The Packet Sorting Service Routine

The Packet Sorting Service Routine is sense-loop driven. While the Shift Register Array Polling Circuit searches for unserviced packets, the Packet Sorting Processor loops on the test bit. Once an unserviced packet is found by the polling circuit, the Packet Sorting Processor exits from the loop. The processor fetches the address of the packet from the halted poller. The packet's syndrome is fetched and sent as an address to the Syndrome Decoder ROM. Concurrently, the packet's service request flag is cleared and the poller is restarted. The ROM output is fetched and exclusively-ored with the fetched packet header. The corrected header is stored back into the array. Using the corrected header information, the Packet Sorting Processor determines the packet's destination. The destination information is then used to determine which Packet Routing Processor is to receive the packet's routing data. This is accomplished by sending the destination data to the Sorting Processor's Address Decoder. This decoder will generate the address of the Packet Routing Data Port associated with the destination of the sorted packet. Since a Routing Processor may route packets destined for different ground stations, the different destination codes of these packets must generate the address of this Routing Processor's port. Since the different codes

Ø-15	16-19		2Ø	21-24	25	26-3Ø	31-32	33	34	35	36	37	38	39
IMMEDIATE OPERAND	ALU SOURCE		B ADDRESS	ALU FUNCTION	CARRY IN	ALU DESTINATION	BUS LATCH	ROM READ	WRITE ENABLE	OUTPUT ENABLE	ROUTING LATCH SELECT	HEADER WRITE	POWER RESET	DAV
	R SOURCE	S SOURCE												

MNEMONIC FIELDS

$D_{15} - - - D_Ø$	$\overline{EA}$, I_9, $\overline{OE_B}$, $I_Ø$	BAS	$I_4 \cdots I_1$	C_n	$\overline{IEN}$ $I_8 \cdots I_5$	ALE RALE	$\overline{R}$	$\overline{WE}$	$\overline{OE_Y}$	RLS	D-STORE	D-RESET	D-DAV

CONTROL BITS

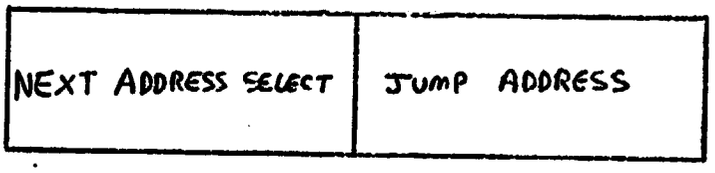
				ALU SOURCE	
$\overline{EA}$	I_9	$\overline{OE_B}$	$I_Ø$	R	S
1	Ø	Ø	Ø	uW	Reg @ B
1	Ø	X	1	uW	Q
1	Ø	1	Ø	uW	Polling Port
1	1	Ø	Ø	IBUS	Reg @ B
1	1	X	1	IBUS	Q
1	1	1	Ø	IBUS	Polling Port

RALE	ALE	BUS LATCH
Ø	Ø	NONE
Ø	1	ADDRESS
1	Ø	ROM

BAS	B SOURCE
Ø	SCRATCH 1
1	SCRATCH 2

RLS	PORT LATCH
Ø	Packet Destination
1	Packet Address

Fig. 4.21 Packet Sorting Processor IEU Microprogram Control Fields



40-41

N_1	N_0	NEXT ADDRESS
0	0	$\mu PC + 1$
0	1	UNCONDITIONAL JUMP
1	0	JUMP ON NEW-D = 0
1	1	JUMP ON DAC-D = 0

42-45

JUMP ADDRESS
JA_3, JA_2, JA_1, JA_0

NEW-D	DAC-D	N_1	N_0	$\overline{FE}$	C_n	S_1	S_0	ADDRESS SOURCE
X	X	0	0	1	1	0	0	$\mu PC + 1$
X	X	0	1	1	0	1	1	JUMP ADDRESS
0	X	1	0	1	0	1	1	JUMP ADDRESS
1	X	1	0	1	1	0	0	$\mu PC + 1$
X	0	1	1	1	0	1	1	JUMP ADDRESS
X	1	1	1	1	1	0	0	$\mu PC + 1$

Fig. 4.22 Packet Sorting Processor MCU Control Fields and Jump Control Logic Function

will enable different address decoder lines, an encoding scheme is needed. WIRE-ANDing the different address lines that must enable the same port will provide the system with the single port address lines needed. The only constraint associated with this scheme is the requirement that the address decoders have low active, open collector outputs.

Once the proper Packet Routing Data Port is addressed, the Sorting Processor checks to determine if the port is empty. If the port still contains valid data, the Packet Sorting Processor waits for the port to be emptied by the automatic port hardware. When the port is empty, the Packet Sorting Processor first sends the packet's destination data to the port. The processor then sends the packet's array address, sets the port's DAV flag and returns to the sense loop. Figure 4.23 contains the flow chart for this routine. A listing of this program is supplied in Figure 4.24.

4.4.5 The Packet Routing Processors

The Microprogram Control Units of the Packet Routing Processors are similar to one used in the three processor design for the Routing Processor (see section 3.2). However, the Instruction Execution Units (IEU) of the Packet Routing Processors are redesigned to handle the Packet Routing Data Ports. Since the Packet Routing Processors need no polling circuits, the Direct Data (DB) is used to supply the processors with the Packet Routing Data. This scheme saves execution cycles since the processors are not required to generate the addresses of the external data RAM's. A single

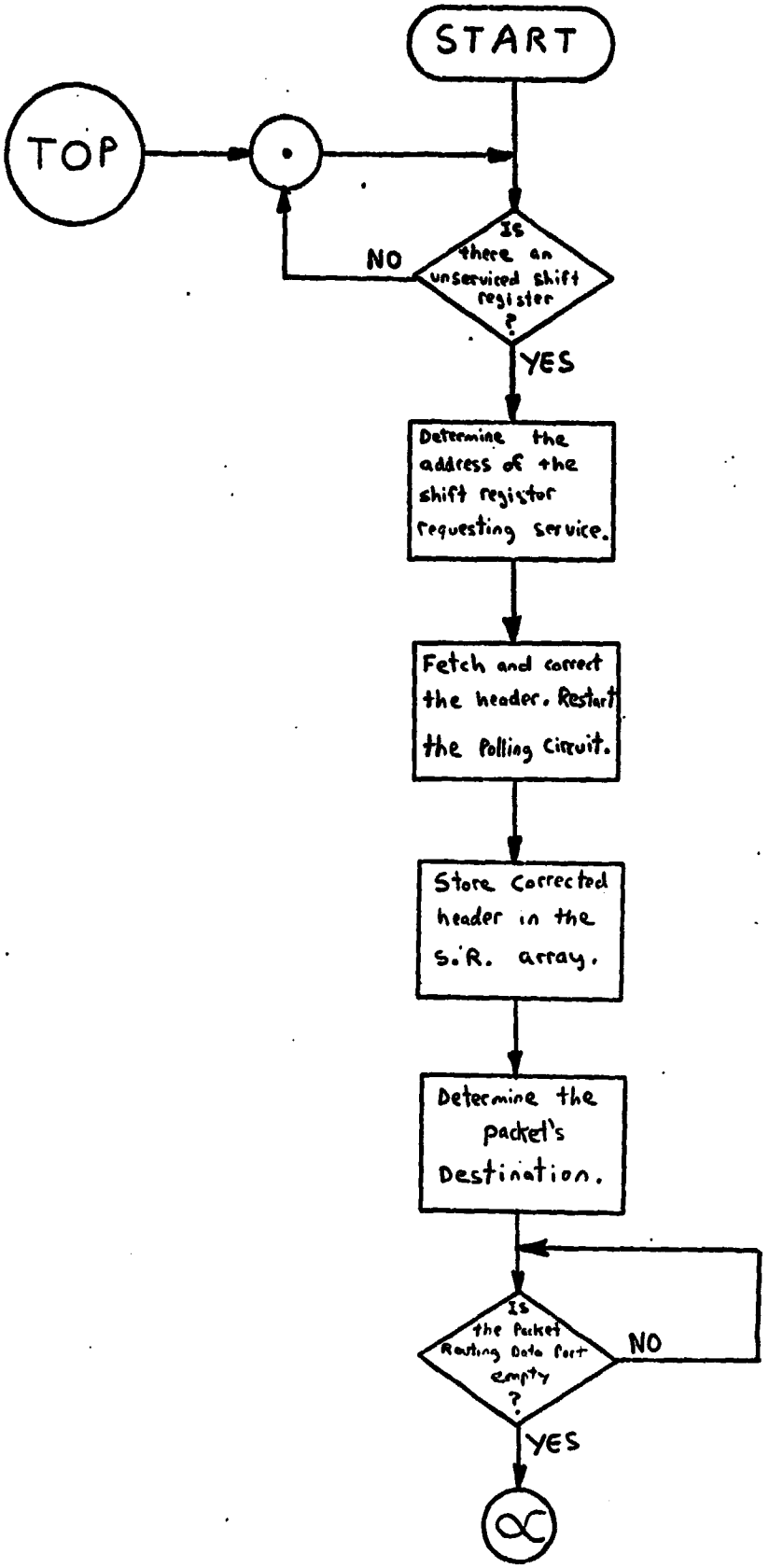


Fig. 4.23 Packet Sorting Service Routine Flowchart

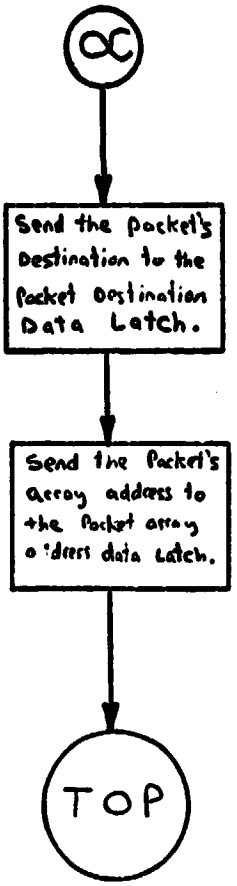


Fig. 4.23 Packet Sorting Service Routine Flowchart, continued

```
START:  If NEW-D=Ø, JMP to START

                                        *Is there a shift register
                                         requiring service?
                                         NO:  Loop @ START.

        SRS Polling Port→Scratch 1

                                        *YES:  Input the address
                                               of the shift register.

        Syndrome Generator Base Address+Scratch 1 Address→Latch (μP1-D)
        Syndrome (R) → Decoder ROM Address Latch; Reset Poller

                                        *Fetch header syndrome and
                                         send it to the Decoder ROM.
                                         clear NEW-0 and restart
                                         the poller.

        Decoder ROM Address→Address Latch (μp2-D)
        [Decoder ROM] @ Syndrome (R)→Q

                                        *Fetch error word from ROM.

        Header Base Address + Scratch 1→Address Latch (μp3-D)
        ALU EXOR Q→Scratch 2, Header Port (R)

                                        *Correct the header.  Store
                                         it into the S.R. Array and
                                         into Scratch 2.

        Scratch 2 AND Destination Mask→Q

                                        *Determine packet destina-
                                         tion

        Q + Packet Routing Processor Base Address→Address Latch
        (μp4-D)

LOOP:   If DAC-D = Ø, JMP to LOOP
        Q→selected Packet Routing Destination Data Port
        Scratch 1→selected Packet Routing Shift Register # Data
        Port; set DAV flag; JMP to START.

                                        *Select the proper Packet
                                         Routing Processor's Data
                                         Port. Send the packet's
                                         destination data.  Then
                                         send the packet's S.R.
                                         array address. Set the port's
                                         DAV flag and return to the
                                         top of the program.
```

Figure 4.24 Packet Sorting Service Routine

control signal from the processor's microprogram word controls the DB SOURCE MUX, which selects either the Packet Destination Data RAM or the Packet Array Address Data RAM. The output from the MUX is tristated because the DB data bus is internally shared by the output of the ALU's register file. The redesigned IEU used by the Packet Routing Processors is displayed in Figure 4.25. The IEU control fields in the Microprogram Word for the Packet Routing Processors are given in Figure 4.26. The MCU control fields in the Microprogram Word and the Jump Control Logic Function are presented in Figure 4.27.

4.4.6 The Packet Routing Service Routine

The Packet Routing Service Routine is sense-loop driven. The Packet Routing Processor loops, testing the status bit which informs the processor when packet routing data is available. When a packet's routing data is available, the Packet Routing Processor leaves the loop. The processor then fetches the packet's destination information. Next, the packet's array address is fetched. Using the destination data, the Packet Routing Processor selects the proper output queue list. Concurrently, the Processor's index pointer for the Packet Routing Data List is incremented. The packet's array address is loaded into the Queue List Data Port by the processor. The Processor then requests access to the queue list. Requests for access are generated until the Packet Routing Processor is allowed to access the queue list. Once access is granted, the hardware automatically strobes the array

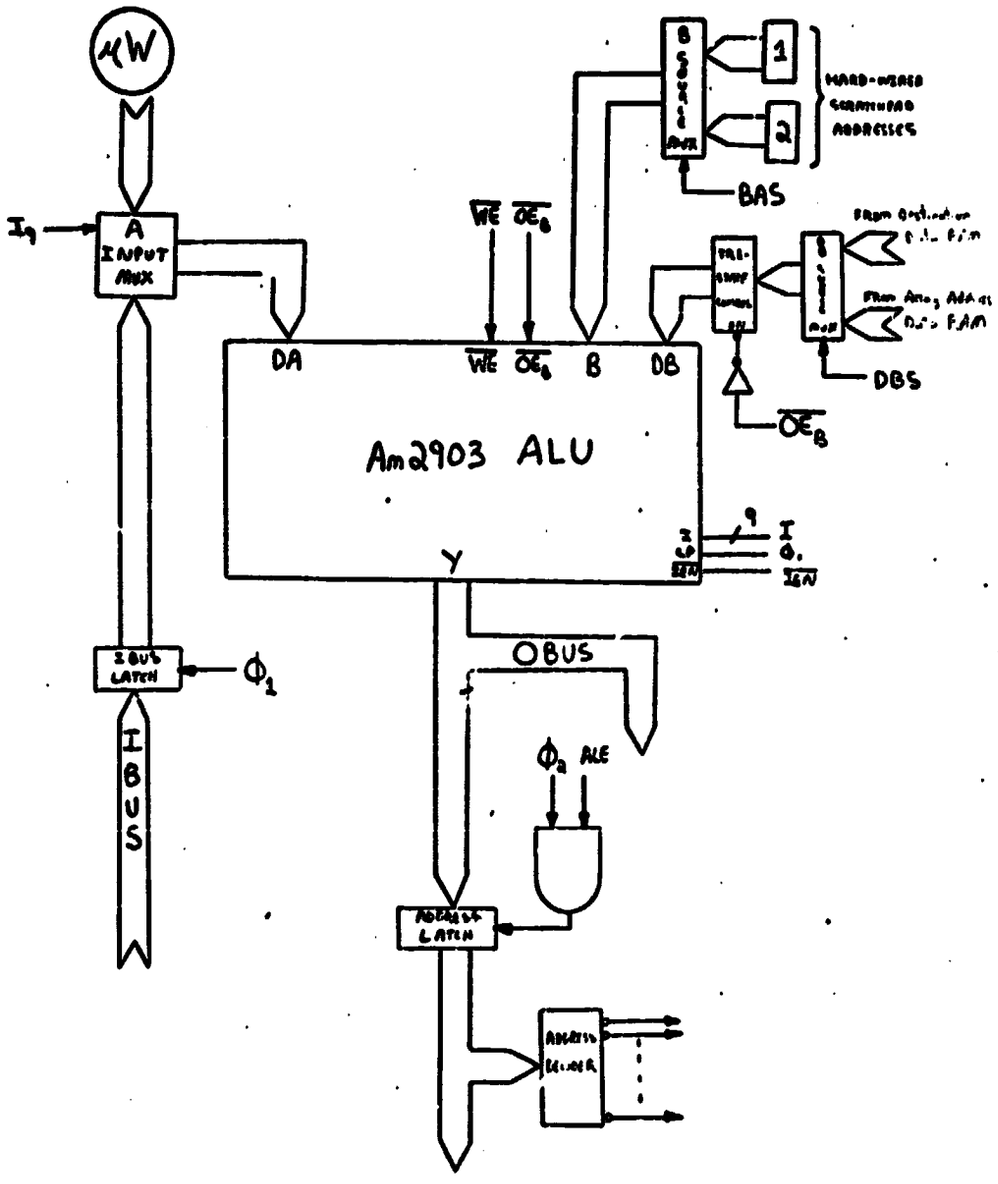


Fig. 4.25 Packet Routing Processor IEU

Ø-15	16-19		2Ø	21	22-25	26	27-31	32	33	34	35	36	37-39
IMMEDIATE OPERAND	ALU SOURCE R SOURCE	ALU SOURCE S SOURCE	B ADDRESS SOURCE	DB SOURCE	ALU FUNCTION	CARRY IN	ALU DESTINATION	BUS LATCH ENABLE	WRITE ENABLE	OUTPUT ENABLE	INDEX UPDATE	OSW CONTROL	OUTPUT QUEUE LIST CONTROL
D_{15} — — — D_0	$\overline{EA}$ I_9, $\overline{OE_B}$, I_8		BAS	DBS	$I_4 \cdots I_1$	C_n	$\overline{IEN}$, $I_8 \cdots I_5$	ALE	$\overline{WE}$	$\overline{OE_Y}$	B-WRITE	B-READ	E-LOAD E-REQUEST E-RELEASE

				ALU SOURCE	
$\overline{EA}$	I_9	$\overline{OE_B}$	I_8	R	S
1	Ø	Ø	Ø	μW	Reg@B
1	Ø	1	Ø	μW	DB source
1	1	Ø	Ø	IBUS	Reg@B
1	1	1	Ø	IBUS	DB source

BAS	B SOURCE
Ø	SCRATCH 1
1	SCRATCH 2

DBS	DB SOURCE
Ø	Instr. Addr. RAM
1	S.R.P. RAM

ALE	BUS LATCH
Ø	NONE
1	ADDRESS

Fig. 4.26 Packet Routing Processor IEU Microprogram Control Fields

NEXT ADDRESS SELECT	JUMP ADDRESS

40 - 42

N_1	N_0	J	NEXT ADDRESS
0	0	0	μPC + 1
0	0	1	UNCONDITIONAL JUMP
0	1	X	JUMP ON ROUTE-B = 1
1	0	X	JUMP ON STATUS-B = 1
1	1	X	JUMP ON IDLE-B = 1

43 - 46

JUMP ADDRESS
JA_3, JA_2, JA_1, JA_0

ROUTE-B	STATUS-B	IDLE-B	N_1	N_0	J	FE	C_n	S_1	S_0	ADDRESS SOURCE
X	X	X	0	0	0	1	1	0	0	μPC + 1
X	X	X	0	0	1	1	0	1	1	JUMP ADDRESS
0	X	X	0	1	X	1	1	0	0	μPC + 1
1	X	X	0	1	X	1	0	1	1	JUMP ADDRESS
X	0	X	1	0	X	1	1	0	0	μPC + 1
X	1	X	1	0	X	1	0	1	1	JUMP ADDRESS
X	X	0	1	1	X	1	1	0	0	μPC + 1
X	X	1	1	1	X	1	0	1	1	JUMP ADDRESS

Fig. 4.27 Packet Routing Processor MCU Control Fields and Jump Control Logic Function

address data from the port into the queue list RAM. Meanwhile, the Packet Routing Processor checks the status of the queue list's corresponding output buffer. If the buffer is in the Idle state, the processor updates the buffer's status to the Empty state, releases the queue list and returns to the sense loop. However, if the buffer is not in the Idle state, the processor simply releases the queue list and returns to the loop. The flow chart for this software routine is shown in Figure 4.28. A listing of this program is given in Figure 4.29.

4.5 The Output System

The Output System consists of the Output Buffers, the Output Processors, the Output Switching Networks, and the Output Polling Circuits. Interfacing to this system are the Output Queue Lists, the Shift Register array and the ELIST Data Distribution System. The architectural organization of this system is presented below.

4.5.1 Architectural Workload Division

The two major system constraints that influence the architectural organization of this sytem are:

1) Only one Output Processor must control an output buffer. Each output buffer must be assigned to only one Output Processor in order to eliminate resource contention.
2) Only one Output Processor can have access to an output queue list.

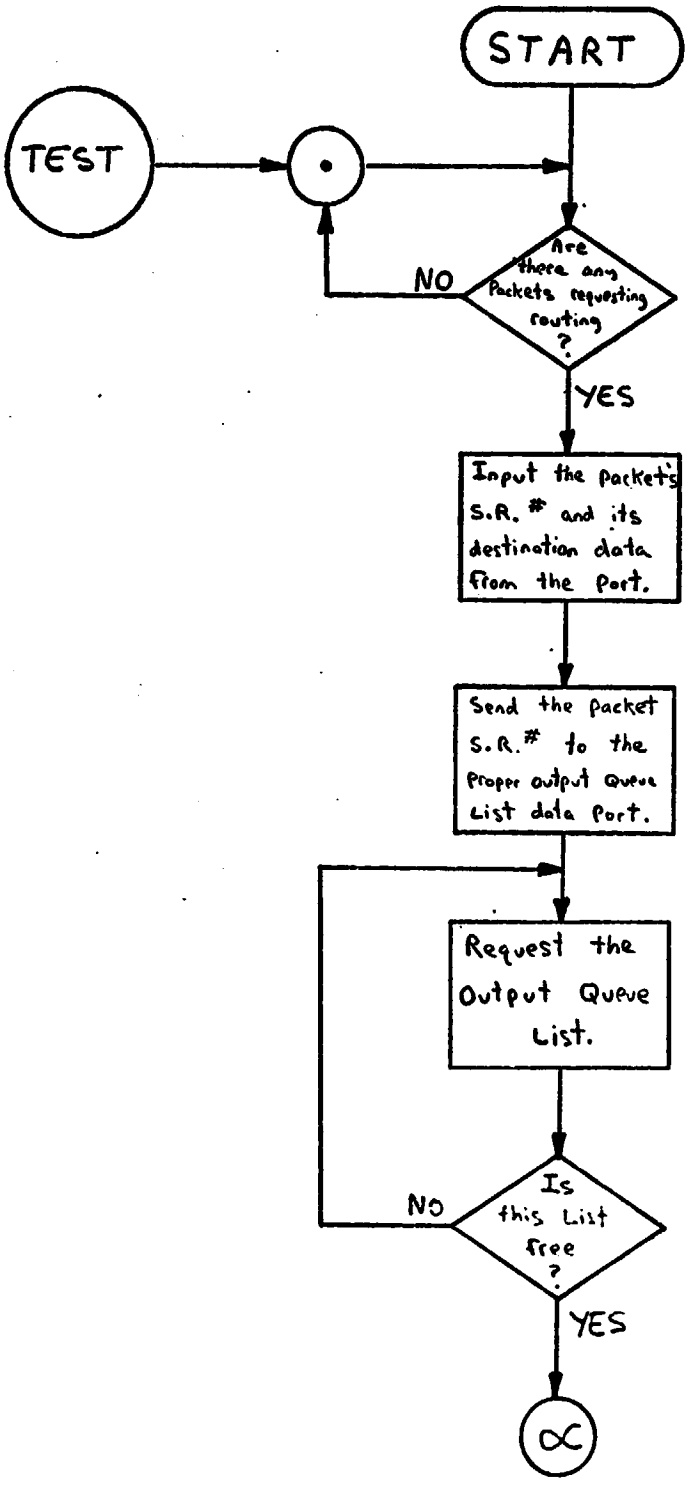


Fig. 4.28 Packet Routing Service Routine Flowchart

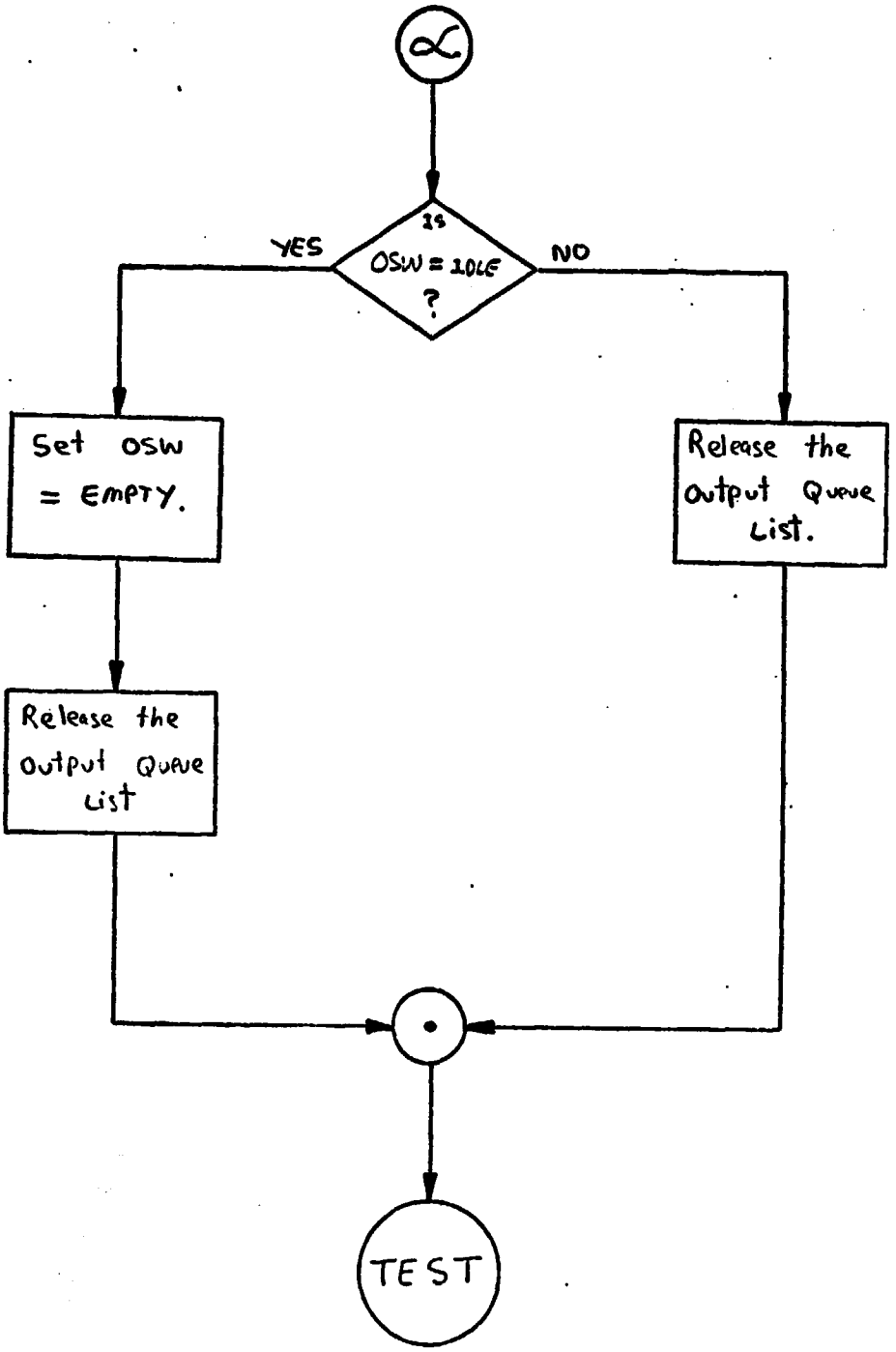


Fig. 4.28 Packet Routing Service Routine Flowchart, continued

```
TEST: If ROUTE-B = 1, JMP to TEST

                                        *Are there any packets re-
                                         questing routing?
                                         NO: Loop @ TEST

      Destination Data RAM → Scratch 1
      Shift Register Address RAM → Scratch 2

                                        *YES: Input the packet's des-
                                         tination and array address.

      Scratch 1+Output Queue List Base Address→Address Latch (μP1-B);
      update packet data pointer

                                        *Select the Output Queue List
                                         and OSW of the destination
                                         buffer

      Scratch 2 → Output Queue List Data Port (N)

                                        *Send the packet's array address
                                         to the Output Queue List Data
                                         Port.

REQUEST: Request Queue List (N)

                                        *Request access to the Output
                                         Queue List selected.  If access
                                         is granted, the data from the
                                         Port is automatically stored.

      If STATUS-B = 1, JMP to REQUEST

                                        *If access is not granted, Loop
                                         @ REQUEST.  Proceed otherwise.

      If OSW = NOT IDLE, JMP to END

                                        *Is the output buffer idle?

      Set OSW=EMPTY; Release Output Queue List (N); JMP to TEST

                                        *YES: Update OSW, release Queue
                                         List and return to the top of
                                         the routine.

  END: Release Output Queue List (N); JMP to TEST

                                        *NO: Release queue list and re-
                                         turn to the top of the routine.
```

Fig. 4.29 Packet Routing Service Routine

The Separate Systems Scheme as discussed in section 4.3.1 is considered the best technique to use in organizing the Output System in order to fulfill the above requirements. The system architecture of a single Output Processor in the Separate System Scheme appears in Figure 4.30. Each output processor is assigned to a fixed number of output buffers. In addition, each processor is assigned a dedicated Output Switching Network, a dedicated Output Polling Circuit and is allowed access to the Output Queue Lists that corresponded to the assigned output buffers. Since the implementation of this architecture did not require the redesigning of the Output Buffers, the Output Polling Circuits or the Output Switching Networks, these hardware blocks are not discussed in detail in this chapter (see section 3.1 for a review). The Output Processors and their software are discussed below.

4.5.2 The Output Processors

Both the Instruction Execution Units and the Microprogram Control Units used by the Output Processors are similar to those used by the Output Processors in the three processor architecture (see section 3.2). Shown in Figure 4.31 are the IEU control fields of the Output Processors' Microprograms Word. Figure 4.32 displays the MCU control fields of the Microprogram Word and also the Jump Control Logic Function for this class of processor.

4.5.3 The Output Service Routine

Like all the software routines, the Output Service Routine is also sense-loop driven. The Output Processor

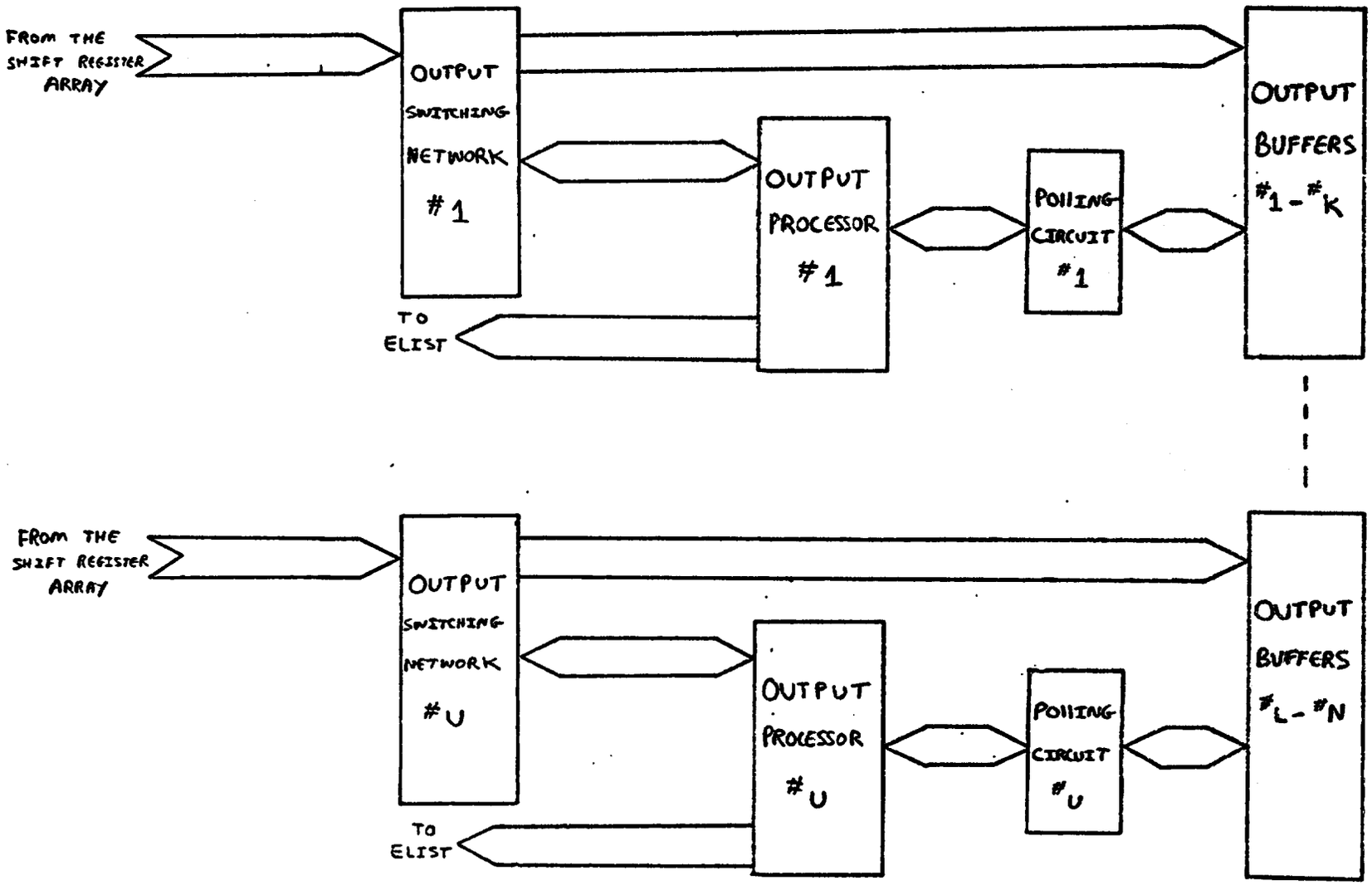
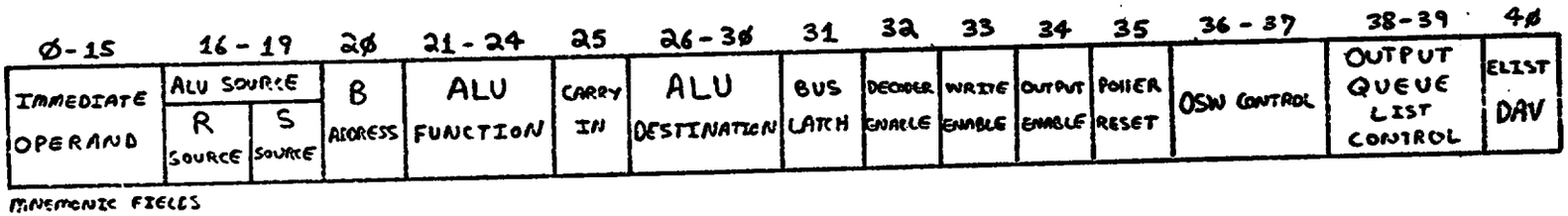
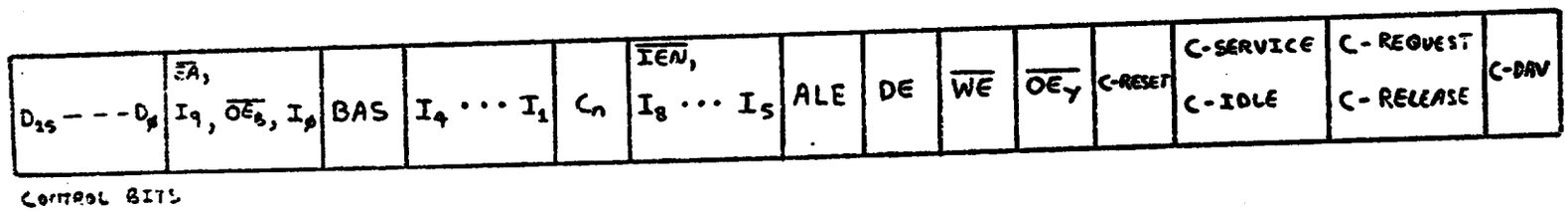


Fig. 4.30 System Architecture for a Single Output Processor





				ALU SOURCE	
EA	I_9	OE_B	$I_Ø$	R	S
1	Ø	Ø	Ø	µW	Reg @ B
1	Ø	X	1	µW	Q
1	Ø	1	Ø	µW	Polling Port
1	1	Ø	Ø	IBUS	Reg @ B
1	1	X	1	IBUS	Q
1	1	1	Ø	IBUS	Polling Port

ALE	BUS LATCH
Ø	NONE
1	ADDRESS

BAS	B SOURCE
Ø	SCRATCH 1
1	SCRATCH 2

Fig. 4.31 Output Processor IEU Microprogram Control Fields

NEXT ADDRESS SELECT	JUMP ADDRESS

41 - 43

N_1	N_0	J	NEXT ADDRESS
Ø	Ø	Ø	μPC + 1
Ø	Ø	1	UNCONDITIONAL JUMP
Ø	1	Ø	JUMP ON SERVICE-C = Ø
Ø	1	1	JUMP ON STATUS-C = 1
1	Ø	Ø	JUMP ON DAC-C = Ø
1	Ø	1	JUMP ON EMPTY-C = Ø

44-48

JUMP ADDRESS
$JA_4, JA_3, JA_2, JA_1, JA_Ø$

SERVICE-C	STATUS-C	DAC-C	EMPTY-C	N_1	N_0	J	$\overline{FE}$	C_n	S_1	$S_Ø$	ADDRESS SOURCE
X	X	X	X	Ø	Ø	Ø	1	1	Ø	Ø	μPC + 1
X	X	X	X	Ø	Ø	1	1	Ø	1	1	JUMP ADDRESS
Ø	X	X	X	Ø	1	Ø	1	Ø	1	1	JUMP ADDRESS
1	X	X	X	Ø	1	Ø	1	1	Ø	Ø	μPC + 1
X	Ø	X	X	Ø	1	1	1	1	Ø	Ø	μPC + 1
X	1	X	X	Ø	1	1	1	Ø	1	1	JUMP ADDRESS
X	X	Ø	X	1	Ø	Ø	1	Ø	1	1	JUMP ADDRESS
X	X	1	X	1	Ø	Ø	1	1	Ø	Ø	μPC + 1
X	X	X	Ø	1	Ø	1	1	Ø	1	1	μPC + 1
X	X	X	1	1	Ø	1	1	1	Ø	Ø	JUMP ADDRESS

Fig. 4.32 Output Processor MCU Control Fields and Jump Control Logic Function

leaves the loop once its polling circuit locates an empty output buffer. After leaving the loop, the processor fetches the address of the buffer from the halted poller. Using this information, the Output Processor selects the buffer's corresponding output queue list. A request for access to this list is generated by the processor until access is granted. When access is granted, the Output Processor determines if the queue list is empty. If the processor finds the list empty, the processor changes the buffer's status from the Empty state to the Idle state. Concurrently, the processor releases the queue list, restarts the poller and returns to the loop.

However, if the queue list accessed is not empty, the Output Processor fetches the address of the packet to be transmitted. Simultaneously, the output buffer's status is changed from the Empty state to the Busy state, the queue list is released and the poller is restarted. After the Output Processor has completed all these tasks, it finds a free data path in its dedicated Output Switching Network. This data path is linked to the shift register containing the packet to be transmitted. The Output Processor then links the empty buffer to the data path. Once the path is complete, the processor initiates the packet's transfer into the output buffer. While this transfer is taking place, the Output Processor checks the status of its ELIST Data Distribution I/O port. If the Data Accepted (DAC) flag is not set, the processor loops until it becomes set. Once the Output

Processor finds the flag set, it sends the array address of the freed shift register. After loading the I/O port, the processor sets the port's DAV flag and returns to the sense loop. Figure 4.33 contains the flow chart for this routine and the listing of this program appears in Figure 4.34.

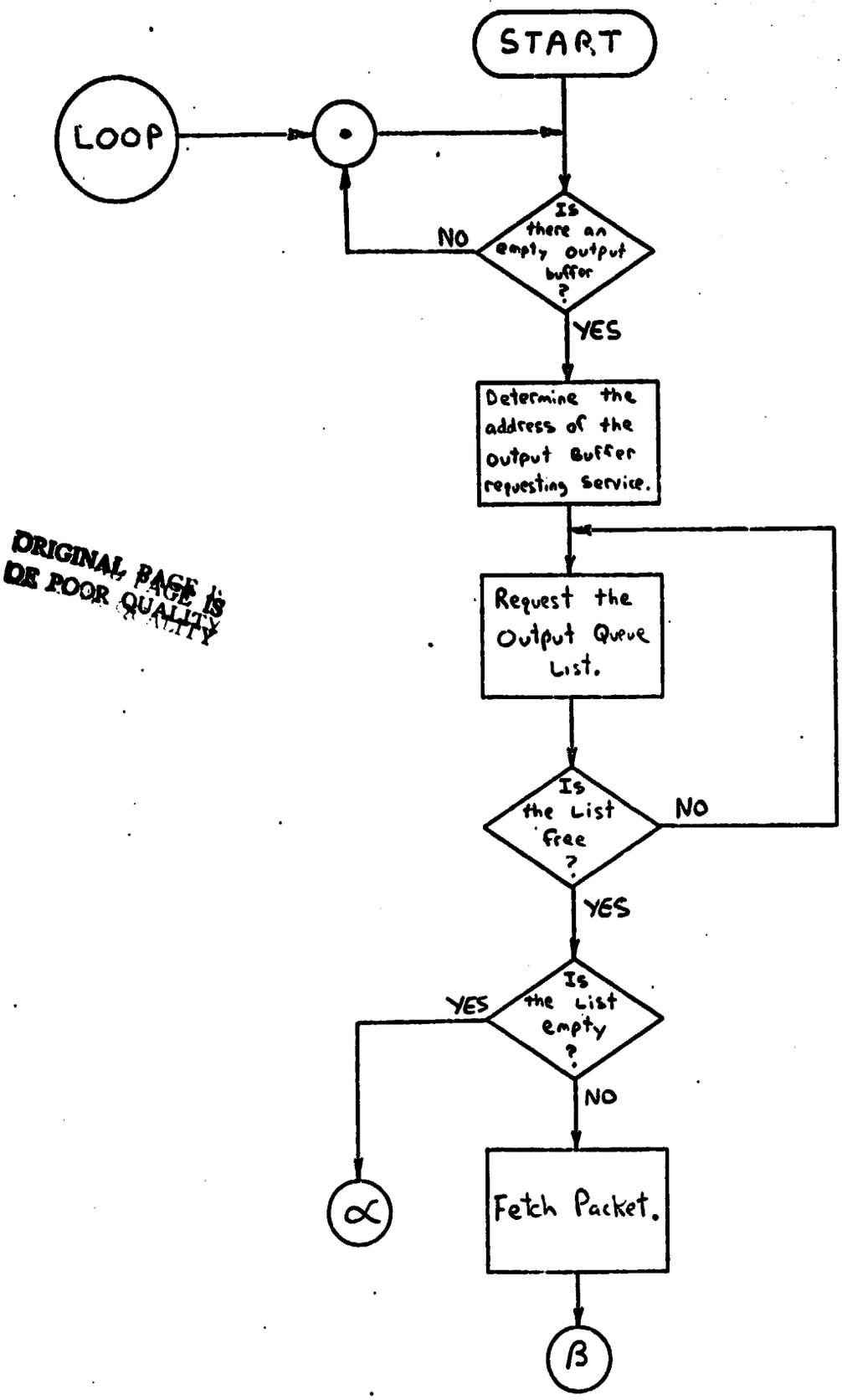


Fig. 4.33 Output Service Routine Flowchart

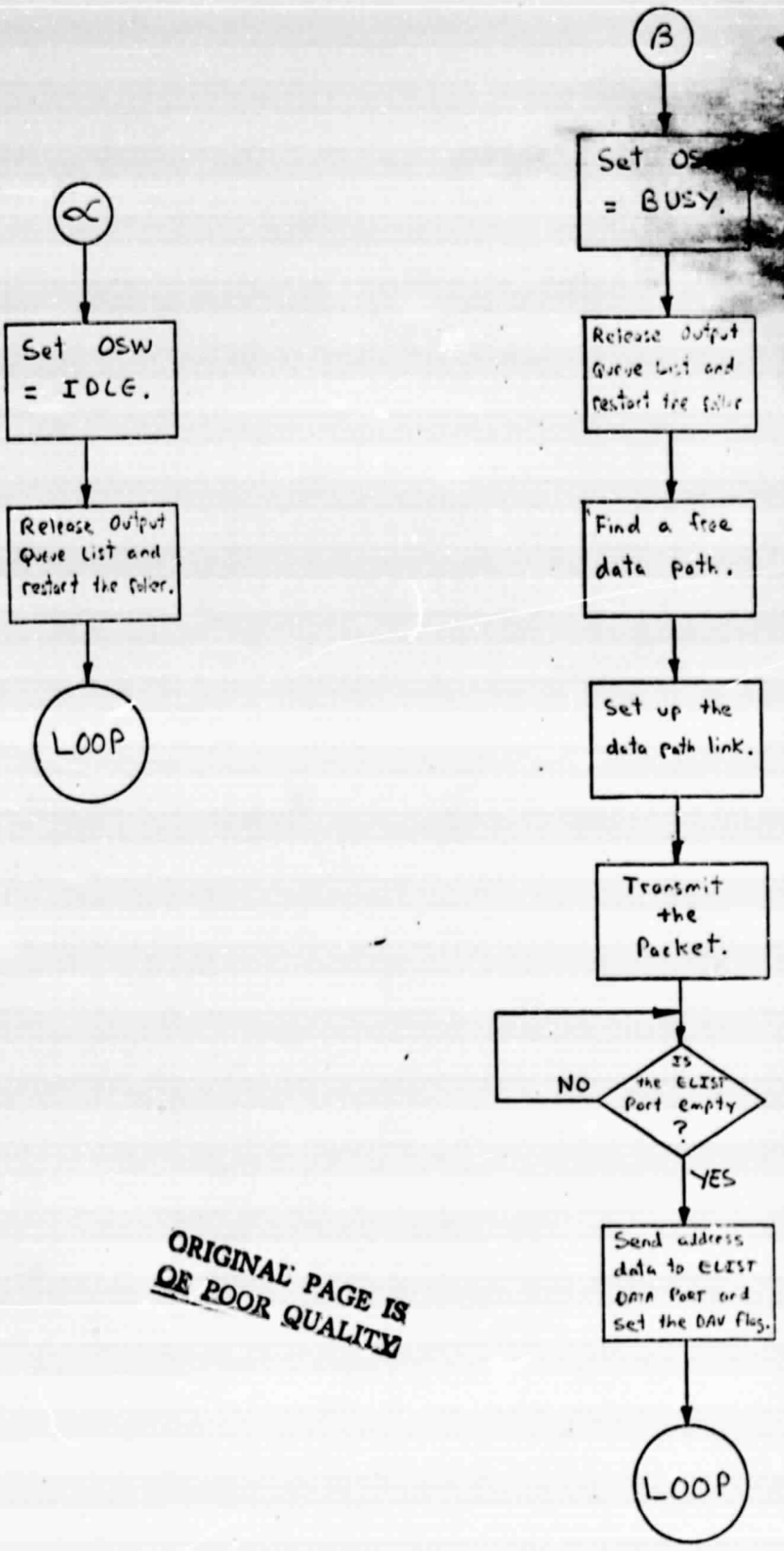


Fig. 4.33 Output Service Routine Flowchart, continued.

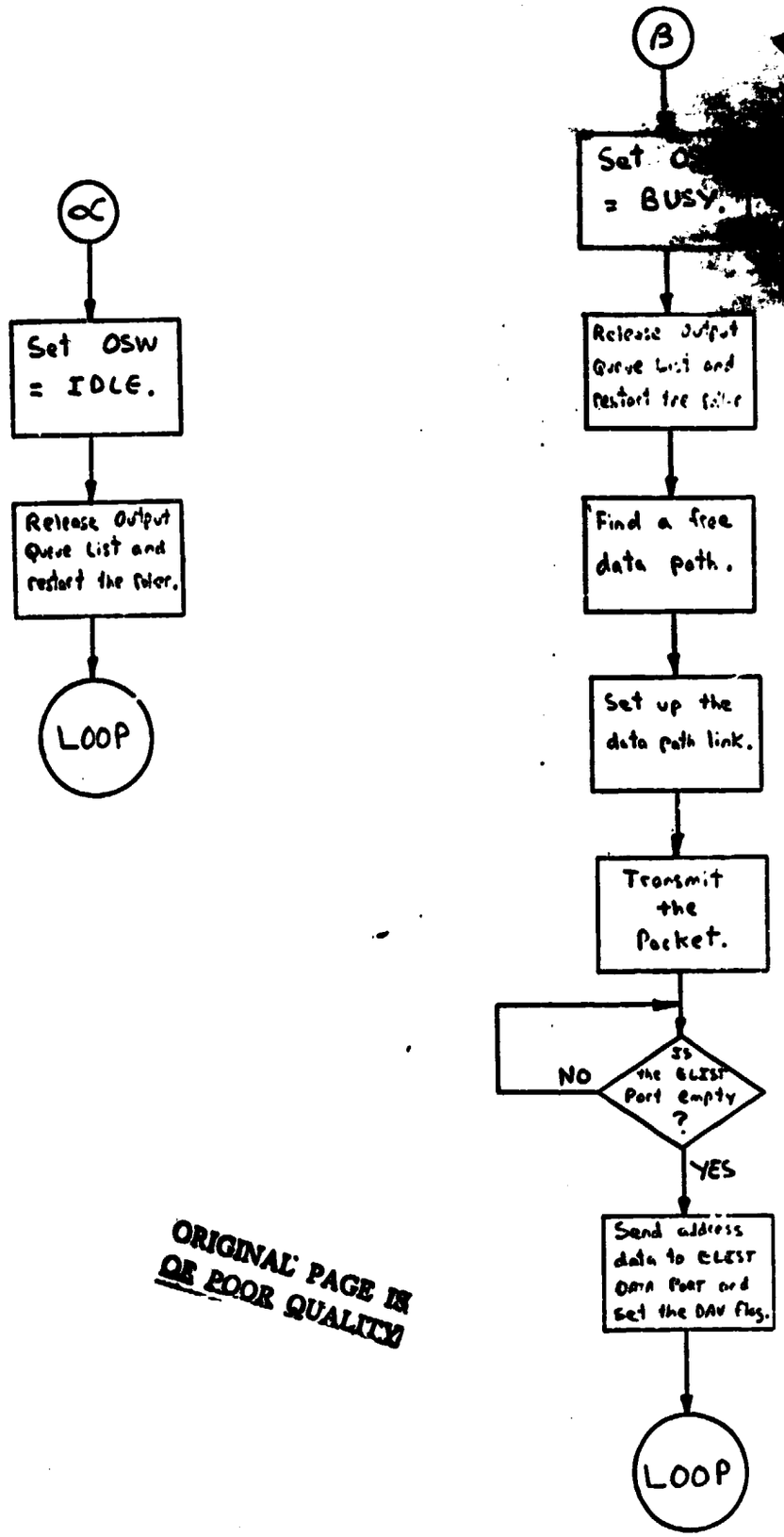


Fig. 4.33 Output Service Routine Flowchart, continued.

```
OUTPUT:  If SERVICE-C = Ø, JMP to OUTPUT

                                        *Is there an output buffer
                                         requesting service?
                                         NO:  Loop @ OUTPUT

         Output Polling Port→Q

                                        *YES:  Input the address of
                                         the buffer.

         Q + Queue List Base Address→Address Latch (μp1-C)

REQUEST: Request Output Queue List (N)

                                        *Select the buffer's Output
                                         Queue List and OSW.  Then
                                         request access.

         If STATUS-C=1, JMP to REQUEST

                                        *Was access granted?
                                         NO:  Request access again.

         If EMPTY-C=Ø, JMP to IDLE

                                        *YES:  Determine if the list
                                               is empty.
                                         List Empty:  Branch to IDLE

         [Output Queue List (N)] @ OPTR (N)→Scratch 1; Set
         OSW=BUSY; Release Output Queue List; Reset Poller

                                        *LIST NOT EMPTY:  Input the
                                         S.R.# which contains the
                                         packet to be transmitted.
                                         Then update the OSW, restart
                                         the poller and release the
                                         queue list.

         Output Path Status Port Address→Address Latch (μp2-C)
         Data Path Busy Status Port→Scratch 2

                                        *Find a free data path.

         Scratch 2+Data Path Latch A Base Address→Address Latch (μp3-C
         Scratch 1 → Data Path MUX select Latch A(D)

                                        *Link the shift register to
                                         the data path.
```

Fig. 4.34 Output Service Routine

```
                Scratch 2→Data Path Latch B Base Address→Address Latch (µp4-C
                Q → Data Path DeMUX select Latch B(D)

                                        *Link the output buffer to
                                         the data path.

                Data Path Transmit Control Base Address→Address Latch
                Scratch 2→Data Bus Decoder (M1-C)

                                        *START Packet transfer.

                ELIST Data Port Address→Address Latch (µp5-C)

LOOP:           If DAC-C=Ø, JMP to LOOP
                Scratch 1→ELIST Data Port

                                        *Send the empty S.R.# to the
                                         ELIST data port when the
                                         port is empty.

                Send a DAV; JMP to OUTPUT

                                        *Send a DAV to the port and
                                         return to the top of the
                                         program.

IDLE:           Set OSW=IDLE; Release Output Queue List; Reset poller;
                JMP to OUTPUT

                                        *Update OSW, release queue
                                         list, restart poller and
                                         return to the top of the
                                         program.
```

Fig. 4.34 Output Service Routine, continued.

5.0 EVALUATION AND THROUGHPUT ANALYSIS

The evaluations of the two packet switch architectures are presented in this chapter. The evaluation of the packet switch's performance is in terms of throughput. This evaluation is based on the software execution times. In the multiple processor architecture, additional parameters affect the system throughput. Therefore, equations relating the number of processors and the number of users to the system throughput are presented.

5.1 Performance Evaluation

In order to compute the maximum system throughput, two assumptions must be made. Both assumptions hold true for the two architectures. The first assumption is that the system is heavily loaded such that all output queues contain at least one packet awaiting transmission. The second assumption arises from the fact that processors never wait for internal hardware and that each system is virtually free from resource contention. Thus, each processor is assumed to be busy 100% of the time under heavily loaded conditions. Therefore, a processor can process one packet in the amount of time required to execute the assigned software routine completely without interruption. Using these assumptions, an estimation of throughput for each multiprocessor architecture is presented below.

5.1.1 Throughput Estimation for the Three Processor System

In order to estimate the system throughput, equations and relationships are developed. In these calculations, system

parameters are introduced. These parameters are:

1) t_{P1} = Input Service Routine execution time

2) t_{P2} = Routing Service Routine execution time

3) t_{P3} = Output Service Routine execution time

4) R = Bit Rate per user

5) N = Number of Users

6) B = Number of Bits per Packet

7) F_P = System Throughput in Packets per Second

8) F_B = System Throughput in Bits per Second

A processor can process one packet in the amount of time required to execute the assigned software routine. Since each packet must be serviced by all three routines, the processor with the longest execution time will determine the maximum system throughput. The software execution time for each processor is listed in Table 5.1. A processor clock cycle of 120 nanoseconds is assumed. Table 5.1 shows the number of instruction cycles required and the time taken. Some routines have several execution times listed. Each of the different values illustrate the various effects of resource contention, the state of the output queue lists and the state of the output buffers.

Normal Operation (No memory contention):	
Input Service Routine	11 cycles = 1.32 μ Sec
Output Service Routine	
(a) Transmit Packet	16 cycles = 1.92 μ Sec
(b) Empty Queue	7 cycles = Ø.84 μ Sec
Packet Routing Service Routine	
(a) Enqueue Packet	15 cycles = 1.8Ø μ Sec
(b) Enqueue Packet and Update OSW	15 cycles = 1.8Ø μ Sec
Worst Case Due to Memory Contention:	
Input Service Routine	11 cycles = 1.32 μ Sec
Output Service Routine	
(a) Empty Queue	Ø cycles
(b) Transmit Packet	18 cycles = 2.16 μ Sec
Packet Routing Service Routine	
(a) Enqueue Packet (Default)	19 cycles = 2.28 μ Sec
(b) Enqueue Packet and Update OSW (Default)	19 cycles = 2.28 μ Sec
(c) Enqueue Packet	17 cycles = 2.Ø8 μ Sec
(d) Enqueue Packet and update OSW	17 cycles = 2.Ø8 μ Sec

Table 5.1 Software Execution Times for the Three Processor System

As stated earlier, the packet switch's maximum throughput is achieved when the processors are busy 100% of the time and when no output queue lists are empty. Therefore, in order to determine the maximum throughput, the slowest execution time must be selected from one of the following values:

1) The execution time for the Input Service Routine under normal operating conditions.
2) The execution time for the Packet Routing Routine when it enqueues a packet under normal operating conditions.
3) The execution time for the Output Service Routine when it transmits a packet under normal operating conditions.

Selecting and comparing the above values from Table 5.1, the execution time for the Output Processor is found to be the largest of the three values. Therefore, the three processor system has a maximum throughput which is limited by:

$$F_P < 1/t_{p3} \quad . \tag{5.1}$$

The system throughput in terms of bit rate is found by multiplying the maximum packet throughput by the packet bit length:

$$B \times F_P = F_B < B/t_{p3} \quad . \tag{5.2}$$

The system throughput in terms of bit rate is related to the number of users by:

$$F_B = N \times R \quad . \tag{5.3}$$

This can be expressed as:

$$N \times R < B/t_{p3} \quad . \tag{5.4}$$

Or,

$$t_{p3} < B/(N \times R) \quad . \tag{5.5}$$

In a heavily loaded system free from resource contention, the Output Processor services one packet every 1.92 microseconds. Therefore, the maximum packet throughput is:

$$F_P < 1/1.92 \ \mu\text{Seconds} = 520{,}833 \text{ packets/second} \tag{5.6}$$

If a packet length of 10,240 bits/packet is used, the maximum system bit rate is:

$$F_B = 10{,}240 \times F_P = 5.3 \times 10^9 \text{ bits/second.} \tag{5.7}$$

An important point to note is that the system is designed such that the processing time of each packet is independent of the packet size. Therefore, an increase in the

packet length will increase the system bit rate proportionally. However, due to the two internal serial transfers, a packet's delay is affected by the packet's size. An additional drawback of overly large packet sizes is that a significant portion of a user's throughput is wasted when short messages are transmitted. Therefore, the system's throughput in terms of a bit rate may be quite large while the actual information rate could be small. All these points also hold true for the multiple processor architecture.

5.1.2 Throughput Estimation for the Multiple Processor System

The maximum throughput in bits/second of the multiple processor packet switch varies depending on the values of two parameters. These parameters are the packet size and the number of processors implemented. In this section, the relationship between the throughput and the number of processors is presented. In order to evaluate this packet switch, new parameters are needed. These new parameters are:

1) c_1 = Number of Processors in the Input Processor Class

2) c_2 = Number of Processors in the Packet Processor Class

3) c_3 = Number of Processors in the Packet Routing Processor Class

4) c_4 = Number of Processors in the Output Processor Class

5) C = Total Number of Processors

6) t_{P1} = Input Service Routine execution time

7) t_{P2} = Packet Sorting Routine execution time

8) t_{P3} = Packet Routing Routine execution time

9) t_{P4} = Output Routine execution time

10) F_{Pc1} = Input Processor Class throughput in packets per second

11) F_{Pc2} = Packet Sorting Processor Class throughput in packets per second

12) F_{Pc3} = Packet Routing Processor Class throughput in packets per second

13) F_{Pc4} = Output Processor Class throughput in packets per second

The maximum throughput of the switch is limited by the maximum throughput of the class of processors which has the smallest maximum throughput. The throughput of each class of processor depends on the software execution times and the number of processors assigned to each class. Therefore, the throughput for each processor class is:

$$F_{Pc_i} < (1/t_{P_i})c_i, \quad 1 \leq i \leq 4 \tag{5.8}$$

In order to use this equation in the performance evaluation of the multiple processor packet switch, the software

execution times must be known. Table 5.2 contains the software execution times for each class of processor. Various values are listed since the execution times of some routines vary depending on the current state of the system. As stated earlier, the packet switch's maximum throughput is achieved when the processors are busy 100% of the time and when no output queue list is empty. Therefore, the execution times used in this throughput estimation are:

1) The execution time of the Input Routine when data from the ELIST is available immediately.
2) The execution time of the Packet Sorting Service Routine when the Packet Routing Data Port's DAC flag is set.
3) The execution time of the Packet Routing Service Routine when it enqueues a packet under normal operating conditions without updating an OSW.
4) The execution time of the Output Service Routine when it transmits a packet under normal operating conditions.

Using the data from Table 5.2 in equation 5.8, a table listing the throughputs as a function of the number of processes is constructed. Table 5.3 contains this data compiled from the evaluation. A graph displaying the relationship between the number of processors and the upper bound on the system throughput is presented in Figure 5.1. This graph is plotted using the data contained in Table 5.3.

Normal Operation (No Memory Contention):	
Input Service Routine	13 cycles = 1.56 μ Sec
Packet Sorting Service Routine	13 cycles = 1.56 μ Sec
Packet Routing Service Routine	
(a) Enqueue Packet	9 cycles = 1.08 μ Sec
(b) Enqueue Packet and Update OSW	9 cycles = 1.08 μ Sec
Output Service Routine	
(a) Transmit Packet	19 cycles = 2.28 μ Sec
(b) Empty Queue	7 cycles = 0.84 μ Sec
Wort Cast Due to Memory Contention:	
Input Service Routine	13 cycles = 1.56 μ Sec
Packet Sorting Service Routine	13 cycles = 1.56 μ Sec
Packet Routing Service Routine	
(a) Enqueue Packet (DEFAULT)	13 cycles = 1.56 μ Sec
(b) Enque Packet and Update OSW (DEFAULT)	13 cycles = 1.56 μ Sec
(c) Enqueue Packet	11 cycles = 1.32 μ Sec
(d) Enqueue Packet and Update OSW	11 cycles = 1.32 μ Sec
Output Service Routine	
(a) Transmit Packet	23 cycles = 2.96 μ Sec
(b) Empty Queue	11 cycles = 1.32 μ Sec

Table 5.2 Software Execution Times for the Multiple Processor System

NUMBER OF PROCESSORS	THROUGHPUT FOR EACH PROCESSOR CLASS (PACKETS X 10^6 PER SECOND)			
	INPUT	PACKET SORTING	PACKET ROUTING	OUTPUT
ONE	.641	.641	.926	.439
TWO	1.28	1.28	1.85	.877
THREE	1.92	1.92	2.78	1.32
FOUR	2.56	2.56	3.70	1.75
FIVE	3.21	3.21	4.63	2.19
SIX	3.85	3.85	5.56	2.63
SEVEN	4.49	4.49	6.48	3.07
EIGHT	5.13	5.13	7.41	3.51
NINE	5.77	5.77	8.33	3.95
TEN	6.41	6.41	9.26	4.39

Table 5.3 Throughput for each Processor Class

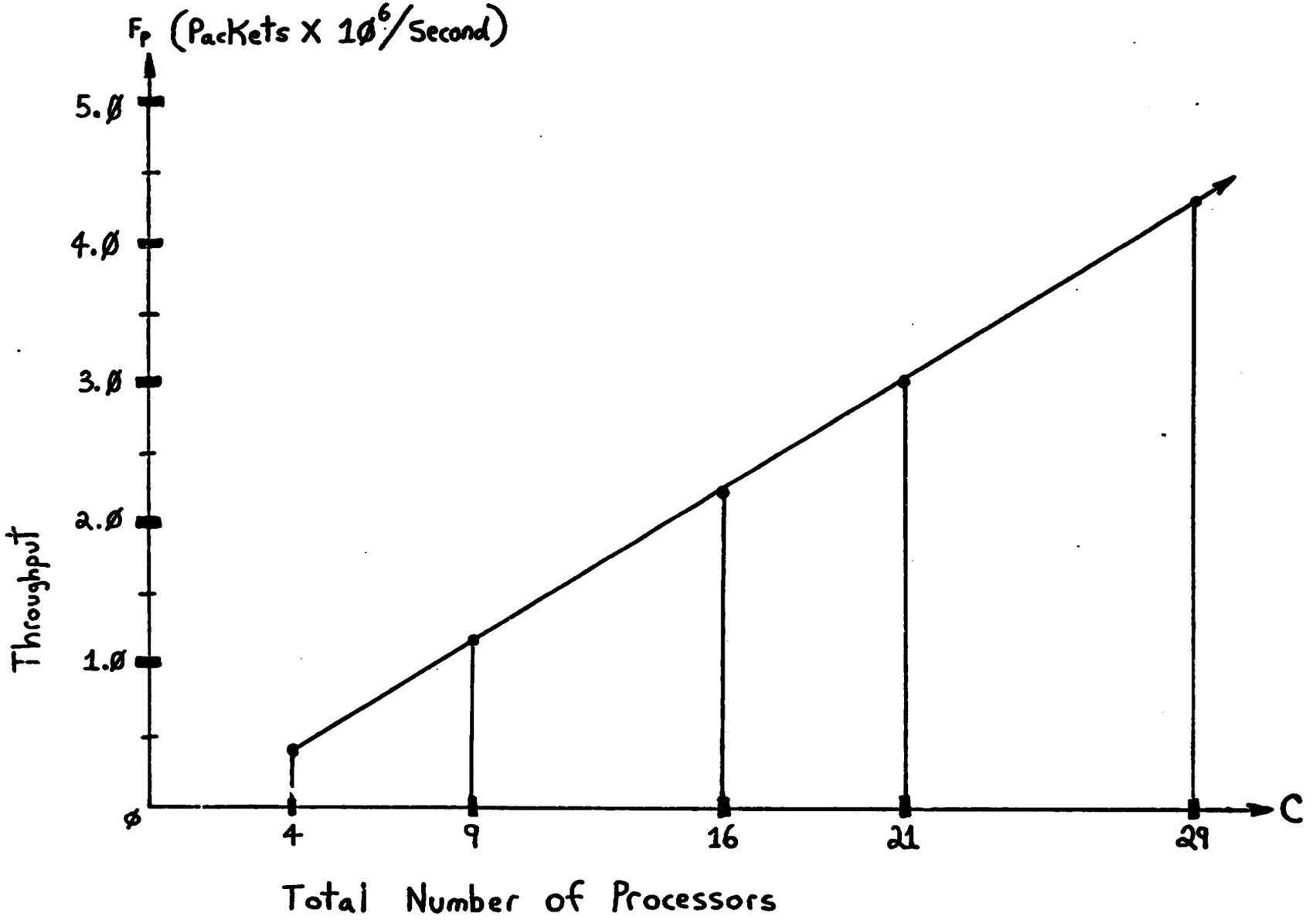


Fig. 5.1 System Throughput as a Function of the Number of Processors

A specific example is presented below to illustrate how the number of processors required for a desired throughput is determined:

Packet Length

$B = 10{,}240$ bits per packet

Desired Throughput

$F_B \leq 30 \times 10^9$ bits per second

$F_B/B = F_P \leq 3.0 \times 10^6$ packets per second

The processor assignments are determined using equation 5.8.

Number of Input Processors

$$F_{Pc1} = 3 \text{ MPS} \leq (1/t_{P1})C_1$$

$$C_1 \geq (3 \times 10^6 \text{ packets/sec})(1.56 \times 10^{-6} \text{ seconds/packet/processor})$$

$$C_1 \geq 4.68 \text{ processors.}$$

Since C_i must be an integer value, $C_1 \geq 5$ processors.

Number of Packet Sorting Processors

$$F_{Pc3} = 3 \text{ MPS} \leq (1/t_{P2})C_2$$

$$C_2 \geq (3 \times 10^6 \text{ packets/sec})(1.56 \times 10^{-6} \text{ seconds/packet/processor})$$

$C_2 \geq 4.68$ processors

$C_2 \geq 5$ processors.

Number of Packet Routing Processors

$F_{Pc3} \geq 3$ MPS $\leq (1/t_{P3})C_3$

$C_3 \geq (3\times10^6$ packets/sec$)(1.08\times10^{-6}$ seconds/packet/processor)

$C_3 \geq 3.24$ processors

$C_3 \geq 4$ processors.

Number of Output Processors

$F_{Pc4} = 3$ MPS $\leq (1/t_{P4})C_4$

$C_4 \geq (3\times10^6$ packets/sec$)(2.28\times10^{-6}$ seconds/packet/processor)

$C_4 \geq 6.84$ processors

$C_4 \geq 7$ processors.

There is an important point to note regarding the system throughput. As mentioned earlier, the system throughput depends on the packet size and the number of processors implemented. The important point of this relationship is that the number of processors that can be implemented is limited by the number of users. Each user is considered to

have one input and one output buffer. If one ground station user is allocated two sets of buffers, he is viewed as two distinct users by the switch. The number of users limits the throughput because the number of Input, Packet Routing and Output Processors can never exceed the number of users. This limitation arises since each user's workload cannot be efficiently divided among more than one processor of the same class. Therefore, the maximum attainable packet throughput for a fixed number of users is achieved when one processor from each class listed above is assigned to one user. As seen in Table 5.2, the Output function requires the longest execution time of the three classes listed above. As a result, this function limits the system's maximum attainable packet throughput as given by

$$F_P < (1/t_{P4})N \quad . \tag{5.9}$$

This equation, which expresses the relationship between the maximum throughput and the number of users, is plotted in the graph of Figure 5.2. The importance of this relationship is illustrated in the example given below.

Desired System Features:

$N = 5$ users

$B = 10{,}240$ bits per packet

$F_B = 30 \times 10^9$ bits per second

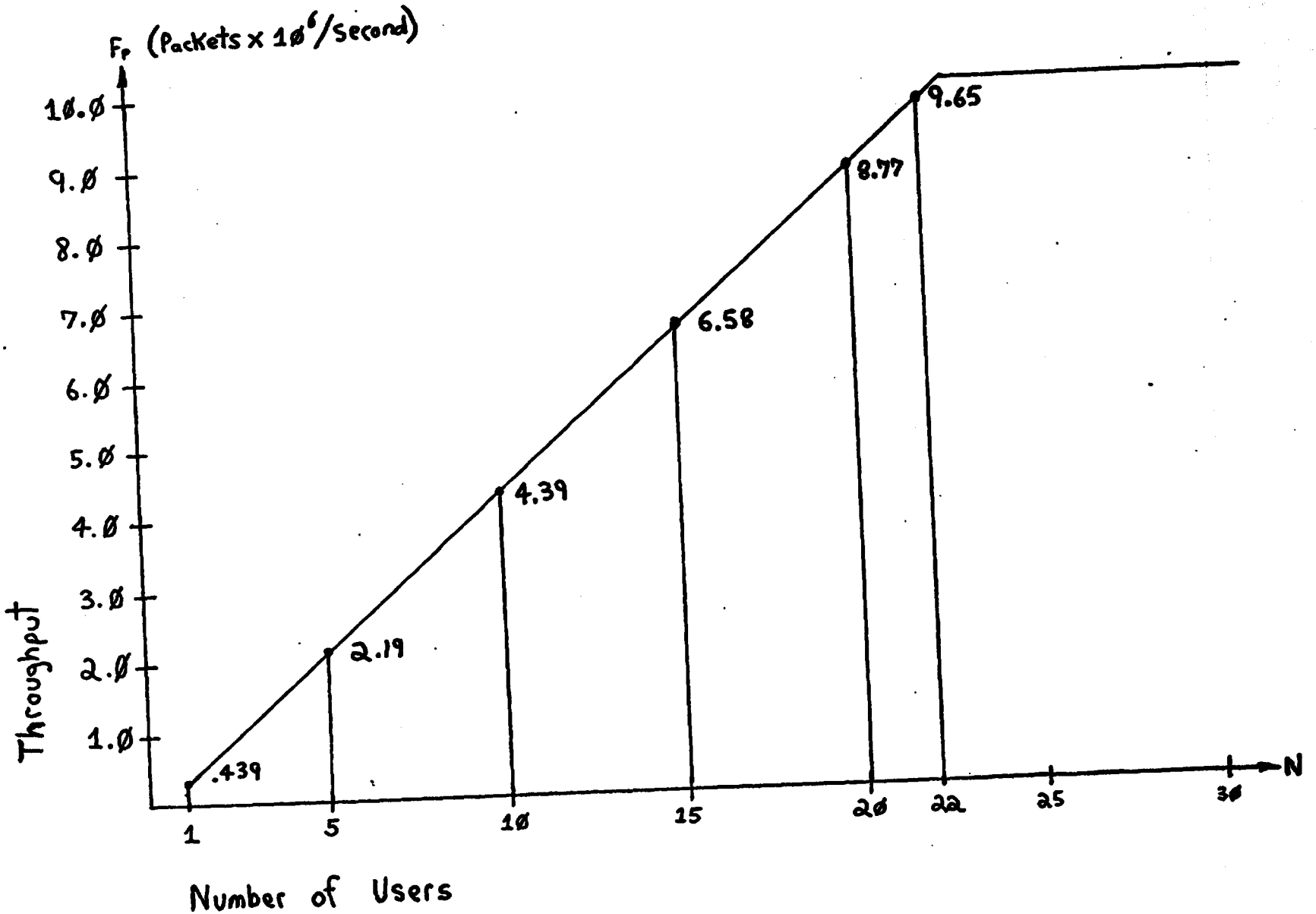


Fig. 5.2 System Throughput as a Function of the Number of Users

System Performance Evaluation Using Equation 5.9:

$$F_P < (1 \text{ packet}/2.28 \text{ microseconds}) \cdot 5$$

$$F_P < 2.19 \times 10^6 \text{ packets per second}$$

$$F_B = B \times F_P < 22.5 \times 10^9 \text{ bits per second}$$

As seen by the results above, the system performance falls short of the desired goals. The system designer has three options available:

1) Build the system and reduce each user's throughput to meet the lower performance rating.
2) Increase the packet length. This solution faces the problems described in section 5.1.1.
3) Assign the ground station users additional sets of buffers so that the packet switch serves more than five users. This sclution allows additional processors to be implemented, which will increase the system's throughput rating.

The purpose of the above example is not so much to explain how to solve performance problems as to stress the importance of the last relationship presented in equation 5.9. Without this relationship, one would determine the number of processors required by referencing Figure 5.1. This obtained value may be impossible to implement due to the user/processor limitations.

A final point regarding the maximum obtainable throughput of the multiple processor system is that Equation 5.9 has a finite upperbound which is not solely limited by the number of users. As stated earlier, service for each packet requires a read and a write operation at ELIST. Therefore, ELIST will limit the maximum packet throughput of the packet switch. Using the hardware technology currently available, ELIST is designed to provide and accept address data approximately every 100 nanoseconds. This fact limits the system maximum attainable packet throughput as given by

$$F_P < (1/t_{P4})N < (1/100\times10^{-9})$$

$$F_P < (1/t_{P4})N < 10\times10^{6} \text{ packets/second} \qquad (5.10)$$

A system using a packet length of 10,240 bits will have a maximum bit rate limited by

$$F_B = B\times F_P < (10{,}240 \text{ bits/packet})\times(10\times10^{6} \text{ packets/second})$$

$$F_B < 102.4\times10^{9} \text{ bits/second.} \qquad (5.11)$$

As new and faster hardware and processor technology becomes available, the overall performance of this packet switch will improve.

5.2 Evaluation of the Processor

Implementation of the packet switch may require the construction of a customized processor chip. Therefore, a review of the characteristics of the AMD 2903 ALU will provide the system designer with an insight into the design of a processor which is better tailored for this particular application. This review begins with the available features of the AMD 2903 ALU and ends with the features not provided by this chip that would enhance processor performance.

The AMD 2903 ALU provides ample arithmetic and operations for the packet switch. In fact, the number of operations can be reduced to save hardware complexity. The only functions required are the addition operation, the logical AND and the logical OR. The on-chip register file is ideal for holding scratchpad variables. In both multiprocessor designs, the full capacity of this file is never used. Therefore, this component could be reduced in size without degrading system performance. The single Q Register, which provides a work area for some operations, was quite adequate. The provided ZERO flag went unused and could be eliminated from the custom designed processor.

There are several features the AMD 2903 ALU architecture does not support. These features would make the processor better suited for this particular application. They are:

1) Internal tristate control of the DB Direct Data Input Bus. This bus is not currently tristate because this bus is bidirectional. This allows data to enter the

ALU from external hardware as well as allowing data from the register file to be sent directly to external hardware. Since direct transmission of data from the register file to external hardware is not required, this bus could be tristated internally to save external hardware. A possible alternative would be to increase the size of the internal select MUX. In this scheme, the DB input bus would no longer need to share the internal data bus with the register file.

2) Additional Direct Data Inputs. These inputs save execution cycles since the processor does not need to generate a device's address before a read operation can be performed. These inputs can be used whenever the processor is required to access a single unique system device. Since the Data Path Busy Status Ports are unique system devices, this feature would reduce the software execution times for the Input and Output Processors in both architectures. This scheme may require larger internal Select MUXs and more select control signals. However, there does exist one way to increase the number of direct data inputs without increasing the Select MUX size or the number of control lines. As mentioned earlier, only a small portion of the register file is used. In fact, the A-Register File is never used. Therefore, this component could be removed and its input to the Select MUX could be replaced with a direct data input. This particular

feature would increase system throughput directly and should be considered an important design criterion.

3) Internal data bus latches. This feature would provide for the stabilization of ALU data inputs without the use of external latches.

All these features are recommended for any processor custom designed for the packet switches.

5.3 Packet Losses

If the throughput rating of the packet switch is exceeded, packets will be lost even when there are no hardware or software failures in the system. However, an important point to make concerning these packet losses is that the system will always recover at some point in time. In both architectures, packets can be lost due to overflow in three components. Overflow can take place in an additional component of the multiple processor system. The components which are susceptible to overflow are:

1) The Input Buffers
2) The Output Queue Lists
3) ELIST
4) The Packet Routing Data Ports' queues.

Even with double buffering, an input buffer will overflow if its user exceeds his allotted channel capacity. The oldest of the two packets residing in the input buffer will be lost as the new packet is shifted into the buffer.

If any output queue list becomes full, the packet switch will encounter serious problems. When a queue list becomes full, the two index pointers will be equal in value. This is the same situation for an empty list. When the two pointers are equal, the Output Processor assumes the list is empty and does not access the list until new data is placed into the queue list. Therefore, the list remains full until new data is placed into the list, overwriting valid data. Only after overflow has occurred can the Output Processor access the list. Two serious problems arise from this overflow condition. The first problem is that once overflow takes place in the queue, no less than the entire list of original data will be lost. The second problem is a result of the first problem. As stated earlier, the data stored in the Output Queue Lists are the array addresses of routed packets. Therefore, if these addresses are lost, the routed packets will never be transmitted and they will remain in the Shift Register Array indefinitely. Since they are never transmitted, their array addresses will never be returned to ELIST. This fact could cause ELIST to become empty. An empty ELIST and the associated problems of this situation are discussed next.

If ELIST becomes empty and a new packet arrives at the input, the oldest packet in the shift register array will be lost as the new packet is stored in its place. Packets will continue to be lost until the Output Processors return enough array addresses to ensure that the next shift register address fetched by an Input Processor is valid data. ELIST will

become empty when the system users exceed the packet switch's throughput rating.

In the multiple processor design, if a Packet Destination Data list becomes empty, the system will face problems similar to those caused by a full Output Queue list. This is due to the fact that both lists share the same data structure. Again, packets will be trapped in the Shift Register Array because the data lost during overflow is needed for routing. If a packet is never routed, it can never leave the array. There is no way to re-sort these packets, which means the lost routing information can never be recovered. As with a full Output Queue list, the entire list of original data will be overwritten before the system can recover.

Packet losses reduce the actual throughput of a system since users must retransmit all packets lost in transmission. Since a large and effective throughput is the primary goal of this work, care must be taken to ensure against packet losses. The system designer must research the queuing problems of the switch before deciding on the size of the Shift Register Array and all the various queue lists. If the packet switch is built with an insufficient amount of array locations and/or queue lengths for its throughput rating, packet losses will be inevitable. In addition, part of the responsibility of ensuring against packet losses belongs to the users themselves. They must not exceed the channel capacities assigned to them.

5.4 Fault Detection and Fault Tolerance

Since the packet switches presented in this work are part of a proposed communication satellite network, fault detection and fault tolerance are desirable features. Once the satellite is placed into orbit, maintenance and repair work will be quite expensive or impossible. Therefore, if the packet switch could handle its own maintenance problems, the useful life of the satellite will be extended.

The failure of some components will cause an entire channel to fail. An example of such a component is an input buffer. If an input buffer fails, the channel it serves will also fail. Some component failures will cause intermittent packet losses. An example of this type of failure would occur if one location in the Shift Register Array failed. Only the packets stored in this location would be lost or corrupted. Both of these types of failures will degrade system performance but the packet switch can still operate. However, there are certain component failures which will cause the entire packet switch to fail. These components should be either fault tolerant through the use of redundant circuitry or self-diagnostic. The self-diagnostic components should be able to hand over their tasks to a spare component upon detection of a fault. The components which fall into this category for the three processor system are:

1) The Input Processor
2) The Routing Processor
3) The Output Processor

4) All the polling circuits

5) Both Data Path Busy Status Ports

6) ELIST

The components which can cause a channel loss in the three processor design due to a failure are:

1) Input Buffers

2) Output Queue Lists

3) Output Stat[illegible]ord[illegible]

4) Output [illegible]

The components which can cause intermittent packet losses in the three processor design due to a failure are:

1) Data paths in the Input Switching Network

2) Shift Register Array locations

3) Data paths in the Output Switching Network

In the multiple processor design, the only system component that may cause the entire packet switch to fail, should it fail, is the ELIST. Single or multiple channel failures could result if one of the following fails:

1) Input Buffers

2) Input Polling Circuits

3) Input Processors

4) Data Path Busy Status Ports

5) Packet Destination Data Ports

6) Packet Routing Processors

4) All the polling circuits

5) Both Data Path Busy Status Ports

6) ELIST

The components which can cause a channel loss in the three processor design due to a failure are:

1) Input Buffers

2) Output Queue List

3) Output St

4) Output

The components which can cause intermittent packet losses in the three processor design due to a failure are:

1) Data paths in the Input Switching Network

2) Shift Register Array locations

3) Data paths in the Output Switching Network

In the multiple processor design, the only system component that may cause the entire packet switch to fail, should it fail, is the ELIST. Single or multiple channel failures could result if one of the following fails:

1) Input Buffers

2) Input Polling Circuits

3) Input Processors

4) Data Path Busy Status Ports

5) Packet Destination Data Ports

6) Packet Routing Processors

7) Output Queue Lists

8) Output Processors

9) Output Status Words

10) Output Polling Circuits

11) Output Buffers

As noted above, if the Data Path Busy Status Port of an Input or Output Switching Network fails, the loss of some channels will occur as a result. However, if only a single Input (Output) Switching Network is used by the switch (as in the case of the three processor system), a status port failure will result in the failure of the entire packet switch. Thus, system reliability and elimination of resource contention is achieved with multiple Switching Networks.

The components which can cause packet losses in the multiple processor design due to a failure are:

1) Data paths in the Input Switching Network

2) Shift Register Array locations

3) Shift Register Polling Circuits

4) Packet Sorting Processors

5) Data paths in the Output Switching Network

Now that the impact of each component failure is identified, the system designer can decide what level of fault detection and fault tolerance is needed for each component.

6.0 QUEUE THEORETIC MODELLING FOR CALCULATION OF THE AVERAGE RESPONSE TIMES AND THE AVERAGE QUEUE SIZES

6.1 Introduction

In this section queue theoretic analysis and evaluation of the proposed designs are presented. Analytical relationships between the average response times and the design parameters of the switch are obtained. These expressions are to be used to evaluate the performance of the three designs of the switch for various values of these parameters. Also, the average queue sizes in the shift register array are obtained. This queue size gives an idea as to the required size of these shift register arrays in the various designs.

6.2 Design Parameters of the Switch

The average response time of the switch and the average size of the shift register array depends on a number of parameters. The more important of these are:

1) ϕ = clock cycle time of the microprocessor - This speed determines the time taken by the processor to serve a packet at the various stages of its service.
2) t_{p1} = duration of the input interrupt service routine.
3) t_{p2} = duration of the output buffer interrupt service routine for packets.
4) t_{p3} = duration of the routing service routine.
5) t_{p4} = duration of the sorting service routine.
6) R = bit rate/user.
7) N = number of input lines connected to the switch.
8) B = number of bits/packet.

9) γ_i: destination function - this function determines the fraction of the total number of arriving packets going to individual output lines.

10) S_i = output line speed - this speed determines the time required to transmit a packet to a particular destination. Different lines may have different speeds.

11) F_p = system packet rate in packets/sec.

12) F_B = system throughput in bits/sec.

13) M = number of output lines.

14) K = number of packet size storage locations in the shift register array.

15) τ_i = time taken for unsuccessful polling of one line at the i-th queue.

16) λ = overall average arrival rate (packets/sec.).

17) T = time needed to shift one bit internally.

18) N_j, j=1,2,3,4 = number of processors at the input, output, routing and sorting service points respectively.

6.3 The Single Processor Design

6.3.1 Introduction

It appears from the proposed single processor architecture and operation of the switch that queues build up in the switch as shown in Figure 6.1. In this queueing model packets queue for service by the processor in three places. Firstly, the arriving packets queue for inputting into the shift register array. Secondly, these packets await

the routing service which includes header analysis, error analysis, generation of ACK's and NACK's, and separating the packets into software queues. Finally, these packets queue for outputting. The routing service is to be performed by the processor whereas the inputting and the outputting functions involve service by a polling circuit in addition to that by the processor. Also, the inputting function has the highest priority, the outputting function has the second highest priority and the routing service has the lowest priority. This priority assignment is assumed as the incoming packets have to be attended to upon their arrival, otherwise they will be lost. Also, the output lines, being slower than the switch itself, causes a bottleneck in the system. Hence, whenever an output line is free to transmit messages, it should be serviced as quickly as possible. Thus, the outputting process is given the second highest priority.

The packets change priority class after receiving service and the whole system can be modelled as a single server (the processor) serving customers of three levels of priority as shown in Figure 6.2. The packets of various priorities queue separately for service. The average time spent by a packet in the switch (average response time) is the sum of the waiting times and the service times at the three queues. Next, expressions are derived for the average waiting times, the overall average response time, and the average queue sizes at the various queues.

6.3.2 Parameters of the Input Queue (highest priority)

(a) The Arrival Process

The total arrival at the input queue is the sum of the arrivals on all the input lines. It is assumed that the arrival on the i-th input line is Poisson with average rate λ_{1i}. Then the overall arrival at the input queue is Poisson with arrival rate

$$\lambda_1 = \sum_{i=1}^{N} \lambda_{1i} \triangleq \lambda \tag{6.1}$$

(b) Service Time

The service time at this queue consists of polling time to locate the packet, transfer setting up time and the actual transfer time. However, the processor is free to service other lines as soon as a transfer is set up and also there are sufficient number of transfer paths available so that the actual process of transfer of any packet does not cause any delay in servicing any other packets. Thus, for the purpose of calculating the average waiting time for packets in this queue, we consider the service time

$$\begin{aligned} T_1 &= \text{polling time} + \text{setting-up time} \\ &= t_1 + t_{p1} \end{aligned} \tag{6.2}$$

where t_{p1} is a constant.

We need the mean and the second moment of T_1 and, hence, those of t_1. If there are N input lines, polled equally,

then a particular packet may be polled immediately or it may have to wait until N-1 other lines are polled and the probability of staring the scan at any one particular line is $\frac{1}{N}$. Thus, the average number of lines polled before the particular one is polled is

$$\sum_{i=0}^{N-1} \frac{i}{N} = \frac{N-1}{2} \tag{6.3}$$

and the average time spent for unsuccessful polling is $(\frac{N-1}{2})\tau_1$ where τ_1 is the time taken for unsuccessful poll of one line. Also, the mean square value of the polling time is

$$\sum_{i=0}^{N-1} \frac{(i\tau_1)^2}{N} = \frac{(N-1)(2N-1)\tau_1^2}{6} \tag{6.4}$$

Hence, the average service time

$$E[T_1] = \frac{N-1}{2}\tau_1 + t_{pl} \tag{6.5}$$

and the mean square value of T_1 is

$$E[T_1^2] = \frac{(N-1)(2N-1)\tau_1^2}{6} + t_{pl}^2 \tag{6.6}$$

(c) Utilization Factor

$$\rho_1 = \lambda_1 \cdot E[T_1] = (\sum_{i=1}^{N} \lambda_{1i})[\frac{N-1}{2}\tau_1 + t_{pl}] \tag{6.7}$$

6.3.3 Parameters of the Output Queue (Second highest priority)

(a) The Arrival Process

This queue, in fact, consists of M separate queues, one for each output line. A packet from this queue is serviced when the corresponding output buffer is empty. An empty output buffer produces an interrupt that is recognized by a polling circuit, and is serviced by the processor if there is a packet to be transmitted in the corresponding output queue. If there is no packet in the corresponding output queue, then this interrupt is disabled until a packet is available.

The time spent in this queue is calculated in two stages. Firstly, the time spent in waiting for and being serviced by the processor and secondly, the time spent in transferring and transmission of packets from the shift register array to the output lines.

All the packets in all the output queues and the packets in the input queue affect the time spent by any packet waiting in any of the output queues for the processor. However, the time for transferring and transmission of a packet depends only on the speed of the corresponding output line because the processor can attend to other packets as soon as a transaction has been set up. Hence, to find the waiting time, we shall consider all transactions in the output queues to form one queue. It should be noted that it is the interrupts by the output buffers that are serviced by the processor. However, the interrupts are serviced only if there is a transaction available for transfer in the corresponding output queue. Thus, we are

assuming that the arrival of the interrupts follows the same distribution as the arrival of the packets to the output queues. This arrival process is, in fact, nonpoisson. However, we shall assume it to be Poisson with the understanding that the results obtained are the worst case ones. The arrival rate is $\lambda_2 = \lambda_1 = \lambda$.

(b) Service Time

The relevant service time for calculating the waiting time is

$$T_2 = \text{polling time} + \text{setting time}$$
$$= t_2 + t_{p2} \tag{6.8}$$

where t_{p2} is a constant.

The transfer time is not included here because it does not affect the waiting time for service by the processor. Following the arguments given in connection with the polling time for the input queue, it can be shown that the average service time

$$E[T_2] = \frac{M-1}{2}\tau_2 + t_{p2} \tag{6.9}$$

and

$$E[T_2^2] = \frac{(M-1)(2M-1)\tau_2^2}{6} + t_{p2}^2 \tag{6.10}$$

(c) The Utilization Factor

The utilization factor connected with the service by the processor, for this queue is

$$\rho_2 = \lambda_2 \cdot E[T_2] \tag{6.11}$$

6.3.4 Parameters of the Queue for Routing Service (third highest priority)

(a) The Arrival Process

The arrival process is not exactly Poisson. However, for the purpose of this analysis, it is assumed to be Poisson with the understanding that the results obtained are the worst case ones. The arrival rate is $\lambda_3 = \lambda_1 = \lambda$.

(b) Service Time

The service time T_3 = Polling Time + Processing Time

$$= t_3 + t_{p3} \tag{6.12}$$

where t_{p3} is a constant. Following the arguments given in connection with the input queue, it can be shown that

$$E[T_3] = \frac{K-1}{2} \tau_3 + t_{p3} \tag{6.13}$$

and

$$E[T_3^2] = \frac{(K-1)(2K-1)}{6} \tau_3^2 + t_{p3}^2 \tag{6.14}$$

where K is the number of storage locations (in packets) in the shift register array and τ_3 is the time spent in unsuccessful polling of a storage location.

(c) Utilization Factor

The utilization factor for this queue is

$$\rho_3 = \lambda_3 \cdot E[T_3] \tag{6.15}$$

6.3.5 Expression for the Average Response Time

Equations derived in the previous section are now used to obtain expressions for the response time of the switch.

The queues use random dispatching (polling) and pre-emptive queueing disciplines and do not give preference to packets with shorter service times. As this dispatching discipline is independent of service time, the mean waiting times are the same as those for Head-of-Line service discipline. However, we take the polling function into account by adding the average time due to unsuccessful polling to the actual processing time by the processor. Then the average waiting time at the queue with the j-th priority is [7,8]

$$E[t_{w_j}] = \frac{1}{(1 - \sum_{i=1}^{j-1} \rho_i)} \left[E(T_j)(\sum_{i=1}^{j-1} \rho_i) + \frac{\sum_{i=1}^{j} \lambda_i E[T_i^2]}{2[1 - \sum_{i=1}^{j} \rho_i]} \right] \qquad (6.16)$$

$$j = 1,2,3.$$

The average of the total time spent by a packet in the input queue (highest priority) (time spent in waiting, being serviced by the processor and being transferred to the shift register array from the input buffers) is

$$E(t_{q1}) = E[t_{w1}] + E[T_1] + E[T_{t1}] \qquad (6.17)$$

where T_{t1} is the transfer time at this queue 1. The average of the total time spent by a packet in the output queue (second highest priority) is calculated in the following way:

(a) Arrival Process

This queue consists of M separate queues and the waiting time is different in the different queues as the waiting time in a queue depends on the arrival process and the speed of the corresponding output line. The arrival to each of the queues is assumed to be Poisson. However, the arrival rate may be different for different queues. The arrival rate to the i-th component queue of this second priority queue is

$$\lambda_{2i} = \gamma_i \lambda_2 = \gamma_i \lambda \tag{6.18}$$

where γ_i is specified by the destination function such that γ_i of the total arrivals at this second priority output queue go to its i-th component queue.

$$\gamma_i \leq 1 \quad ; \quad \sum_{i=1}^{M} \gamma_i = 1 \tag{6.19}$$

Hence, the average service time at the i-th component queue is

$$\begin{aligned} E[T_{2i}] &= E[t_{\omega 2}] + t_{p2} + E[T_{t_{2i}}] \\ &= E[t_{\omega 2}] + t_{p2} + T_{t_{2i}} \end{aligned} \tag{6.20}$$

where $T_{t_{2i}}$ = transfer time, is a constant and t_{p2} the setting up time, is also a constant. $E[T_{t_{2i}}]$ = average transfer time from the shift register array to the output buffer + average transmission time over the i-th output line

$$\cong \text{average transmission time} = \frac{B}{S_i} \tag{6.21}$$

where S_i is the transmission speed in bits/sec of the i-th output line. The utilization factor at the i-th component queue is

$$\rho_{2i} = \lambda_{2i} \cdot E[T_{2i}] \ . \tag{6.22}$$

Also

$$E[T_{2i}^2] \cong \frac{B^2}{S_i^2} + t_{p2}^2 \tag{6.23}$$

neglecting the cross multiplication terms and $E(t_{w2}^2)$ as small. Then, the average time spent in waiting at the i-th component queue of the second priority queue is

$$E[t_{w_{2i}}] = \frac{\lambda_{2i} \cdot E[T_{2i}^2]}{2(1 - \rho_{2i})} \tag{6.24}$$

Thus, the average total time spent in the i-th component queue of the second priority queue is

$$E[t_{q_{2i}}] = E[T_{2i}] + \frac{\lambda_{2i} \cdot E[T_{2i}^2]}{2(1 - \rho_{2i})} \tag{6.25}$$

The overall average time spent in waiting and in service at the second priority queue is

$$E[t_{q2}] = \sum_{i=1}^{M} \frac{\lambda_{2i} \cdot E[t_{q_{2i}}]}{\lambda_2} = \sum_{i=1}^{M} E[t_{q_{2i}}] \tag{6.26}$$

The total average time spent by a packet in the queue for routing service (the third highest priority queue) is

$$E[t_{q3}] = E[t_{w3}] + E[T_3] \tag{6.27}$$

Thus, the overall average response time = the average total time spent by a packet in the switch

$$E[t_q] = E[t_{q1}] + E[t_{q2}] + E[t_{q3}] \tag{6.28}$$

Putting back the expressions for the relevant quantities in equation (6.28) we get the overall average response time

$$\begin{aligned} E[t_q] &= E[t_{q1}] + E[t_{q2}] + E[t_{q3}] \\ &= E[t_{w1}] + t_{p1} + E[T_{t1}] + \sum_{i=1}^{M} \gamma_i \left[E[t_{w2}] + t_{p2} \right. \\ &\quad \left. + \frac{B}{S_i} + \frac{\gamma_i \left(\sum_{j=1}^{N} \lambda_{1j} \right) \left(t_{p2}^2 + \frac{B^2}{S_i^2} \right)}{2\left(1 - \gamma_i \lambda \left(E(t_{w2}) + t_{p2} + \frac{B}{S_i} \right)\right)} \right] \\ &\quad + E[t_{w3}] + t_{p3} + \frac{K-1}{2} \tau_3 \end{aligned} \tag{6.29}$$

neglecting $E[t_{w2}^2]$ compared to $t_{p2}^2 + \frac{B^2}{S_i^2}$, where $E(t_{wj})$ $j=1,2,3$ are given by equation (6.16).

Equations (6.16) and (6.29) show the relationship of the average response time for the packets to the various design parameters of the switch, namely, the total arrival rate λ, the number of input lines N, the size of storage at the shift register array K, the number of output lines M, packet size B, transmission rates of the output lines S_i, the

processor times t_{p1}, t_{p2} and t_{p3}, the times τ_1, τ_2, τ_3 needed for unsuccessful polling of a packet at the first, second and third priority queues respectively, and γ_i, the destination function. This relationship can be used to study the effect of variation in any of these parameters on the average response time. In this respect, it is useful to draw graphs showing the variation in the average response time as some or all of these parameters are varied. Graphs of this type are presented in Figures 6.3 - 6.22. Further explanation of these graphs is presented in section 6.3.7.

6.3.6 The Average Queue Sizes

For this pre-emptive resume queue one can also obtain average queue sizes. The average number of packets waiting in the j-th queue is [7,8]

$$E[W_j] = \frac{1}{(1-\sum_{i=1}^{j-1} \rho_i)} \left[\rho_j \sum_{i=1}^{j-1} \rho_i + \frac{\lambda_j \sum_{i=1}^{j} \lambda_i E[T_i^2]}{2[1 - \sum_{i=1}^{j} \rho_i]} \right] \quad (6.30)$$

where $\lambda_1 = \lambda_2 = \lambda_3 = \sum_{i=1}^{N} = \lambda_{1i} = \lambda$, ρ_1, ρ_2, and ρ_3 are given by equations (6.7), (6.11) and (6.15) respectively, and $E[T_1^2]$, $E[T_2^2]$ and $E[T_3^2]$ are given by equations (6.6), (6.10) and (6.14) respectively. We are specifically interested in the queue size in the shift register array. This shift register array stores the packets that are waiting for the output function and the routing function. Hence, the required average queue size is $E[W_2] + E[W_3]$. A number of graphs showing the variation in

$E(W_j)$, j=1,2,3 have been obtained from equation (6.30). These graphs are shown in Figure 6.23 - 6.29. Further explanation of these graphs is presented in section 6.3.7. These graphs show the average queue sizes. However, we may be interested in finding queue size necessary for given utilization factor and probability of overflow. These results can be used to obtain an approximate answer to this question. If the utilization factor is about .6 and the probability of overflow is 10^{-3}, then the required buffer size is approximately ten times the average buffer occupancy. For smaller utilization factors, the required buffer size is further less [9].

6.3.7 Interpretation of the Graphs Showing the Effect of Various Design Parameters on the Performance of the Proposed Packet Switch

A number of graphs showing the effect of the various design parameters on the average waiting times and the average queue sizes at the three queues and the overall average response time are presented in Figures 6.3 through 6.29.

(a) The Average Waiting Times at the Three Queues

Effect of λ, N, M, ϕ and K on the average waiting times at the three queues are shown in Figures 6.3 through 6.13.

Average waiting time at queue 1 vs. λ, N, t_1 and t_{p1}.

Figure 6.3 shows the effect of ρ_1, the utilization factor on $E(t_{w1})$, the average waiting time at queue 1. $E(t_{w1})$ increases as ρ_1 increases and becomes very large as ρ_1 approaches 1. The effect of λ, N and t_{p1} on $E(t_{w1})$ can also be obtained from this graph by calculating the corresponding ρ_1 using equations (6.1) through (6.7) and using this value of ρ_1 in Figure 6.3.

Average waiting time at queue 2 vs. λ, N, t_1, t_2, t_{p1}, t_{p2} and M.

The effect of ρ_2, the utilization factor on $E(t_{w2})$, the average waiting time at queue 2 is shown in Figure 6.4. Because the packets at the input queue (queue #1) has priority over those at the output queue (queue #2), the $E(t_{w2})$ depends on both ρ_1 and ρ_2. The family of graphs in Figure 6.4 show the effect of ρ_2 on $E(t_{w2})$ for a number of values of ρ_1. It should be noted that ρ_1 has a dominant effect on $E(t_{w2})$ and for values of ρ_1 close to 1, $E(t_{w2})$ increases rapidly. This indicates that when the input queue is heavily loaded, the processor does not have much time for the second queue. It is also observed from equations (6.8) through (6.11) that ρ_2 is related to the number of input lines N, the arrival rate λ, the polling time t_2, the processor setting up time t_{p2} and the number of output lines M. Hence, the effect of any of these parameters on $E(t_{w2})$ can be obtained from Figure 6.4 by using the corresponding values of ρ_2 and ρ_1. It can be seen from Equation (6.16) that $E(t_{w2})$ contains a term $\frac{1}{1-\rho_1-\rho_2}$. Hence, if $\rho_1 + \rho_2$ approaches 1, then $E(t_{w2})$ increases rapidly. Also, if $\rho_1 + \rho_2 > 1$, then $E(t_{w2})$ may become negative. Thus, to have a reasonable value of $E(t_{w2})$, $\rho_1 + \rho_2$ should be less than unity.

Average waiting time at queue 3 vs. λ, N, t_1, t_2, t_{p2}, K and M.

Figures 6.5 through 6.10 present the effect of ρ_3, the utilization factor on $E(t_{w3})$, the average waiting time at queue 3 for a number of values of ρ_1, ρ_2 and K, the number of packet-size storage units in the shift register array. Figures 6.5 through 6.7 show the effect of K on $E(t_{w3})$ for same values of

ρ_1, ρ_2 and ρ_3. It is seen from these graphs that for any given values of ρ_1, ρ_2 and ρ_3, (e.g., ρ_1 = .156, ρ_2 = .18 and ρ_3 = .343), $E(t_{w3})$ is smaller for K = 10 than for both K = 5 and K = 20. This indicates that for a given data arrival rate and processor speed, there is an optimum value of K that produces minimum $E(t_{w3})$. For values of K below this optimum value $E(t_{w3})$ increases as there may not be sufficient storage space available. Hence, the processor cannot immediately set up a transfer from the input buffer to the shift register array and thus the processor has to spend more than usual time for servicing each incoming input packet which, in turn, increases the delay in servicing the shift register array. This points to a possible tie-up situation and, hence, sufficient storage should be provided to avoid this breakdown of the process. On the other hand, as K increases, $E(t_{w3})$ increases simply because more time is spent in polling these storage units.

Figures 6.7 through 6.10 show the effect of ρ_1 on $E(t_{w3})$ for given values of ρ_2, ρ_3 and K. These figures show that as ρ_1 increases (with the same values of ρ_2, ρ_3 and K), $E(t_{w3})$ increases very rapidly indicating a dominating effect of ρ_1 on $E(t_{w3})$. This is because if the input queue is utilized heavily, then the processor does not get time to serve the second and the third queues giving rise to higher delay at these latter queues.

It should be pointed out that $E(t_{w3})$ involves a term $\frac{1}{(1-\rho_1-\rho_2-\rho_3)}$, (cf. equation (6.16)), and, hence, as $\rho_1 + \rho_2 + \rho_3$ approaches unity, $E(t_{w3})$ increases rapidly and if $\rho_1 + \rho_2 + \rho_3 > 1$, then $E(t_{w3})$ may be negative. Hence, $\rho_1 + \rho_2 + \rho_3$ should be kept less than unity.

Average waiting times vs. clock cycle time of processor.

One of the objectives of this work has been to find out the effect of the speed of the microprocessor on the performance of the packet switch. For this purpose, graphs have been obtained showing the effect of ϕ, the processor clock cycle time on $E(t_{w1})$, $E(t_{w2})$ and $E(t_{w3})$ as shown in Figures 6.11, 6.12 and 6.13 respectively.

Seven values of the clock cycle time, namely 0, 25 ns, 50 ns, 75 ns, 100 ns, 125 ns and 150 ns have been considered. It is seen from these graphs that the clock cycle time has a prominent effect on the waiting times. An arrival rate of $\lambda = 8 \times 10^4$ packets/sec has been used in generating these graphs and the corresponding values of ρ_1, ρ_2 and ρ_3 as obtained from equations (6.7), (6.11) and (6.15) respectively are also shown on these graphs. For the AMD 2900 bit slice microprocessor used in the present design, the clock cycle time is approximately 120 ns. The corresponding values of $E(t_{w1})$, $E(t_{w2})$ and $E(t_{w3})$ are 250 nS, 1.7 µS and 11.5 µS respectively.

In the future as more powerful microprocessors (with smaller clock cycle times) become available, the corresponding waiting times at the various queues can be obtained from these graphs. Other arrival rates also can be used in obtaining similar graphs provided that the corresponding $\rho_1 + \rho_2 + \rho_3$ remains less than unity.

(b) The Overall Average Response Time

Effect of the various parameters on $E(t_q)$, the overall average response time is shown in Figures 6.14 through 6.22.

Overall average response time vs. packet size B.

Figure 6.14 shows the effect of the packet size B on the overall average response time $E(t_q)$. Four graphs each corresponding to a different set of (ρ_1, ρ_2, ρ_3) are shown. It is seen that in each case the overall average response time increases at the same moderate rate as B goes from 1000 bits to 10,000 bits. This is a very useful result. Because the throughput of the switch increases directly as B, whereas the corresponding response time increases at a much slower rate. Thus, the throughput can be increased considerably without suffering severe penalty in response time. It is to be noted that ρ_1, ρ_2 and ρ_3 do not depend on B. It is the shifting times that depend on B. Hence, the response time for a given B can be reduced by employing a faster hardware for shifting of data.

Overall average response time vs. destination function γ_i.

Figures 6.15 and 6.16 show the effect of destination functions on the overall average response time $E(t_q)$. In figure 6.15, all output lines are assumed to have equal capacities. Also, five different sets of destination functions have been used. The destination function sets 1 and 2 represent random distribution of data to the various output lines. Set 3 represents uniform distribution of data to the output lines. The fourth set is such that half of all the data go to the output line number 1. The output lines 2, 3, 4 and 5 receive only ten percent of the data each. The rest of the lines receive only two percent of the data. This is a biased destination function. The fifth set again represents a biased destination function

with the output line number 2 receiving fifty percent of the data. The capacities of all output lines are the same. It is observed from Figure 6.15 that the overall average response time is minimum for the uniform destination function. Also, for the biased destination functions, the response times are considerably higher than that for the uniform destination function case. The input arrival rate is chosen such that the utilization factor for each of the output lines is less than unity.

For Figure 6.16 the same sets of destination functions and same values of other parameters are used except that in this case the capacities of the output lines are given by $S_i = 5\lambda B\gamma_i$. Here, the capacity of each output line is proportional to the amount of data destined for it. Because of this, the response time remains constant for all the destination functions.

Overall average response time vs. output line speeds S_i.

Figures 6.17 through 6.22 show the variation of the overall average response time due to changes in the capacities of the output lines. Three types of capacity assignments are considered: uniform, proportional and square root. In the uniform capacity assignment, the capacities of all the output lines are the same ($S_i = \frac{\lambda B\alpha}{M}$). In the proportional assignment, each output line is given capacity proportional to the traffic on it ($S_i = \lambda B\gamma_i\alpha$). In the square root capacity assignment, every line is assigned minimum capacity equal to the traffic expected on this line. Additional capacities are then assigned to each line in proportion to the square root of the traffic expected on that line. Figures 6.17 through 6.19 show the

response time for uniform destination functions (γ_i = .1 for all i). With this destination function, identical response times are obtained for all three types of capacity assignments as shown in Figures 6.17 through 6.19. This is so because with this destination function all three capacity assignments result in the same capacity values for the output lines. The case when α = 1, i.e., the capacity assignment is equal to the average traffic on a line, the response time is undefined as the one or more terms in equation (6.29) may be negative. It is observed from these graphs that the response time decreases as α increases, the decrease being sharper initially and more sluggish for $\alpha > 5$. Thus, after certain values of α, increasing the line capacities may not reduce the response time correspondingly. That means a point of diminishing return sets in.

These general comments apply to Figures 6.20 through 6.22 also. However, for these cases, the destination function is a biased one and, hence, the response time does not have the exact same value for the three different capacity assignment strategies.

(c) The Effect of the Various Design Parameters on the Average Queue Sizes

The number of packets waiting at the various queues for various design parameters is shown in Figures 6.23 through 6.29.

Average queue sizes vs. λ, N, M, K, t_1, t_2, t_3, t_{p1}, t_{p2} and t_{p3}.

Figure 6.23 shows the variation in the average queue size $E(w_1)$ with ρ_1, the utilization factor at queue 1. This curve has similarity with that for $E(t_{w1})$. This follows from Little's

formula which states that the average queue size = average arrival rate x average time spent in the system. As ρ_1 approaches unity, the queue size increases rapidly. However, as the queue 1 has the highest priority, the queue size is rather small for $\rho < .9$.

Figure 6.24 shows the average queue size $E(w_2)$ as a function of ρ_2, the utilization factor at queue 2 for a number of values of ρ_1. For reasonable results $\rho_1 + \rho_2$ should be less than unity. It is also seen from this figure that ρ_1 has a dominant effect on $E(w_2)$.

Figures 6.25 through 6.29 show how $E(w_3)$, the queue size at the third queue changes with ρ_1, ρ_2, ρ_3 and K. Figures 6.25 and 6.26 show $E(w_3)$ for K = 10 and K = 50 respectively for given values of ρ_1, ρ_2 and ρ_3. It is seen that the expected queue size $E(w_3)$ goes up somewhat for K = 50 than for K = 10. This is due to the additional polling time necessary for finding the stored packets. It appears that K = 10 is reasonable for $\rho = .1$. However, it is seen from Figures 6.26 through 6.29 that $E(w_3)$ increases rather quickly as ρ_1 increases. Hence, for higher values of ρ_1, a larger value of K should be used and the corresponding queue size be determined. For the purpose of this report, K = 50 is used and the corresponding $E(w_3)$ are shown. If a higher value of ρ_1 is intended to be used, then a K larger than 50 has to be used.

6.4 The Three Processor Design

6.4.1 Introduction

It appears from the proposed three processor architecture and operation of the switch that queues build up in the switch as shown in Figure 6.30. In this queueing model, packets queue for service by the processors in three places. Firstly, the arriving packets queue for inputting into the shift register array. Secondly, these packets await the routing service which includes header analysis, error analysis, and separating the packets into software output queues. Finally, these packets queue for outputting. All the packet switch functions involve service by polling circuits in addition to processor service.

The average time spent by a packet in the switch (average response time) is the sum of the waiting times and the service times at the three queues. Next, expressions are derived for the average waiting times, the average response times, and the average queue sizes at the various queues.

6.4.2 Expressions for the Waiting Times at the Various Queues and the Overall Average Response Time

The assumptions made for the queueing model for the single processor design are also assumed here. Also, the analytical developments used in section 6.3 are valid here except that in the three processor design, each processor is performing only one function. Hence, the average waiting time at each queue depends on the corresponding utilization

factor only. Thus, the average waiting times at the routing and the output queues depend on ρ_3 and ρ_2 respectively and not on other ρ's.

Following the definitions and analytical developments similar to those for the single processor design (cf. section 6.3), it can be shown that for the three processor design, the overall average response time $E(t_q)$ is

$$E[t_q] = E[t_{q1}] + E[t_{q2}] + E[t_{q3}]$$

$$= E[t_{w1}] + t_{p1} + E[T_{t1}] + \sum_{i=1}^{M} \gamma_i \left[E[t_{w2}] + t_{p2} + \frac{\gamma_i (\sum_{j=1}^{N} \lambda_{1j})(t_{p2}^2 + \frac{B^2}{S_i^2})}{2(1 - \gamma_i \lambda (E(t_{w2}) + t_{p2} + \frac{B}{S_i}))} \right]$$

$$+ E[t_{w3}] + t_{p3} \qquad (6.31)$$

neglecting $E[t_{w2}^2]$ compared to $t_{p2}^2 + \frac{B^2}{S_1^2}$, where $E(t_{wj})$ $j=1,2,3$, the average waiting times at the j-th queue are given by [7,8]

$$E(t_{wj}) = \frac{\lambda E[T_j^2]}{2(1-\rho_j)} \quad , \quad j=1,2,3 \qquad (6.32)$$

where $E[T_j^2]$, $j=1,2,3$ and ρ_j, $j=1,2,3$ are given by equations (6.6), (6.10), (6.14) and (6.7), (6.11) and (6.15) respectively. The difference between equations (6.16) and (6.32)

should be noted. Also, the polling times τ_1, τ_2 and τ_3 are assumed to be negligible as polling at all three queues are done by hardware in this case.

Equations (6.31) and (6.32) show the relationship of the average response time for the packets to the various design parameters of the switch, namely, the total arrival rate λ, the number of input lines N, the number of output lines M, packet size B, transmission rates of the output lines S_i, the processor times t_{p1}, t_{p2} and t_{p3}, and γ_i, the destination function. This relationship can be used to study the effect of variation in any of these parameters on the average response time. In this respect, it is useful to draw graphs showing the variation in the average response time as some or all of these parameters are varied. Some graphs of this type are presented in Figures 6.31 - 6.55.

6.4.3 Expressions for the Average Queue Sizes at the Various Queues

Following the developments in section 6.3.6 for the average queue sizes for the single processor design, it can be shown that for the three processor design the average number of packets waiting at the j-th queue [7,8] is

$$E[W_j] = \rho_j + \frac{\lambda^2 E[T_j^2]}{2(1-\rho_j)} \quad , \quad j=1,2,3 \qquad (6.33)$$

where $E[T_j^2]$, j=1,2,3 and ρ_j, j=1,2,3 are given by equations (6.6), (6.10), (6.14) and (6.7), (6.11) and (6.15) respectively.

We are specifically interested in the queue size in the shift register array. This shift register array stores the packets that are waiting for the output function and the routing function. Hence, the required average queue size is $E[W_2] + E[W_3]$. A number of graphs showing the variation in $E(W_j)$, $j=1,2,3$ have been obtained from equation (6.33). These graphs are shown in Figures 6.56-6.64. Further explanation of these graphs is presented in section 6.4.4. These graphs show the average queue sizes. However, we may be interested in finding queue size necessary for given utilization factor and probability of overflow. These results can be used to obtain an approximate answer to this question. If the utilization factor is about .6 and the probability of overflow is 10^{-3}, then the required buffer size is approximately ten times the average buffer occupancy. For smaller utilization factors, the required buffer size is further less [9].

6.4.4 Interpretation of the Graphs Showing the Effect of the Various Design Parameters on the Performance of the Proposed Three Processor Packet Switch

(a) Effect of Contention on the Average Waiting Times and the Average Queue Sizes at the Various Queues

In the three processor design, the problem of contention among the processors for using common resources has been resolved as much as possible. However, possible contention over the use of the output queue lists by the routing and the output processors could not be totally removed. It appears from Table 5.1 that the durations of the routing service

routines and the output service routine increase by two cycle times each in the presence of contention over those in the absence of contention. Early on we wanted to find out the effect of contention on the average waiting times and the average queue sizes at the three queues. The graphs in Figures 6.31 through 6.41 show the effect of contention on the average waiting times and the average queue sizes. An examination and comparison of the corresponding graphs with and without contention show that the effect of contention on the average waiting times and the average queue sizes at the routing and output queues are negligible. The input queue, of course, is not affected by contention. For this evaluation, two possible situations have been considered: no contention and contention at all times. The corresponding results give the lower and upper bound on the effect of contention. Results for other degrees of contention lie in between these two limits.

(b) <u>The Average Waiting Times and the Response Times at the Three Queues</u>

Figure 6.42 shows the effect of the utilization factors ρ_1, ρ_2 and ρ_3 on the corresponding average waiting times. The average waiting times increase as the corresponding ρ increases. For values of ρ beyond .8, the waiting times become very high and these go to infinity for ρ equal to unity. Actual values of these waiting times for a given value of ρ differs due to the difference in the values of $E[T_1^2]$, $E[T_2^2]$ and $E[T_3^2]$ which happens due to the difference in the values of t_{p1}, t_{p2} and t_{p3} as noted on Figure 6.42. Figure 6.43 shows similar effects on the average response times at the three queues.

Average waiting times vs. clock cycle time of processor.

One of the objectives of this work has been to find out the effect of the speed of the microprocessors on the performance of the packet switch. For this purpose, graphs have been obtained showing the effect of ϕ, the processor clock cycle time on $E(t_{w1})$, $E(t_{w2})$ and $E(t_{w3})$ as shown in Figures 6.44, 6.45 and 6.46 respectively.

Eleven values of the clock cycle time have been considered. The corresponding values of the respective utilization factors are shown on these graphs. It is seen from these graphs that the clock cycle time has a prominent effect on the waiting times. An arrival rate of $\lambda = 8 \times 10^4$ packets/sec has been used in generating these graphs and the corresponding values of ρ_1, ρ_2 and ρ_3 as obtained from equations (6.7), (6.11) and (6.15) respectively are also shown on these graphs. For the AMD 2900 bit slice microprocessor used in the present design, the clock cycle time is approximately 120 ns. The corresponding values of $E(t_{w1})$, $E(t_{w2})$ and $E(T_{w3})$ are 80 ns, 150 ns and 166 ns respectively.

In the future, as more powerful microprocessors (with smaller clock cycle times) become available, the corresponding waiting times at the various queues can be obtained from these graphs. Other arrival rates also can be used in obtaining similar graphs provided that the corresponding ρ's remain less than unity.

(c) The Overall Average Response Time

Effect of the various parameters on $E(t_q)$, the overall average response time, is shown in Figures 6.47 through 6.55.

Overall average response time vs. packet size B

Figure 6.47 shows the effect of the packet size B on the overall average response time $E(t_q)$. Four graphs each corresponding to a different set of (ρ_1, ρ_2, ρ_3) are shown. It is seen that in each case the overall average response time increases at the same moderate rate as B goes from 1000 bits to 10,000 bits. This is a very useful result. Because the throughput of the switch increases directly as B whereas the corresponding response time increases at a much slower rate. Thus the throughput can be increased considerably without suffering severe penalty in response time. It is to be noted that ρ_1, ρ_2 and ρ_3 do not depend on B. It is the shifting times that depend on B. Hence, the response time for a given B can be reduced by employing a faster hardware for shifting of data.

Overall average response time vs. destination function γ_i.

Figures 6.48 and 6.49 show the effect of destination functions on the overall average response time $E(t_q)$. In Figure 6.48 all output lines are assumed to have equal capacities. Also, five different sets of destination functions have been used. The destination function sets 1 and 2 represent random distribution of data to the various output lines. Set 3 represents uniform distribution of data to the output lines. The fourth set is such that half of all the data go to the output line number 1. The output lines 2, 3, 4 and 5 receive only ten percent of the data each. The rest of the lines receive only two percent of the data. This is a biased destination function. The fifth set again represents a biased destination

function with the output line number 2 receiving fifty percent of the data. It is observed from Figure 6.48 that the overall average response time is minimum for the uniform destination function. Also for the biased destination functions the response time is considerably higher than that for the uniform destination function case. The input arrival rate is chosen such that the utilization factor for each of the output lines is less than unity.

For Figure 6.49 the same sets of destination functions and same values of other parameters are used except that in this case the capacities of the output lines are given by $S_i = 5\lambda B\gamma_i$. Here the capacity of each output line is proportional to the amount of data destined for it. Because of this, the response time remains constant for all the destination functions.

Overall average response time vs. output line speeds S_i

Figures 6.50 through 6.55 show the variation of the overall average response time due to changes in the capacities of the output lines. Three types of capacity assignments are considered: uniform, proportional and square root. In the uniform capacity assignment the capacities of all the output lines are the same ($S_i = \frac{\lambda B\alpha}{M}$). In the proportional assignment each output line is given capacity proportional to the traffic expected on it ($S_i = \lambda B\gamma_i\alpha$). In the square root capacity assignment every line is assigned minimum capacity equal to the traffic expected on this line. Additional capacities are then assigned to each line in proportion to the square root of the traffic expected on that line. Figures 6.50

through 6.52 show the response time for uniform destination functions (γ_i = .1 for all i). With this destination function identical response times are obtained for all three types of capacity assignments as shown in Figures 6.50 through 6.52. This is so because with this destination function all three capacity assignments result in the same capacity values for the output lines. The case when $\alpha = 1$, i.e., the capacity assignment is equal to the average traffic on a line, the response time is undefined as the one or more terms in equation 6.29 may be negative. Hence the values of response time for $2 \leq \alpha \leq 10$ are shown in these graphs. It is observed from these graphs that the response time decreases as α increases, the decrease being sharper initially and more sluggish for $\alpha > 5$. Thus after certain values of α increasing the line capacities may not reduce the response time correspondingly. That means a point of diminishing return sets in.

These general comments apply to Figures 6.53 through 6.55 also. However, for these cases the destination function is a biased one and hence the response time does not have the exact same value for the three different capacity assignment strategies.

(d) The Effect of the Various Design Parameters on the Average Queue Sizes

The number of packets waiting at the various queues for various design parameters is shown in Figures 6.56 through 6.64.

Figure 6.56 shows the variation in the average queue size $E(w_1)$ with ρ_1, the utilization factor at queue 1. This curve has similarity with that for $E(t_{w_1})$. This follows from Little's formula which states that the average queue size = average arrival rate x average time spent in the system. As ρ_1 approaches unity the queue size increases rapidly. However, the queue size is rather small for $\rho < .9$. Similar comments also apply to Figures 6.57 and 6.58 which show the variation of average queue sizes at the routing and the output queues respectively.

Figures 6.59 and 6.60 show the effect of varying M, the number of output lines, on the average queue sizes at the output queue for two proportional capacity assignments to these lines. It follows from these graphs that even with proportional capacity assignment the output queue size increases with the number of output lines. This increase is mainly due to the work involved in demultiplexing data to so many lines which may or may not be ready to receive data.

It is seen from Figure 6.61 and 6.62 that the average queue size at the output queue does not increase much with increase in the packet size. This is an encouraging result as the throughput can be increased by increasing packet size without making the corresponding storage requirements too high.

Figures 6.63 and 6.64 show that the queue size at the output queue cannot be decreased much by using faster

processors. This is mainly because at the output queue major part of the service time is due to shifting time and many packets wait for the output buffers to be available rather than for service by the processor itself. It also appears from a comparison of Figures 6.63 and 6.64 that increasing the capacities of the output lines make the queue size to go down considerably.

6.5 The Multiple Processor Design

6.5.1 Introduction

In the multiple processor architecture queues build up in the switch as shown in Figure 6.65. In this queueing model every packet queue for service by appropriate processors in four places. Firstly an incoming packet queue for service by one of the input processors for inputting into the shift register array. Secondly, this packet awaits service by one of the sorting processors that assigns it to one of the routing processors. The routing processor services it by putting it into one of the output queues. Lastly this packet is serviced by one of the output processors. Each of these services involve service by appropriate processors and polling circuits. However, the hardware polling times are negligible.

It is assumed that at every stage of the service, e.g. at the input service, the total number of packets arriving there for service are equally divided among the processors performing that function. This assumption is physically reasonable as this will ensure that all the processors are equally busy. Operation of the multiple processor design indicates that at each stage of service there are a number of single server queues in parallel.

The average time spent by a packet in the switch (average response time) is the sum of the waiting times and the service times at the four queues.

Analytical expressions are derived next for the average waiting times, the average response times and the average queue sizes at the various queues.

6.5.2 Analytical Expressions for the Waiting Times at the Various Queues and the Overall Average Response Time

Assumptions made for the queueing model for the single processor design are assumed here. Also the analytical developments used in section 6.3 are valid here except

i) there is no interdependence among the functions as each function is performed by a number of processors dedicated for this function.

ii) The packet arrival rate to each processor assigned for the j-th function is λ_j/N_j where N_j is number of processors performing this function an λ_j is the overall packet arrival rate for this service. In normal operation $\lambda_j=\lambda$ for j=1,2,3,4.

It should be noted that

j=1 → input function

j=2 → output function

j=3 → routing function

and j=4 → sorting function

Following the analytical developments similar to those for the single and three processor designs, it can be shown that

the average waiting times at the j-th queue are given by [7,8].

$$E(t_{wj}) = \frac{\lambda_j E[T_j^2]}{2(1-\rho_j)} \cong \frac{\lambda\, t_{pj}^2}{2N_j(1 - \frac{\lambda}{N_j} tpj)}; \quad j=1,2,3,4 \qquad (6.34)$$

where $E[T_j^2]$ = mean square value of the service time $\approx t_{pj}^2$ (6.35)

and $\rho_j = \lambda_j E[T_j] = \frac{\lambda}{N_j} t_{pj}$ (6.36)

(neglecting the polling times).

For the queueing analysis the following values of the service times have been used.

$$\begin{aligned} t_{p1} &= 15\ \phi \\ t_{p2} &= 19\ \phi \\ t_{p3} &= 9\ \phi \\ t_{p4} &= 13\ \phi \end{aligned} \qquad (6.37)$$

These values differ slightly from the values shown in table 5.2. The values in table 5.2 are the final refined values obtained after the queueing models have been developed using the earlier estimates of these quantities. However, the queueing results will not be much different using the values in table 5.2.

The average response times at these queues are

$$E(t_{q1}) = E(t_{w1}) + t_{p1} + T_{t1} \qquad (6.38)$$

$$= E(t_{w1}) + 15\ \phi + 2\ ns \times B$$

$$E(t_{q2}) = \sum_{i=1}^{M} \gamma_i [E(t_{w2}) + t_{p2} + \frac{B}{S_i} +$$

$$+ \frac{\gamma_i \lambda\ (\frac{B^2}{s_i 2} + t_{p2}^2)}{2(1-\gamma_i \lambda (E(t_{w2}) + t_{p2} + \frac{B}{S_i}))}\ ] \qquad (6.39)$$

$$E(t_{q3}) = E(t_{w3}) + t_{p3} = E(t_{w3}) + 9\ \phi \qquad (6.40)$$

$$E(t_{q4}) = E(t_{w4}) + t_{p4} = E(t_{w4}) + 13\ \phi \qquad (6.41)$$

$E(t_{q2})$ is obtained by following the development in section 6.3.5.

The overall average response time of the switch is

$$E(t_q) = \sum_{j=1}^{4} E(t_{qj}) \qquad (6.42)$$

where $E(t_{qj})$; j=1,2,3,4 are given by equations (6.38) through (6.41) respectively.

Equations (6.34) through (6.42 show the relationship of the average waiting and response times for the packets to the various design parameters of the switch, namely, the total arrival rate λ, the number of input lines N, the number of output lines M, packet size B, transmission rates of the output lines S_i, the processor times t_{p1}, t_{p2} and t_{p3} and t_{p4}, γ_i, the destination function and N_j the number of processors at the various queues. These relationships can be used to study the effect of variation in any of these parameters on the performance of the switch. In this respect, it is useful to draw graphs showing the variation in the average waiting and response times as some or all of these parameters are varied. Some graphs of this type are presented in Figures 6.66 - 6.79. The aim here is to see how the waiting times and response times vary as the number of processors at every service stage is varied. Hence these Figures show family of graphs with N_j as a parameter. The effect of variation of other parameter should be similar to that shown for the single and three processor designs.

6.5.3 Expressions for the Average Queue Sizes at the Various Queues

Following the developments in section 6.4.3 for the average queue sizes for the three processor design, it can be shown that for the multiple processor design the average number of packets waiting at the j-th queue [7,8] is

$$E[W_j] = \rho_j + \frac{\lambda_j^2 E[T_j^2]}{2(1-\rho_j)} = \frac{\lambda}{N_j} t_{pj} + \frac{(\frac{\lambda}{N_j})^2 t_{pj}^2}{2(1 - \frac{\lambda}{N_j} t_{pj})} ;$$

$$j = 1,2,3,4. \qquad (6.43)$$

We are specifically interested in the queue size in the shift register array. This shift register array stores the packets that are waiting for the output, sorting and the routing functions. Hence, the required average queue size is $E[W_2] + E[W_3] + E(W_4)$. A number of graphs showing the variation in $E(W_j)$, j=1,2,3,4 have been obtained from equation (6.43). These graphs are shown in Figures 6.80 - 6.83. Further explanation of these graphs is presented in section 6.5.4. These graphs show the average queue sizes. However, we may be interested in finding queue size necessary for given utilization factor and probability of overflow. These results can be used to obtain an approximate answer to this question. If the utilization factor is about .6 and the probability of overflow is 10^{-3}, then the required buffer size is approximately ten times the average buffer occupancy. For smaller utilization factors, the required buffer size is further less [9].

6.5.4 Interpretation of the Graphs Showing the Effect of the Various Design Parameters on the Performance of the Proposed Multiple Processor Packet Switch

Major aim of the analysis is to see how the average waiting times at the various queues vary for given overall arrival rate as the number of processors at these queues are varied. Figures 6.66 - 6.68 show the effect of varying the number input processors on the average waiting time at the input queue. These graphs also show the effect on the average waiting time of varying the overall packet arrival rate for a given number of input processors. These three figures differ in the maximum value of λ, the packet arrival rate that is allowed. Maximum packet arrival rates of $2x10^6$, $2x10^7$ and $5x10^7$ packets/sec have been used in Figures 6.66 - 6.68 respectively. The rationale for using these three maximum values of λ is the following: For $\lambda_{max} = 2x10^6$ one can observe clearly how the average waiting time varies for a single input processor. However, the effect is not at all clear for other higher number of input processors. The using of $\lambda_{max} = 2x10^7$ and $5x10^7$ shows the effect on average waiting time of the varying the number of input processors. For the same reason three values of λ_{max} have also been used for the sorting, the routing and the output queues.

Figures 6.69 - 6.71 show the effect of varying the overall packet arrival rate on the average waiting time at the output queue. These graphs also show the effect on the average waiting time of varying the number of output processors. Similar results are shown in Figures 6.72 - 6.74 and Figures 6.75 - 6.77 for the routing and the sorting queues respectively.

Figure 6.78 shows the effect of varying B, the packet size on the overall average response time for a fixed number of processors. The overall average response time increases slightly as B increases.

Figure 6.79 show the effect of destination functions on the overall average response time $E(t_q)$. In this figure all output lines are assumed to have equal capacities. Also five different sets of destination functions have been used. The destination function sets 1 and 2 represent random distribution of data to the various output lines. Set 3 represents uniform distribution of data to the output lines. The fourth set is such that half of all the data go to the output line number 1. The output lines 2, 3, 4 and 5 receive only ten percent of the data each. The rest of the lines receive only two percent of the data. This is a biased destination function. The fifth set again represents a biased destination function with the output line number 2 receiving fifty percent of the data. The capacities of all output lines are the same.

It is observed from Figure 6.79 that in the case of multiple processor design the $E(t_q)$ is almost constant for all the sets of destination functions. One explanation is that in the case of multiple processor design any output line with more packets destined for it may be provided with a dedicated processor. Also since the output lines are slower than the processor no large queue will build up, of course the capacity of the output lines should be high enough to absorb the packets destined for them. It is to be noted that in Figure 6.79, the ratio of maximum packet arrival rate

to the line capacity is .5 M $\lambda B/\lambda B8 = \frac{5}{8}$. Thus the channel capacity is large enough to handle the packet arrival rates for even the line with destination function of .5.

Thus it is seen that $E(t_q)$ is almost constant for all sets of destination functions.

Finally figures 6.80 - 6.83 show the effect of variation in the number of processors on the average queue sizes for a given packet arrival rate at the input, output, routing and the sorting queues respectively. These Figures also show the effect on the average queue sizes of varying the packet arrival rate for a fixed number of processors at the corresponding queues. It should be noted that the queue sizes decrease as the number of processors increase at the various queues.

6.6 Conclusions

Queue theoretic models have been developed for all the three proposed architectures. Graphs showing the average waiting times, the overall average response times and average queue sizes as functions of various design parameters have been obtained. It is observed from these graphs that in most cases the average waiting times and the average queue sizes are reasonable. The overall response times and the queue sizes are much smaller in the three processors case than in the single processor case. These quantities are further reduced in the multiple processor case, however, not proportionately.

The main incentive for using multiple processors is to increase the throughput. However, the response times and the queue sizes (the storage requirement) are also reduced in the process. Thus it seems that the multiple processor design is the one to be used.

7.0 Summary

7.1 Suggestions for Future Work

Several suggestions for future work in the area of processor-controlled packet switches are presented in [2]. An additional feature which is possible in the multiprocessor architectures is the transmission of system status data to each user. This scheme would require an additional processor which would be required to monitor the system status. This processor could monitor the status of ELIST, the Output Queue Lists, and important system hardware. If this processor discovered a hardware failure, a near empty ELIST or a nearly filled queue list it could generate a packet-length message that would inform the user of the system problems. This processor would be required to inform the Output Processor to send a system status data packet to each user. Using the received status information, user could re-route messages around nonfunctioning channels, reduce their overall throughput, or reduce their throughput to a specific user to avoid packet losses.

Any system enhancements will be paid for in terms of throughput and/or the number of required processors.

7.2 System Throughput

All three packet switch architectures are capable of handling large system throughputs as shown in the following examples.

7.2.1 Single Processor Packet Switch

$$F_p < 1.5 \times 10^5 \text{ packets/sec.}$$

Using a packet length of 10,240 bits, the maximum bit rate for the system is

$$F_B = F_p \times B < (1.5 \times 10^5) \times (10,240) \text{ bits/sec.}$$

$$F_B < 1.54 \times 10^9 \text{ bits/sec.}$$

7.2.2 Three Processor Pack Switch

$$F_p < 5.21 \times 10^5 \text{ packets/sec}$$

Using the packet length of 10,240 bits, the bit rate for this system is

$$F_B = F_p \times B < (5.21 \times 10^5) \times (10,240) \text{ bits/second}$$

$$F_B = 5.33 \times 10^9 \text{ bits/second}$$

7.2.3 Multiple Processor Packet Switch

Example System

$$F_B < 30 \times 10^9 \text{ bits/second}$$

$N = 10$ users

$B = 10,240$ bits/second

Packet Throughput Requirement for this system

$$F_p = F_B/B < (30 \times 10^9/10,240) \text{ packets/second}$$

$$F_p < 2.93 \times 10^6 \text{ packets/second}$$

Since the throughput is limited by Equation 5.9, which states the packet throughput is limited by the number of users, the value calculated above may not be obtainable for this system. An evaluation must be made.

$$F_p < 2.93\times10^6 \overset{?}{<} (1/t_{P4})N = 4.39\times10^6$$

is true, the proposed system can be built to handle the desired bit rate. By using Equation 5.8 for each class of processors, the total number of processors required for this system is determined. Twenty-one processors are needed: five Inpur Processors, five Sorting Processors, Four Routing Processors and seven Output Processors. This system using twenty-one processors will provide a bit rate of 30×10^9 bits/second. As shown above in the evaluation using Equation 5.9, this throughput is not the maximum obtainable bit rate. Thus, if additional processors were implemented, a larger throughput could be provided to the ten users.

The cost of achieving these large throughputs is paid for in terms of the number of processors required, the width of the Microprogram ROM and the special purpose hardware and software required to deal with contention problems. The major trade-off in both designs is that a reduction in the software executions is paid for in hardware complexity. Two prime examples of this type of trade-off are the use of hardware pollers and the large number of microprogram control bits, which enable the execution of concurrent tasks.

7.3 Queue Theoretic Results

Queue theoretic models have been developed for all the three proposed architectures. Graphs showing the average waiting times, the overall average response times and average queue sizes as functions of various design parameters have been obtained. It is observed from these graphs that in most cases the average waiting times and the average queue sizes are reasonable. The overall response times and the queue sizes are much smaller in the three processors case than in the single processor case. These quantities are further reduced in the multiple processor case, however, not proportionately.

The main incentive for using multiple processors is to increase the throughput. However, the response times and the queue sizes (the storage requirement) are also reduced in the process. Thus it seems that the multiple processor design is the one to be used.

The major contribution of this work to the area of digital communications is the design of efficient multiprocessor packet switches which can provide large throughputs, special functions and flexibility not available in non-programmable systems. The overall performance of these packet switches will improve as faster hardware and processors become available.

REFERENCES

1. Roberts, Lawrence G., "The Evolution of Packet Switching," Proceedings of the IEEE, Vol. 66, No. 11, pp. 1307-1312, November 1978.

2. "Design of a Microprocessor Based High Speed Space Borne Message Switch," Annual Report to NASA on Grant No. NSG-3191, Clarkson College of Technology, Potsdam, N.Y., April 1979.

3. Burnell, James F., "The Design of a Microprocessor-Based High Speed Packet Switch," M.E. Thesis, Clarkson College of Technology, Potsdam, N.Y., August 1979.

4. Russo, Paul M., "Interprocessor Communication for Multi-Microcomputer Systems," IEEE Computer Magazine, pp. 67-76, April 1977.

5. Madnick, Stuart E. and Donovan, John J., Operating Systems, McGraw-Hill, Inc., New York, N.Y., 1974.

6. Advanced Micro Devices Inc., "The AM 2900 Family Data Book," 1978.

7. L. Kleinrock. Queueing Systems, Vol. 2. John Wiley & Sons, New York, 1975. Ch. 3.

8. James Martin. Systems Analysis for Data Transmission. Prentice Hall, New York, 1972. Ch. 31.

9. M. Schwartz. Computer-Communication Network Design and Analysis. Prentice Hall, New York, 1977. Ch. 7.

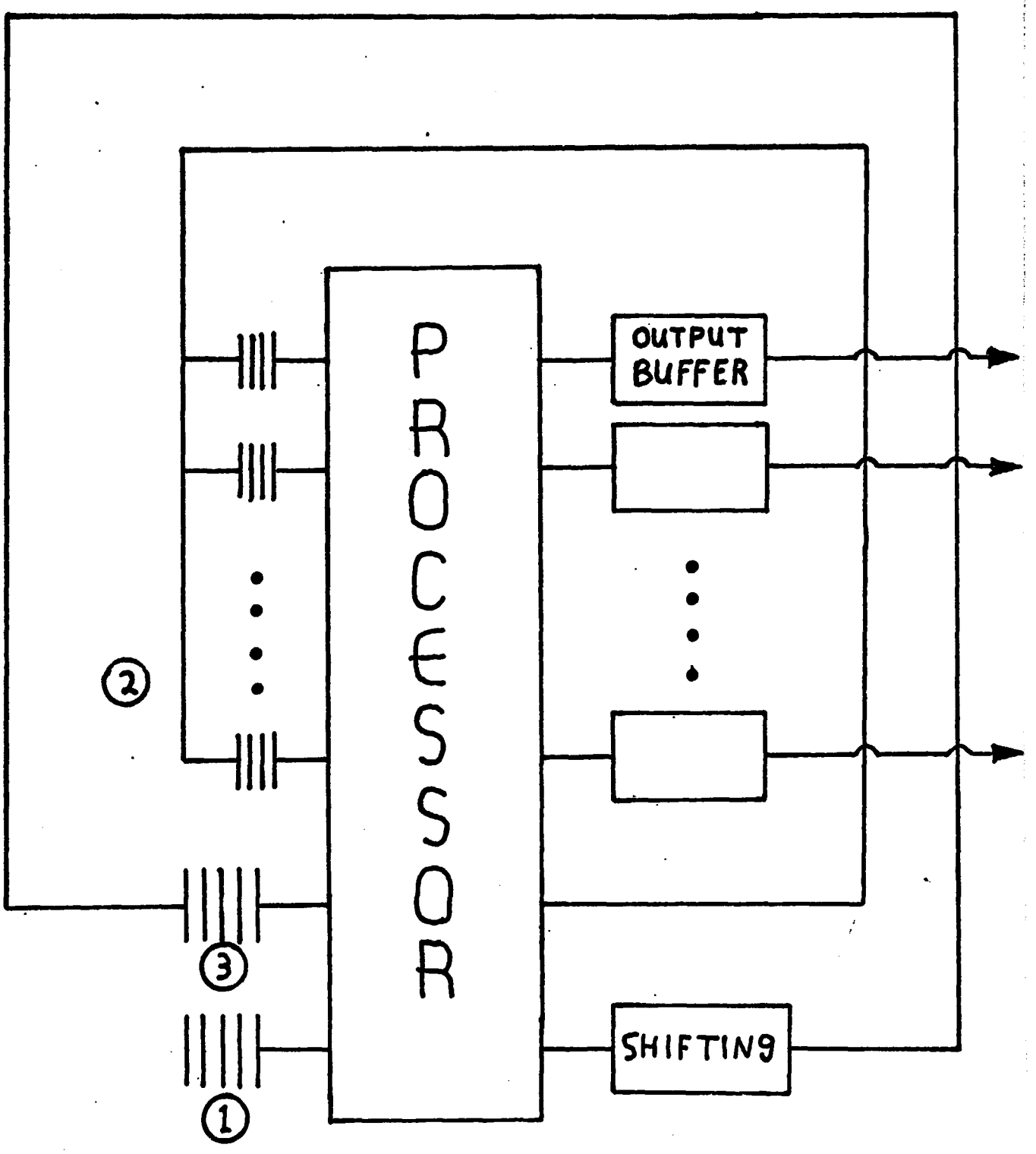


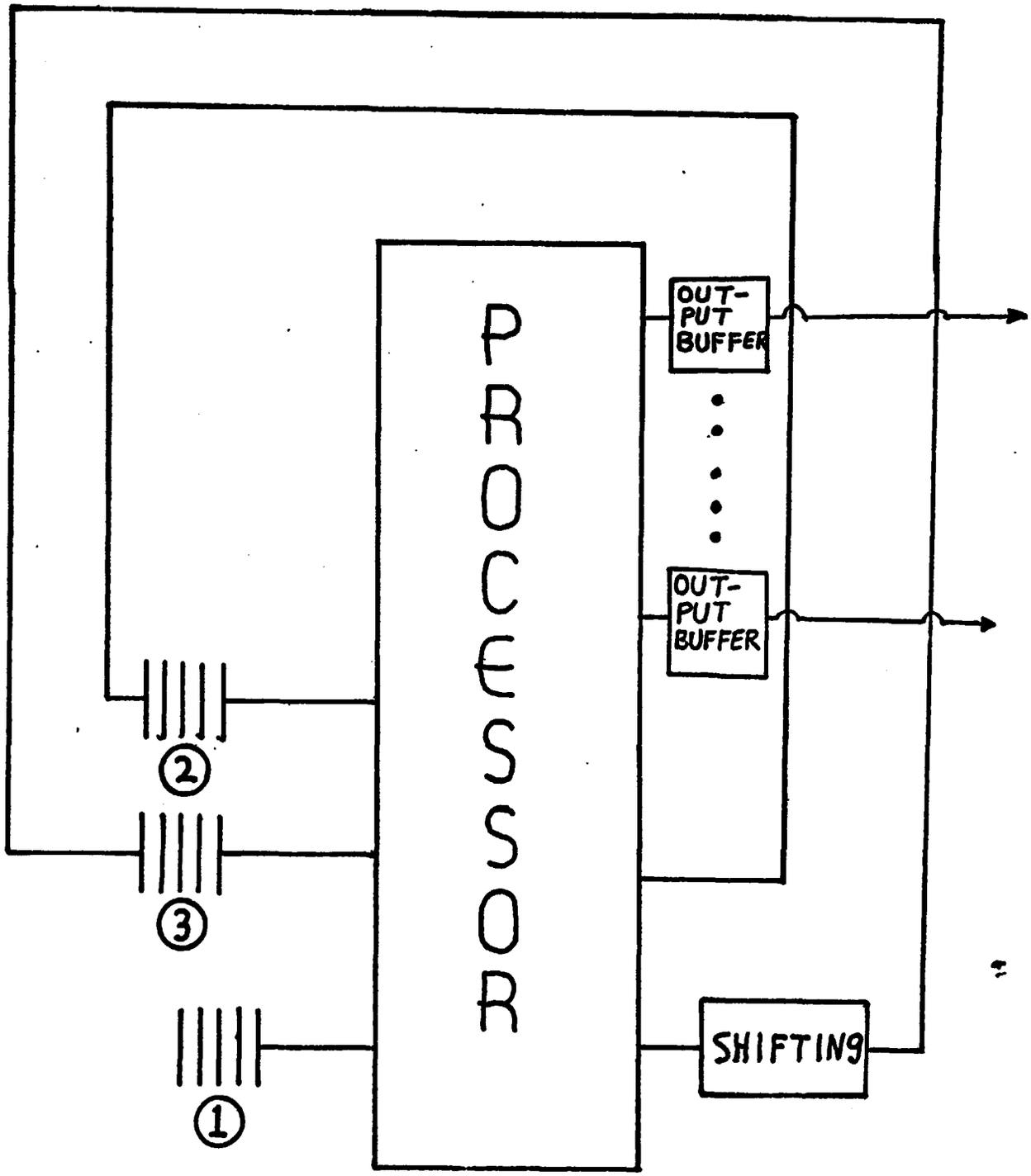
Fig. 6.1. The Queuing Model

Legend:

(1) The input queue with priority 1.

(2) The output queue with priority 2.

(3) The queue for background service with priority 3.



Legend:

(1) The input queue with priority 1.

(2) The output queue with priority 2.

(3) The queue for background service with priority 3.

Fig. 6.2. The Modified Queuing Model.

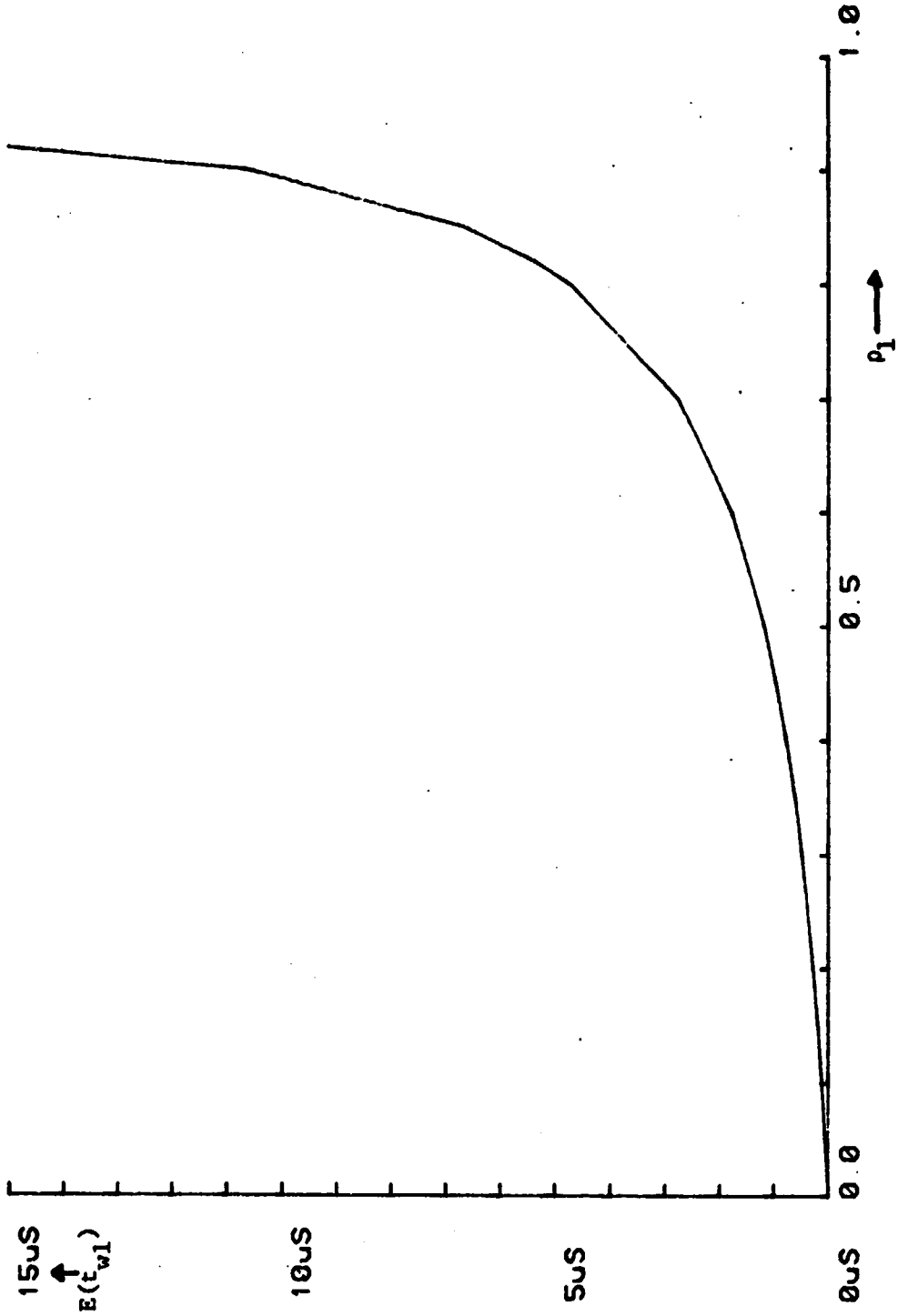


Fig. 6.3. Average Waiting Time Vs. Utilization Factor at Queue 1.

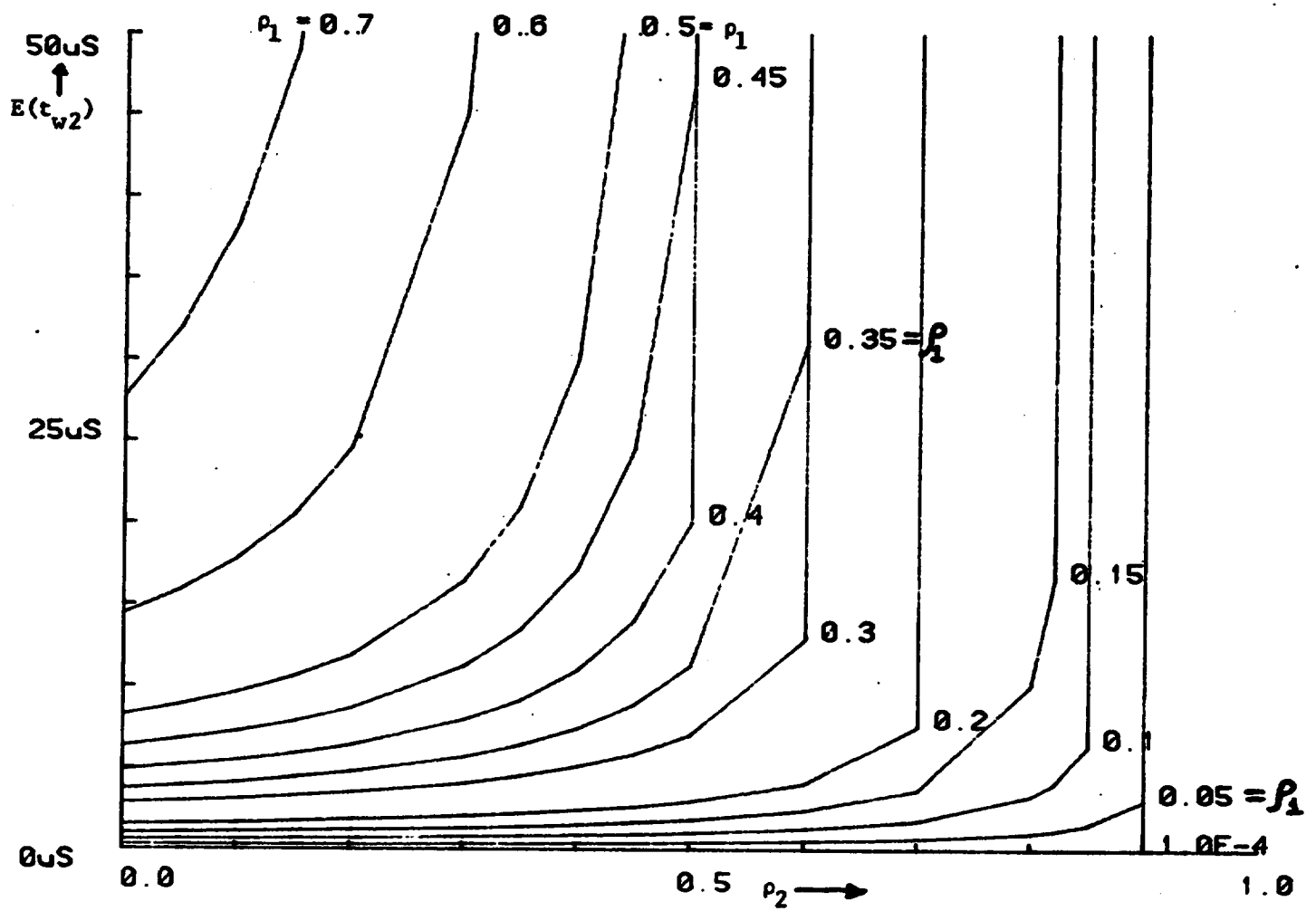


Fig. 6.4. Average Waiting Time Vs. Utilization Factor ρ_2 at Queue 2 With ρ_1 As A Parameter.

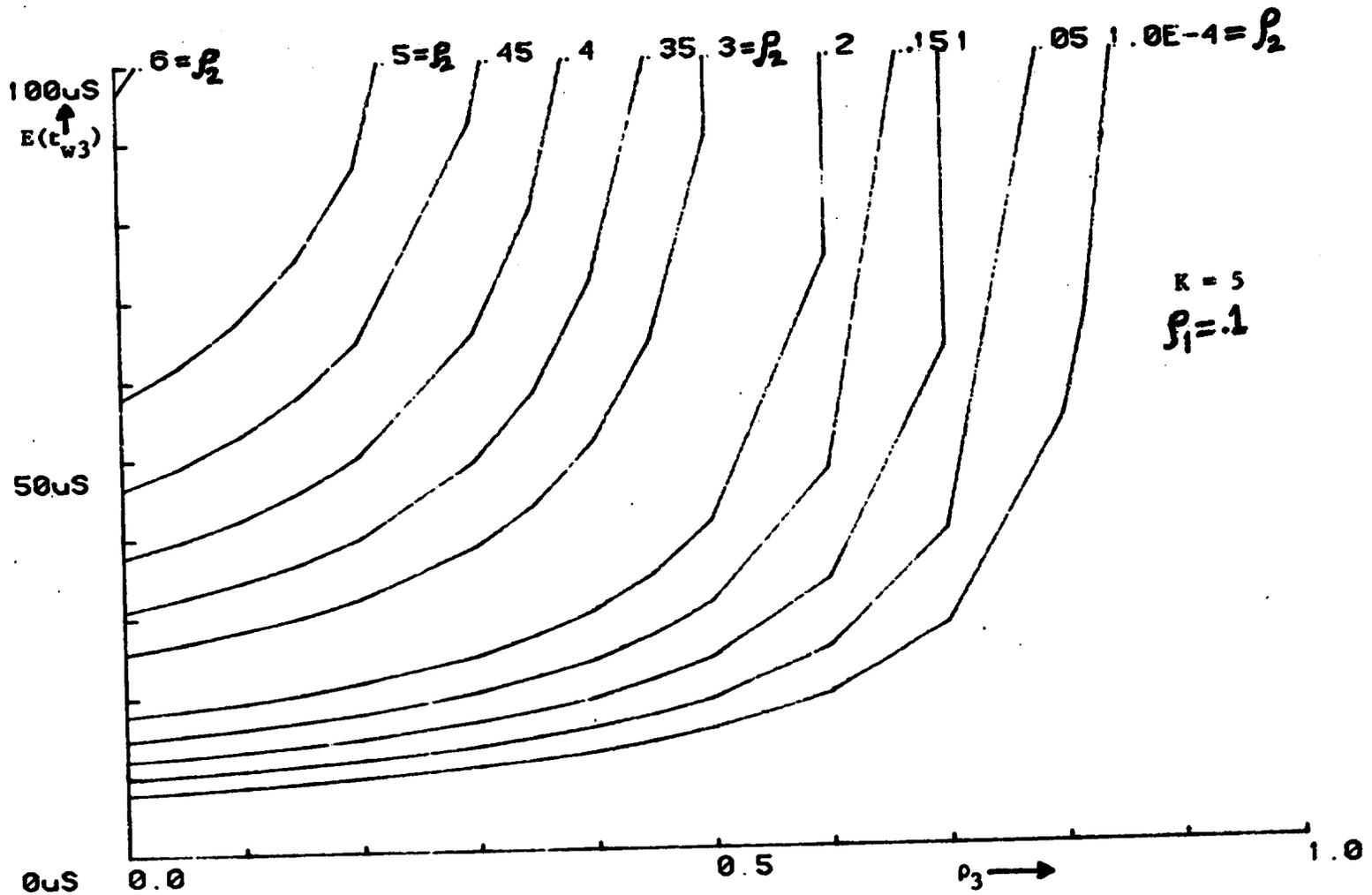


Fig. 6.5. Average Waiting Time Vs. Utilization Factor ρ_3 at Queue 3 With ρ_1 and ρ_2 As Parameters.

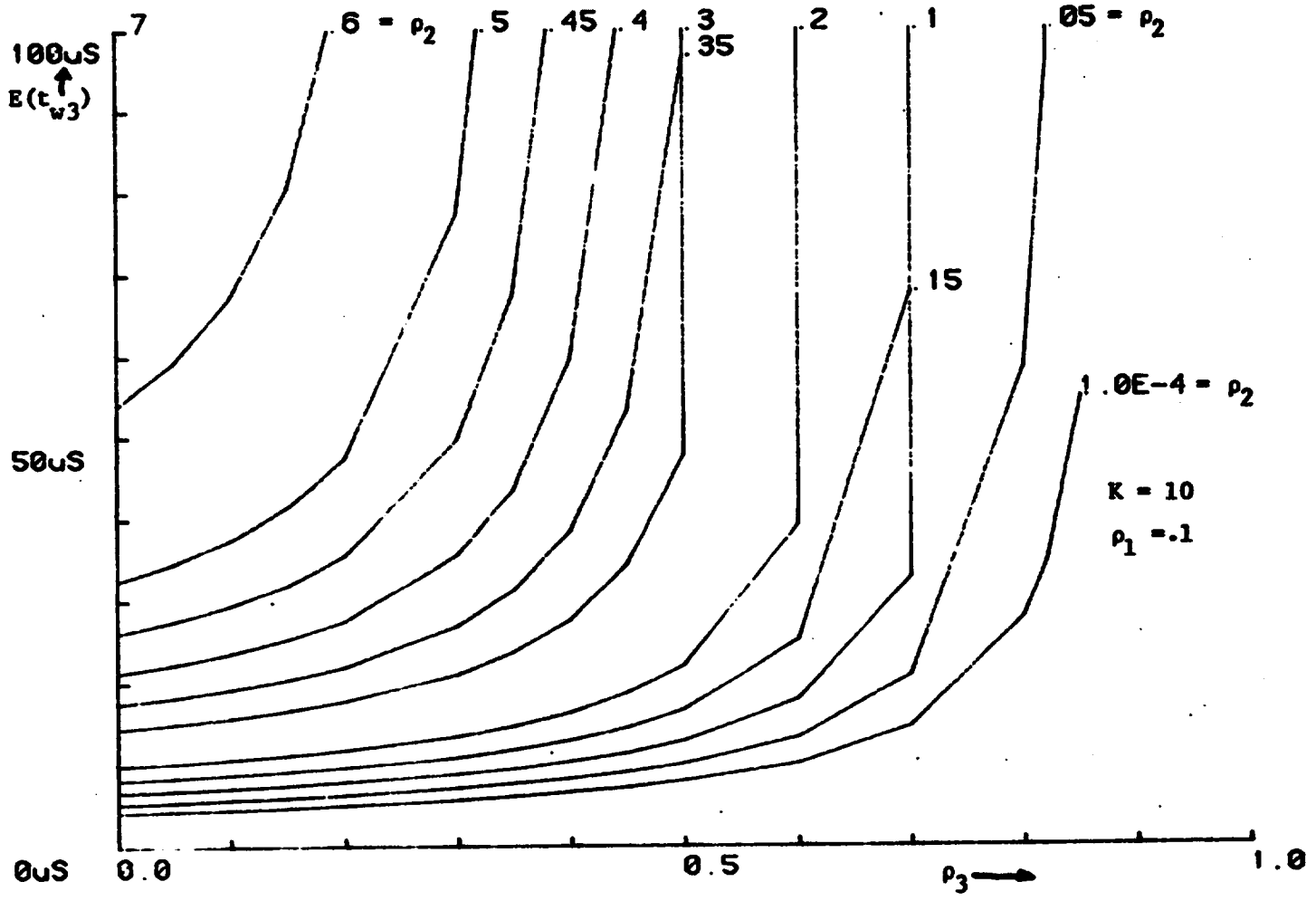


Fig. 6.6. Average Waiting Time Vs. Utilization Factor ρ_3 With ρ_1 and ρ_2 As Parameters. (Queue 3)

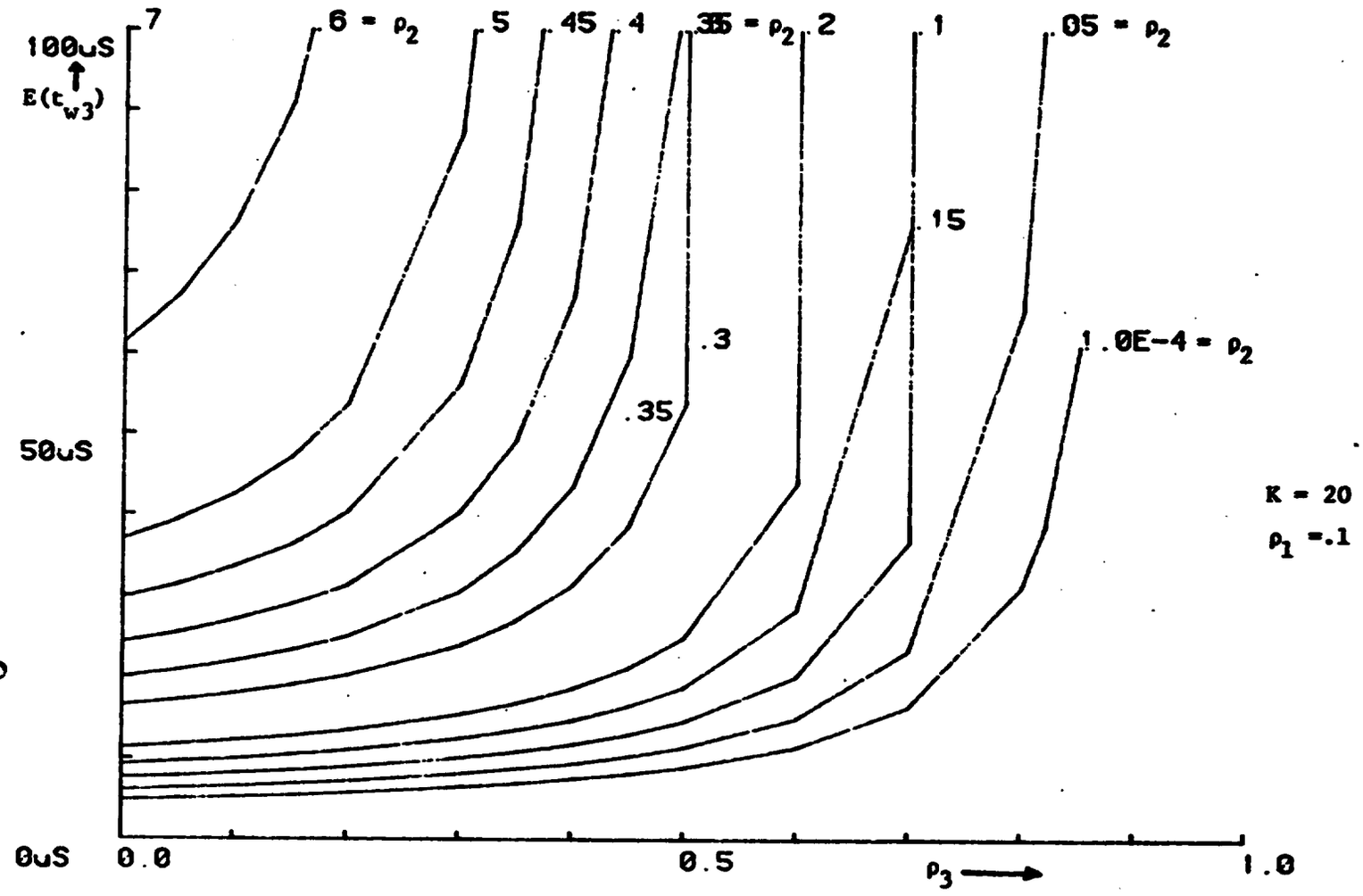


Fig. 6.7. Average Waiting Time Vs. Utilization Factor at Queue 3 With ρ_1 and ρ_2 As Parameters.

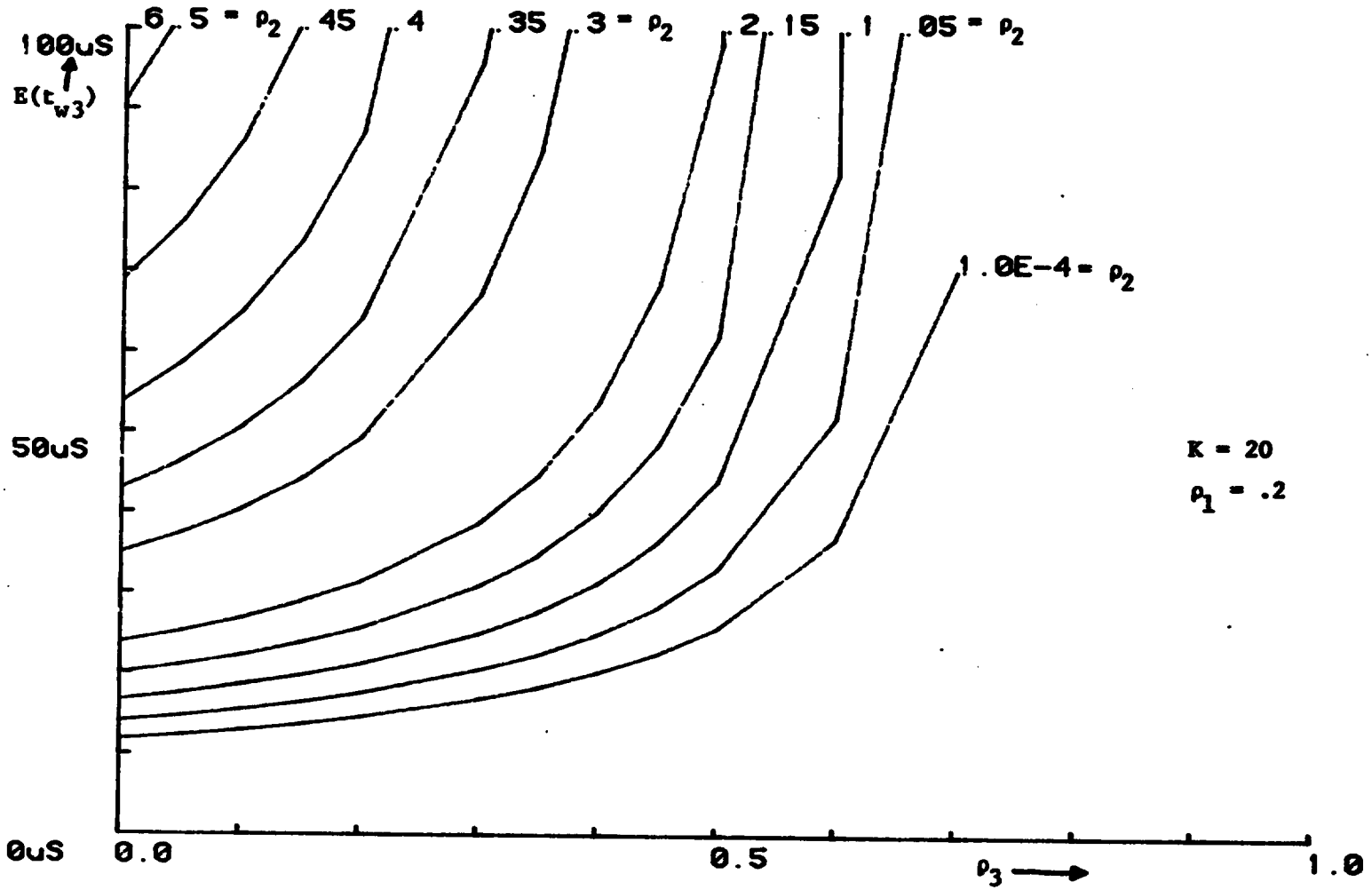


Fig. 6.8. Average Waiting Time Vs. Utilization Factor ρ_3 At Queue 3 With ρ_1 and ρ_2 As Parameters.

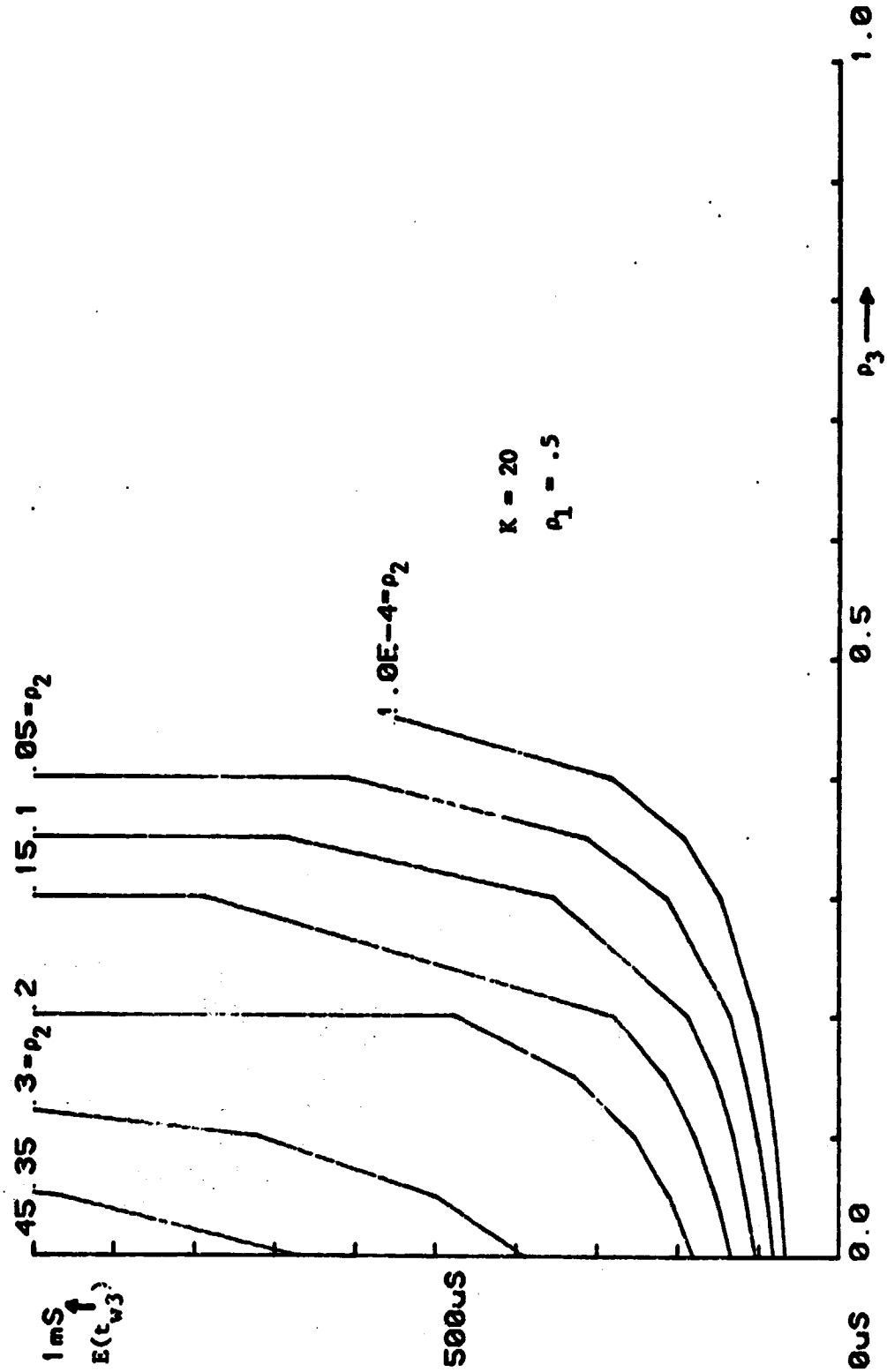


Fig. 6.9. Average Waiting Time Vs. Utilization Factor ρ_3 With ρ_1 and ρ_2 As Parameters.

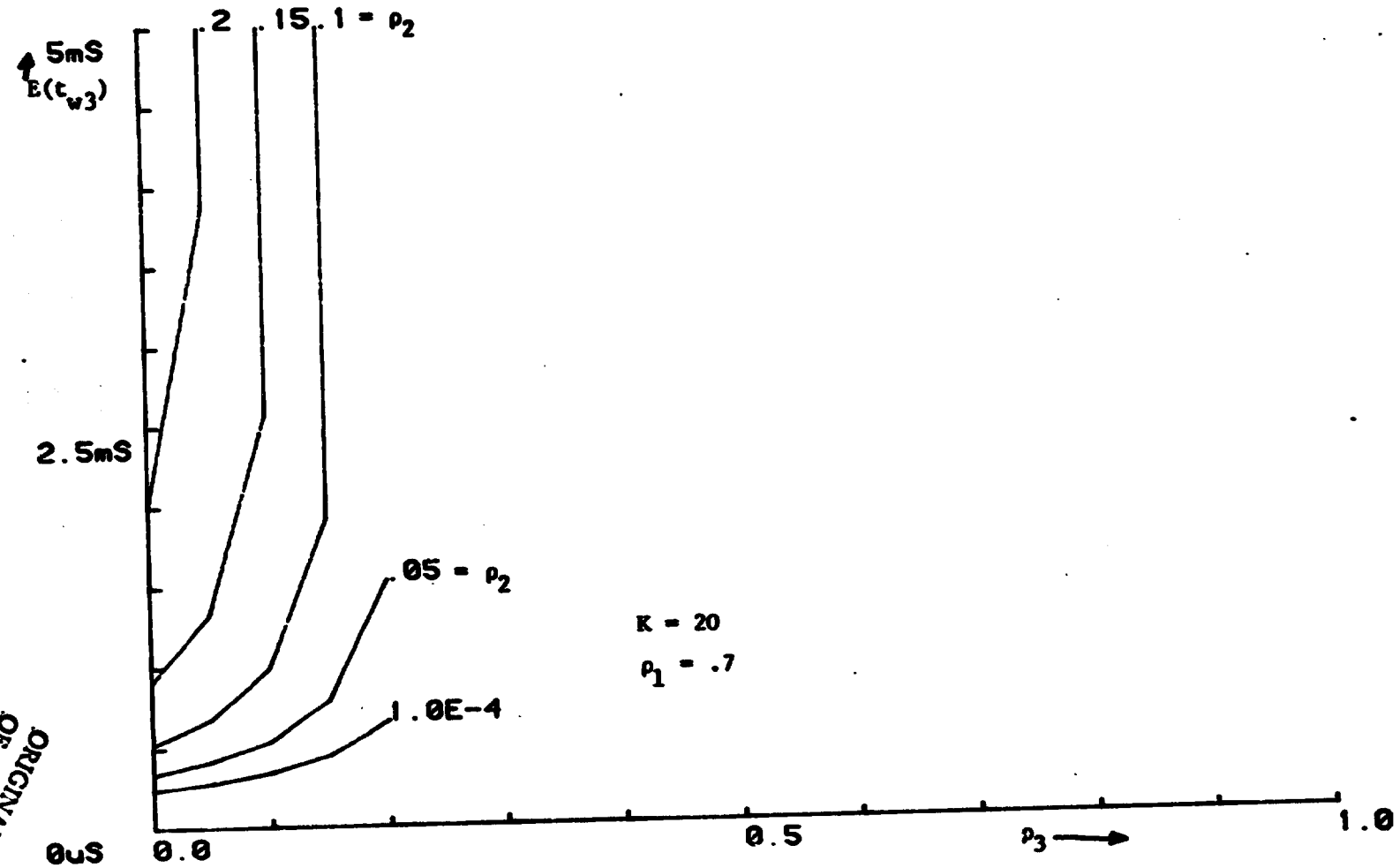


Fig. 6.10. Average Waiting Time Vs. Utilization Factor ρ_3 With ρ_1 and ρ_2 As Parameters.

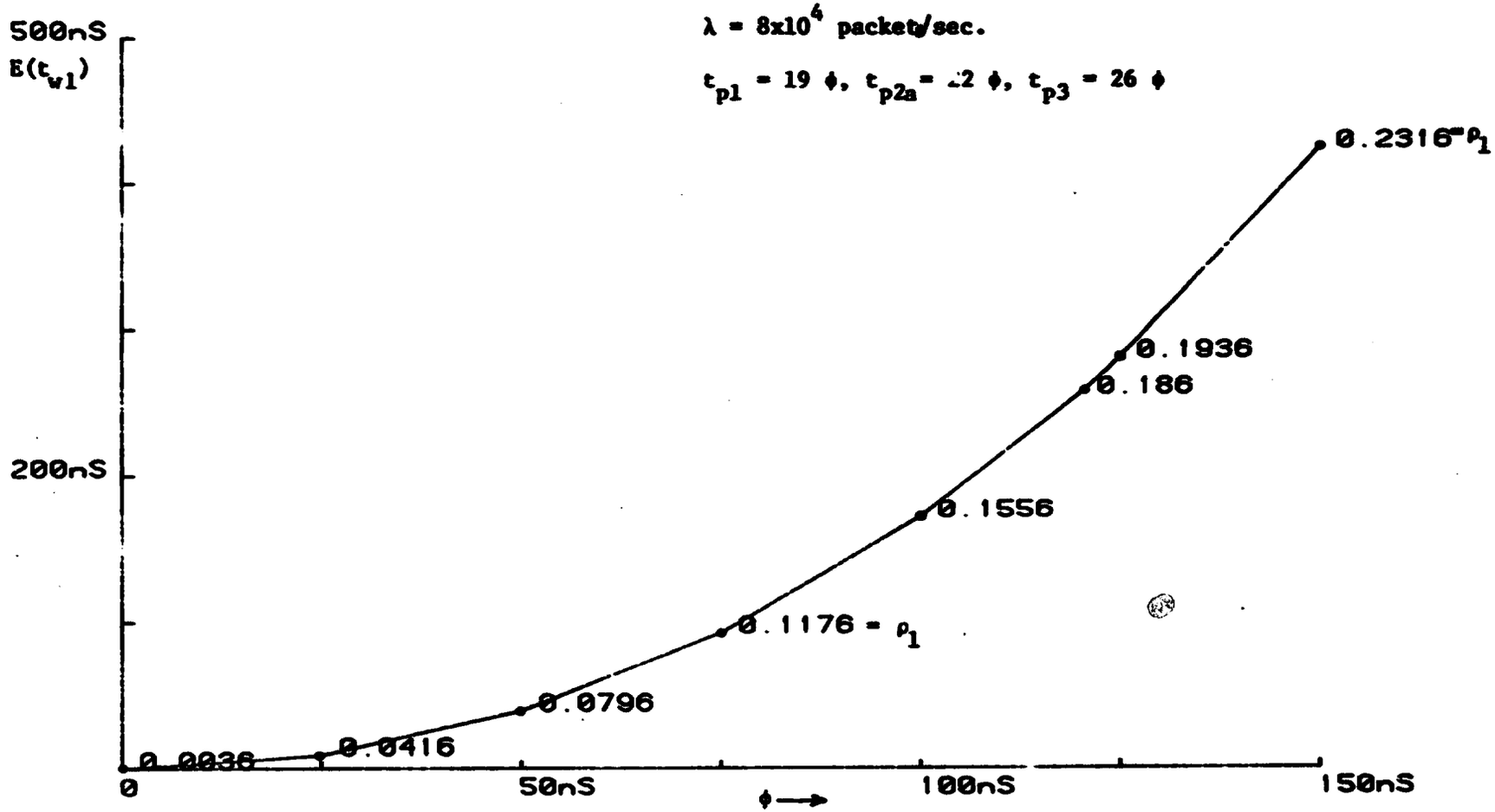


Fig. 6.11. Average Waiting Time at Queue 1 Vs. Clock Cycle Time of The Processor.
(For the Proposed Design ϕ = 120 ns.)

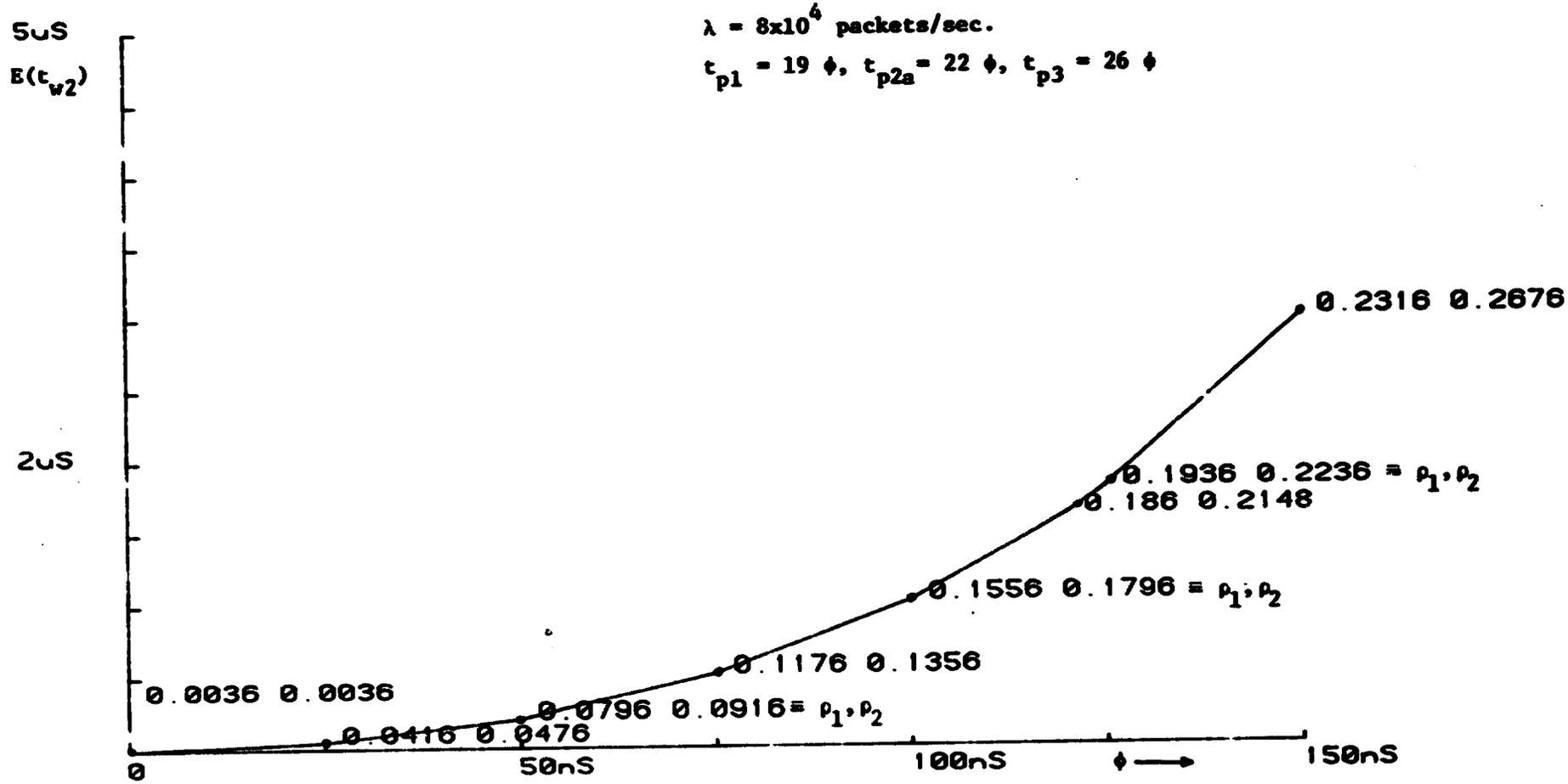


Fig. 6.12. Average Waiting Time at Queue 2 Vs. Clock Cycle Time of The Processor.
(For The Proposed Design ϕ = 120 ns.)

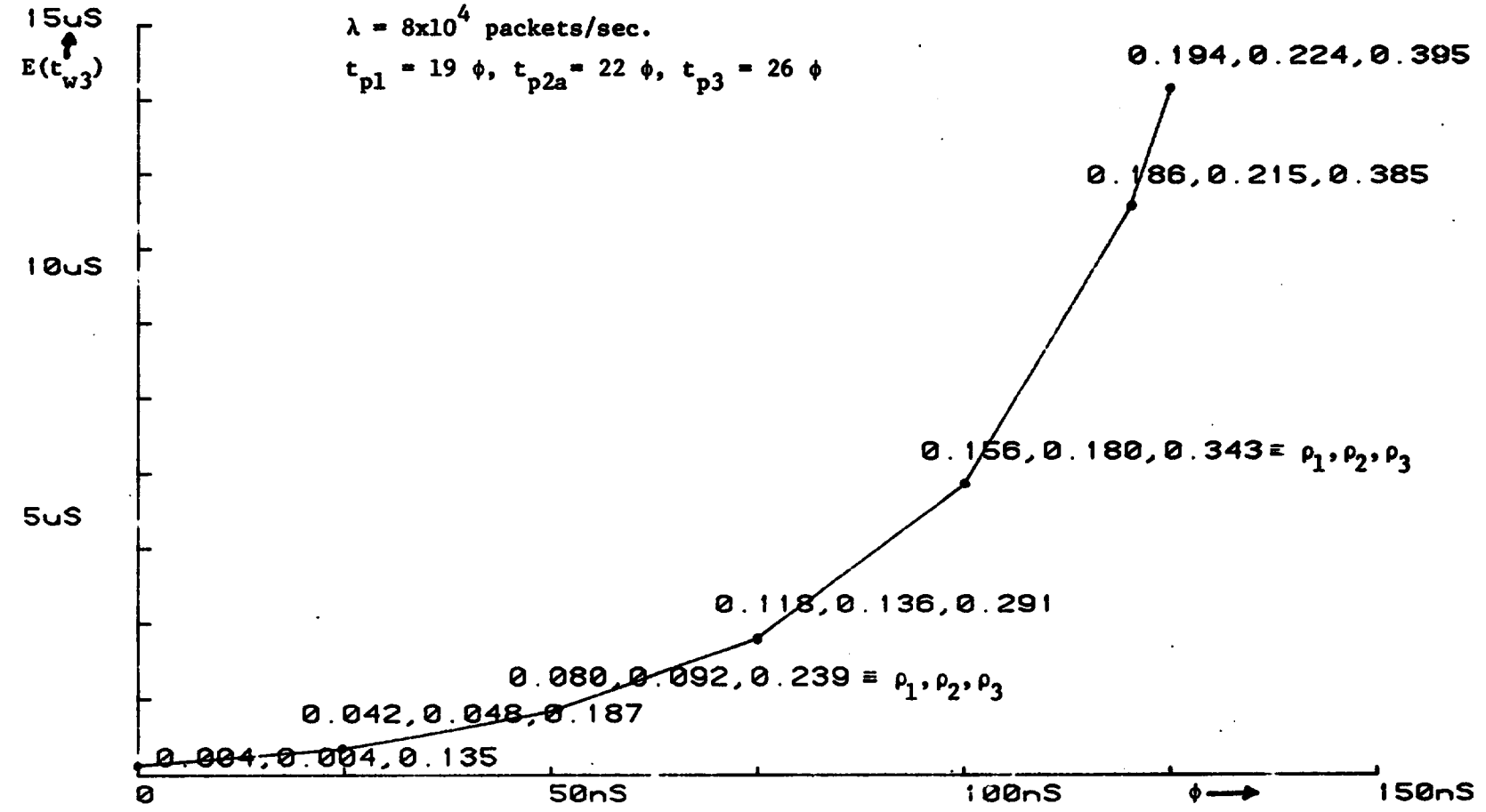


Fig. 6.13. Average Waiting Time At Queue 3 Vs. Clock Cycle Time of The Processor.
(For the Proposed Design ϕ = 120 ns.)

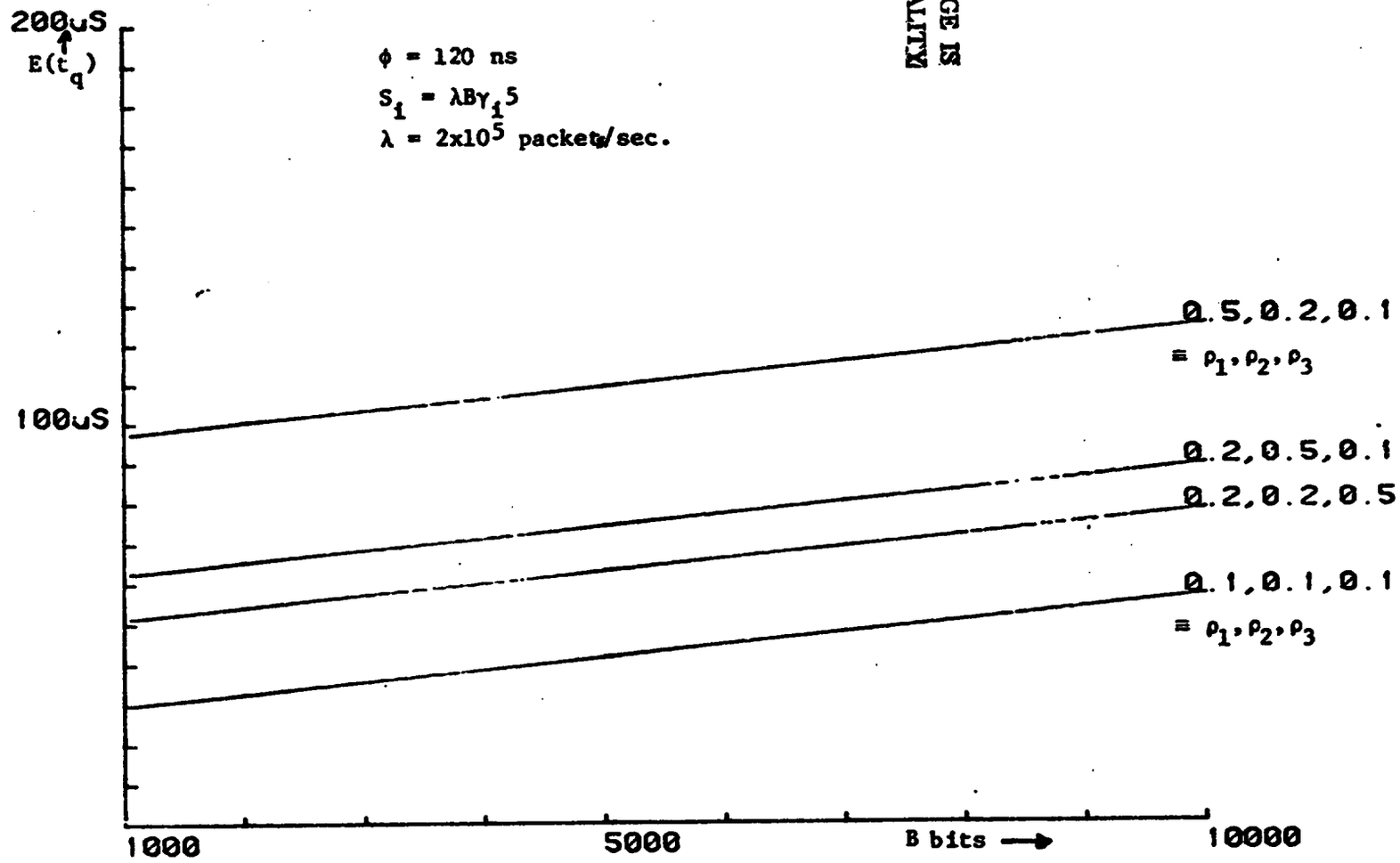
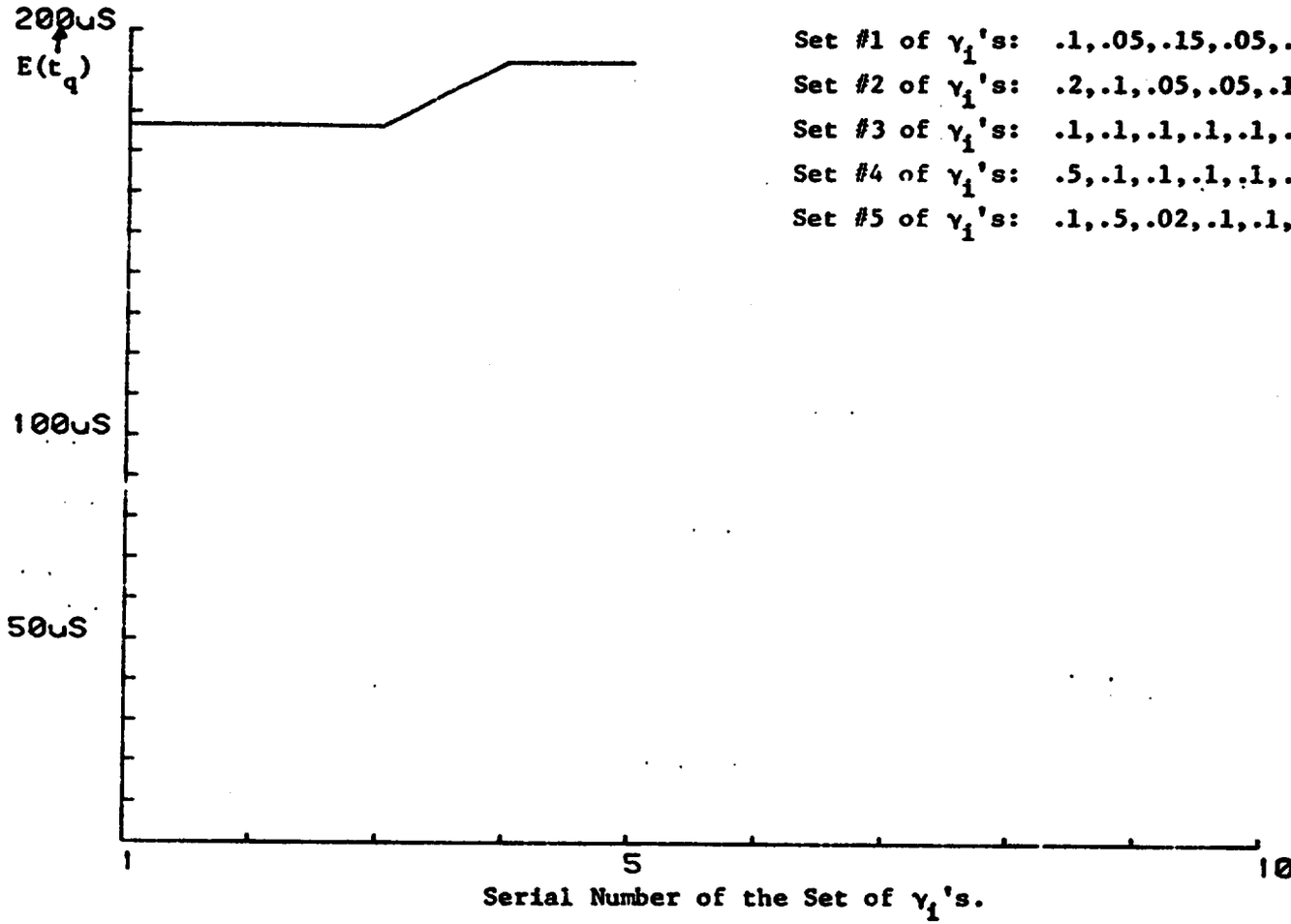


Fig. 6.14. Overall Average Response Time Vs. Packet Size B With ρ_1, ρ_2 and ρ_3 As Parameters.



Set #1 of γ_i's: .1,.05,.15,.05,.15,.025,.075,.2,.15,.05

Set #2 of γ_i's: .2,.1,.05,.05,.1,.075,.025,.2,.15,.05

Set #3 of γ_i's: .1,.1,.1,.1,.1,.1,.1,.1,.1,.1

Set #4 of γ_i's: .5,.1,.1,.1,.1,.02,.02,.02,.02,.02

Set #5 of γ_i's: .1,.5,.02,.1,.1,.1,.02,.02,.02,.02

Fig. 6.15. Overall Average Response Time Vs. Destination Functions.
($\lambda=10^5$ packets/sec., B=1024 bits, ϕ=120 ns, ρ_1=.2325, ρ_2=.2685, ρ_3=.48075, $S_i = \frac{\lambda B 8}{M}$)

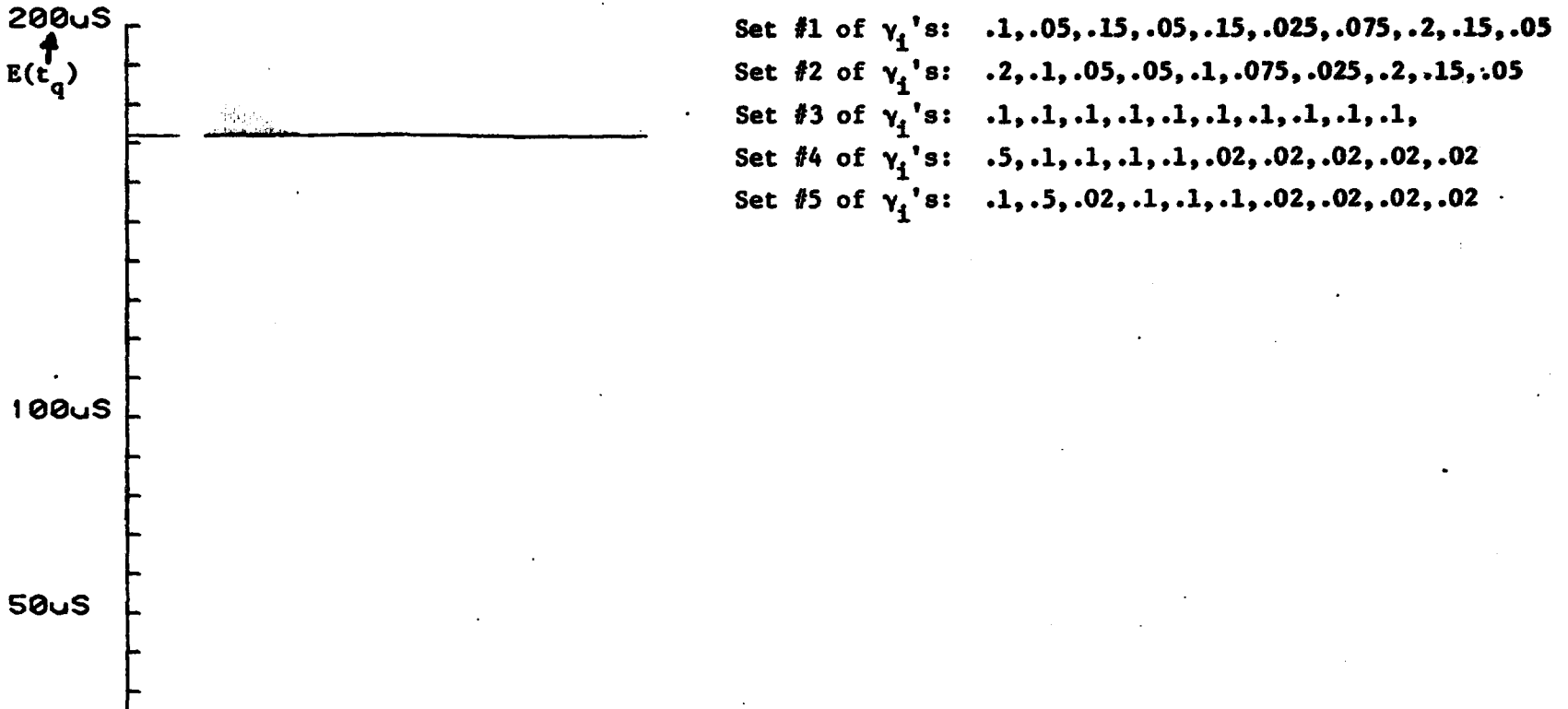


Fig. 6.16. Overall Average Response Time Vs. Desitnation Functions.
($\lambda=10^5$ packets/sec., B=1024 bits, ϕ=120 ns, ρ_1=.2325, ρ_2=.2685, ρ_3=.48075, $S_i = \gamma_i \lambda B5$)

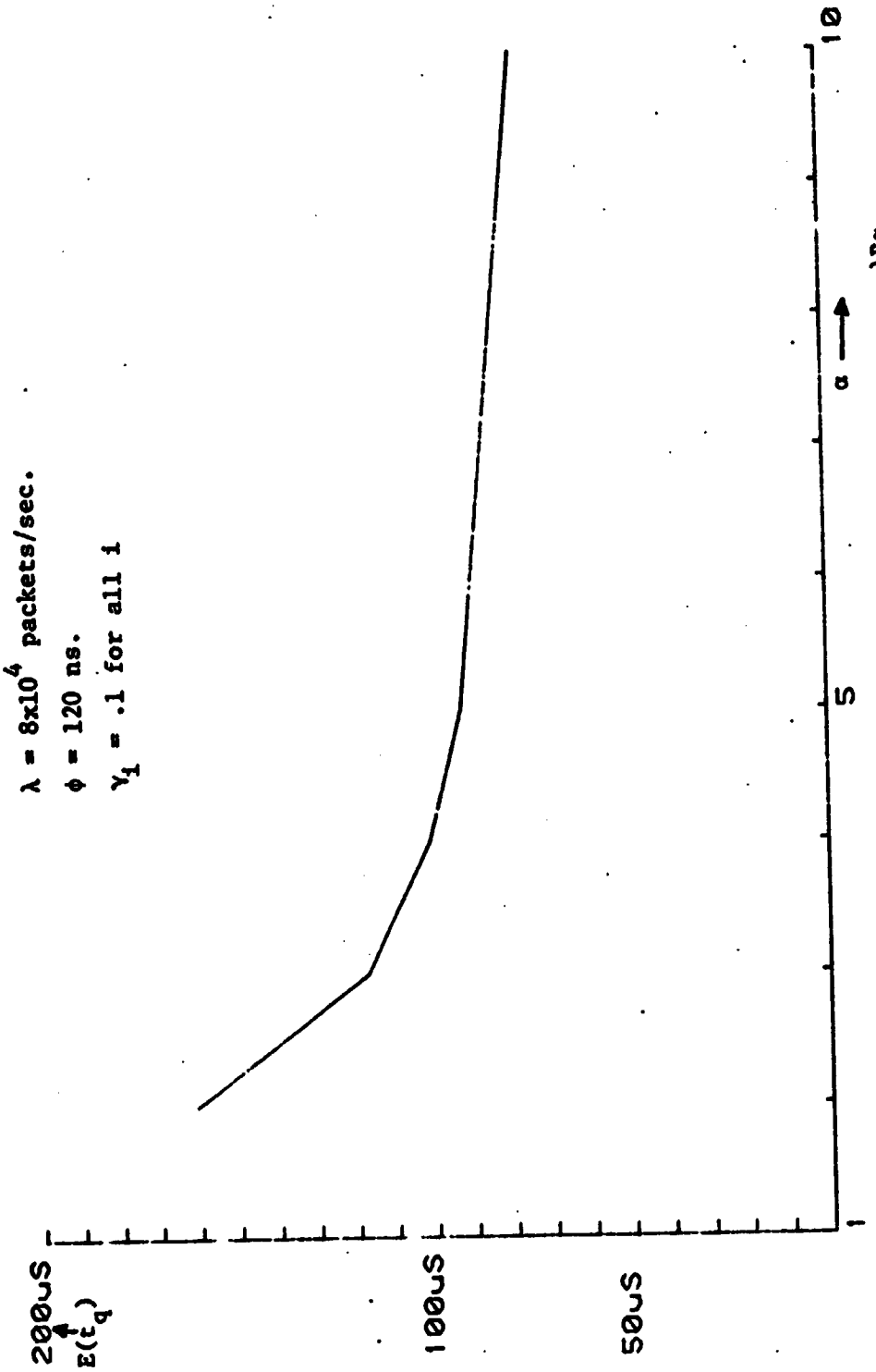


Fig. 6.17. Overall Average Response Time Vs. α in $S_i = \frac{\lambda B \alpha}{M}$.

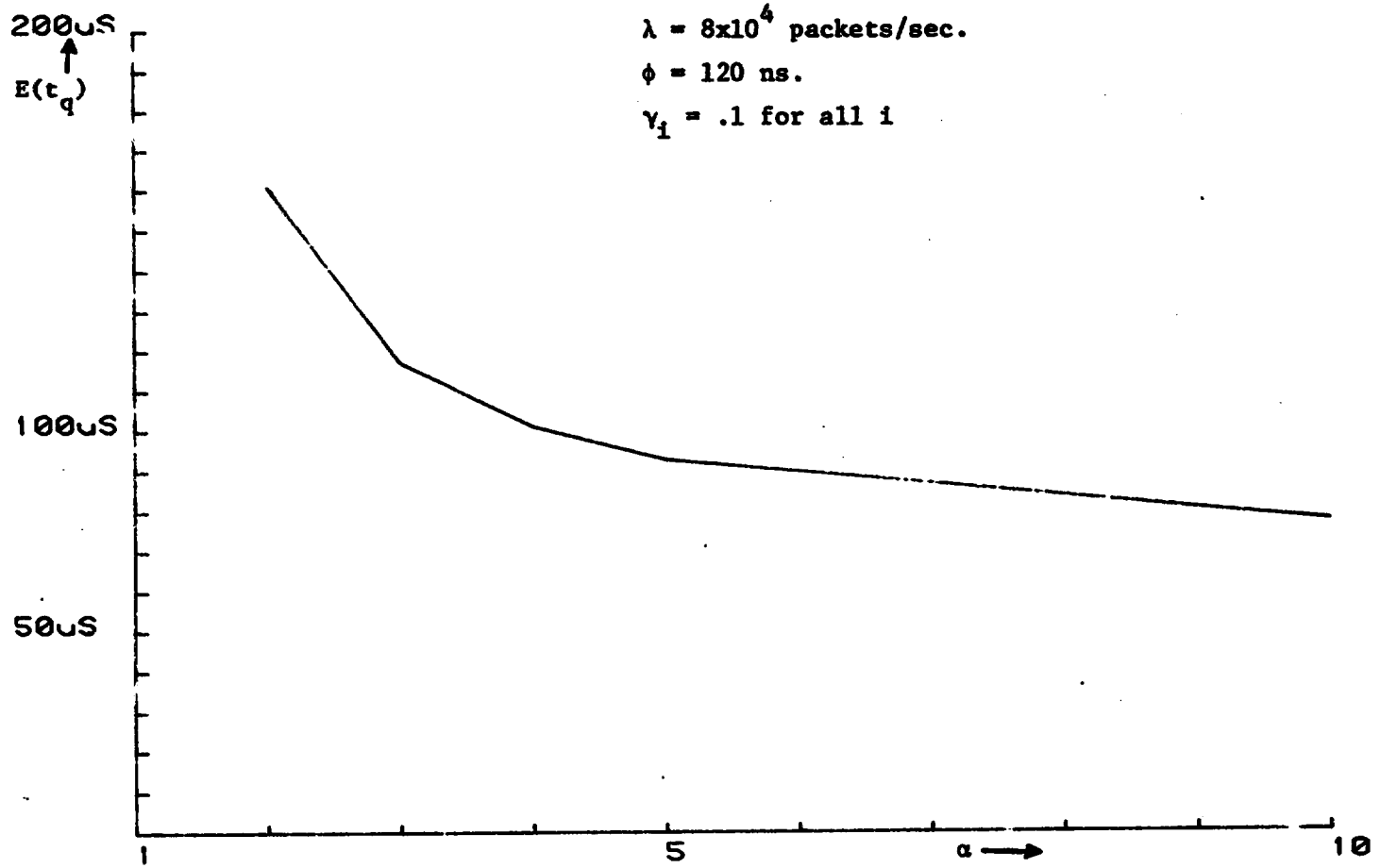


Fig. 6.18. Overall Average Response Time Vs. α in $S_i = \lambda B \gamma_i \alpha$.

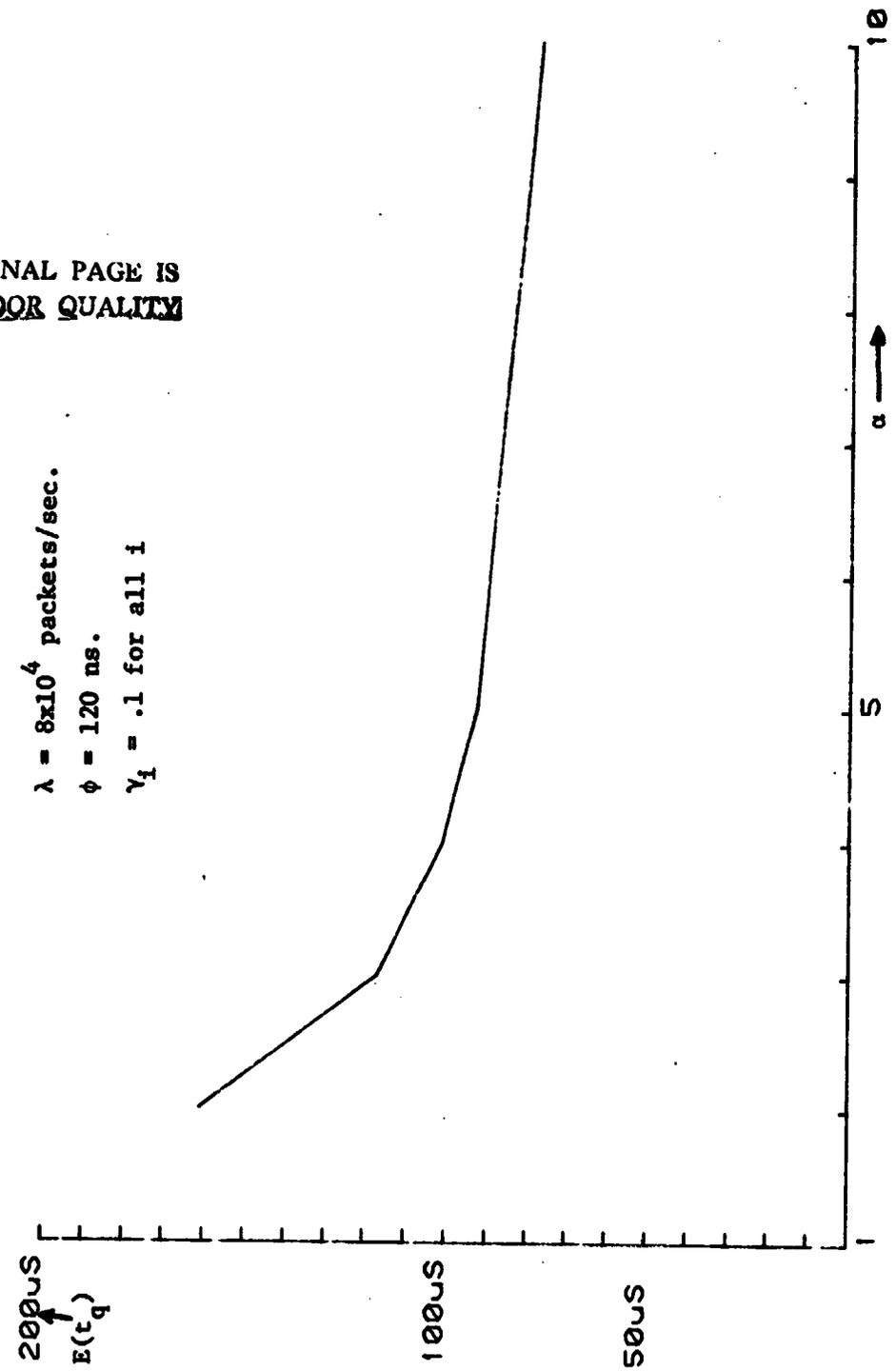


Fig. 6.19. Overall Average Response Time Vs. α in $S_i = \lambda B\gamma_i + \frac{\lambda B(\alpha-1)\sqrt{\lambda B\gamma_i}}{\sum_i \sqrt{\lambda B\gamma_i}}$.

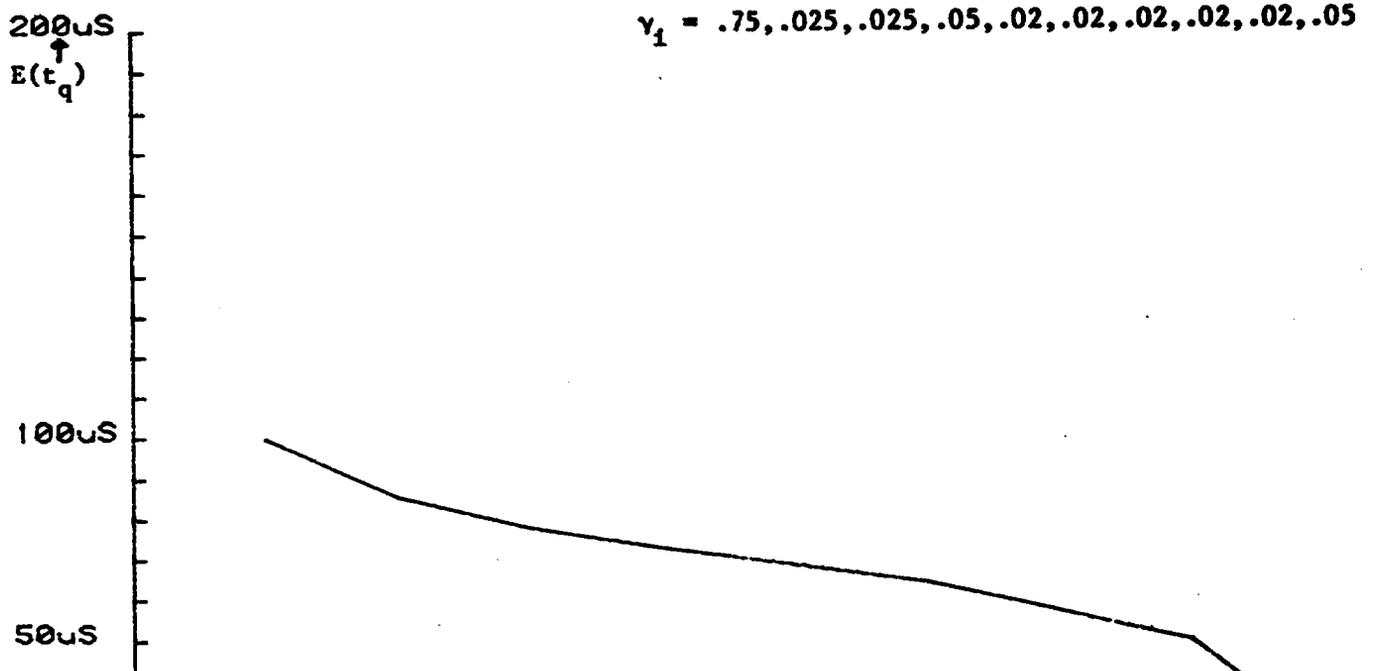


Fig. 6.20. Overall Average Response Time Vs. α in $S_i = \frac{\lambda B \alpha}{M}$.

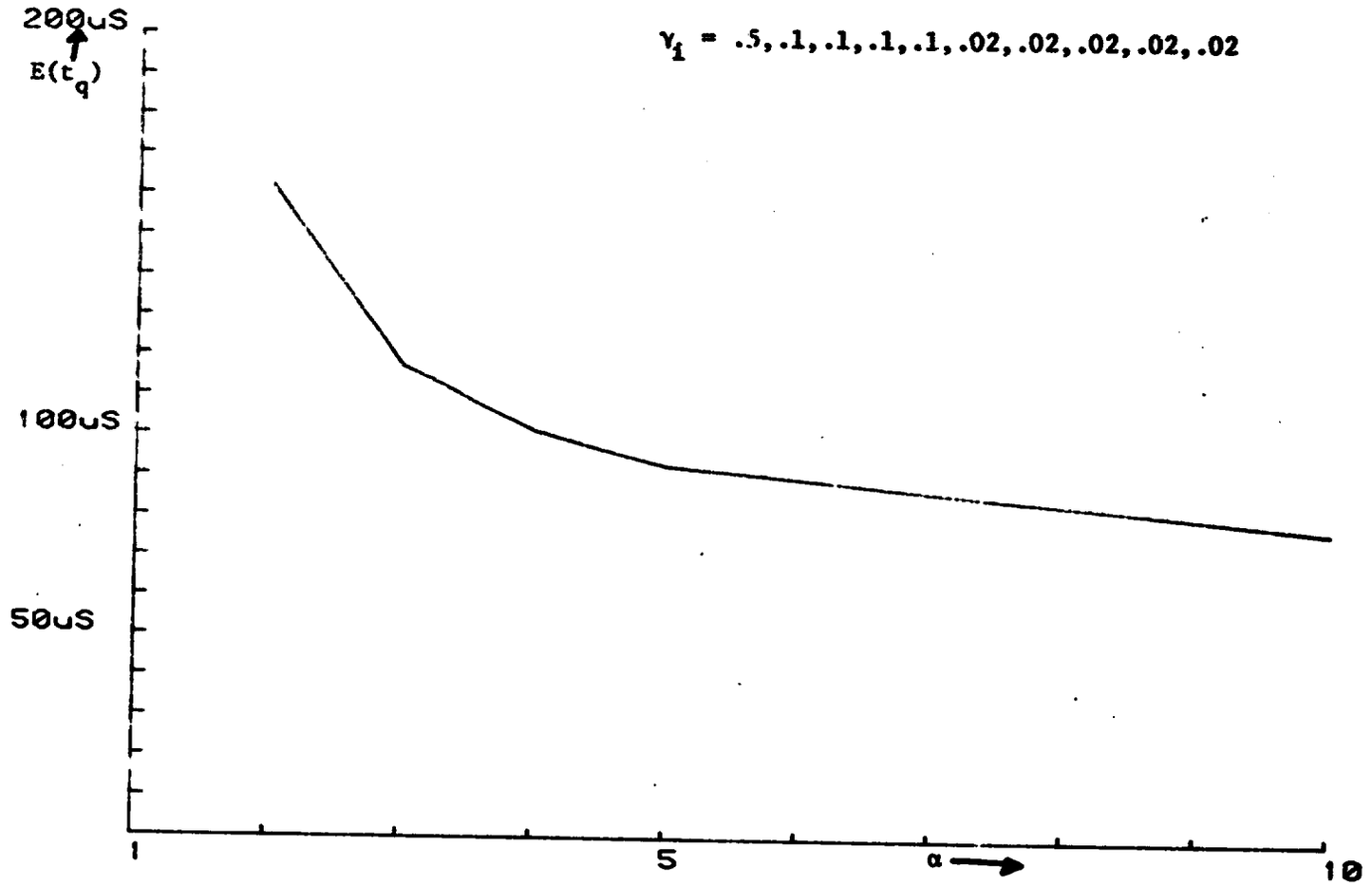
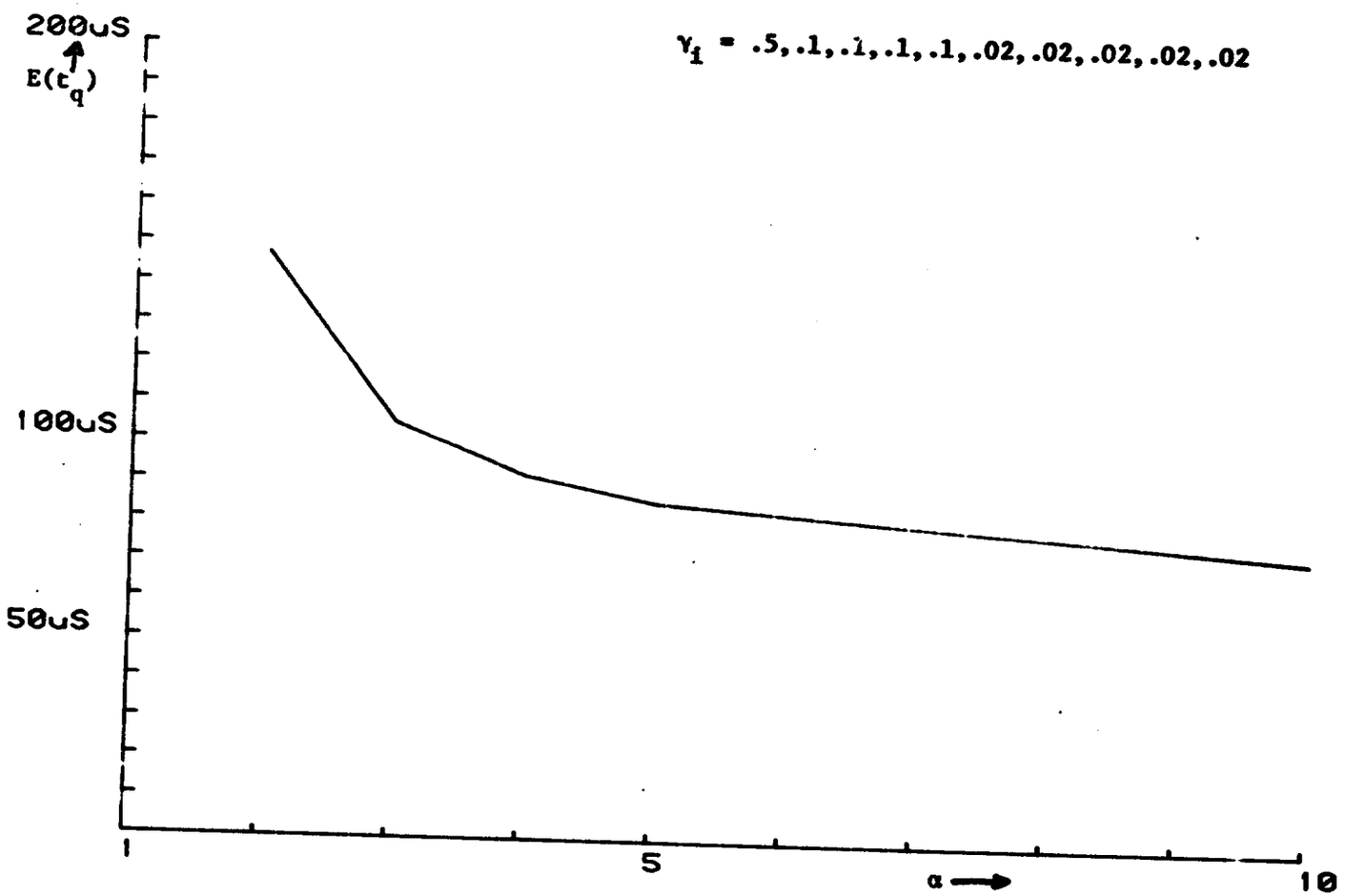


Fig. 6.21. Overall Average Response Time Vs. α in $S_i = \lambda B \gamma_i \alpha$.



γ_i = .5,.1,.1,.1,.1,.02,.02,.02,.02,.02

Fig. 6.22. Overall Average Response Time Vs. α in $S_i = \lambda B\gamma_i + \dfrac{\lambda B(\alpha-1)\sqrt{\lambda B\gamma_i}}{\sum_i \sqrt{\lambda B\gamma_i}}$.

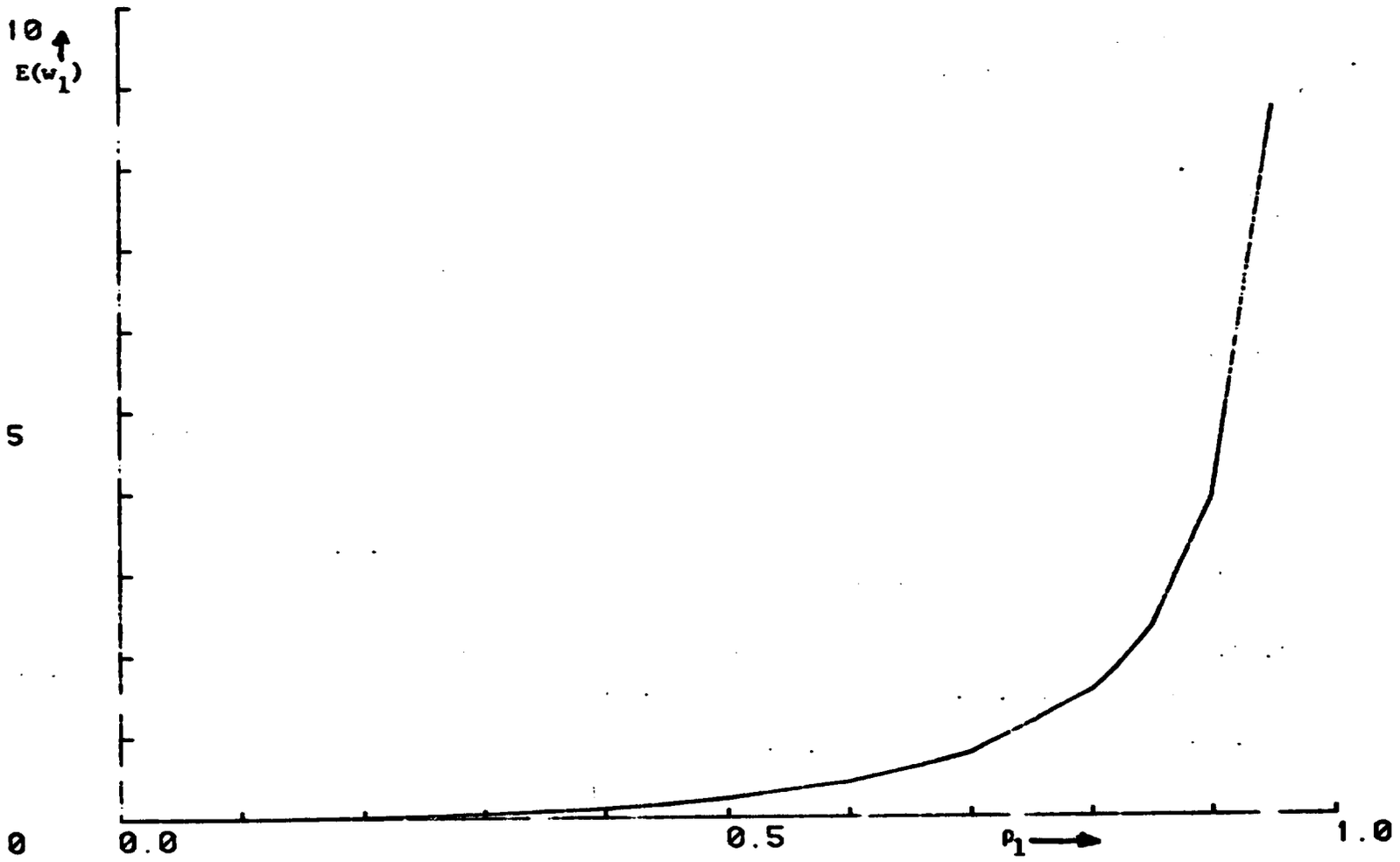


Fig. 6.23. Average Queue Size Vs. Utilization Factor at Queue 1.

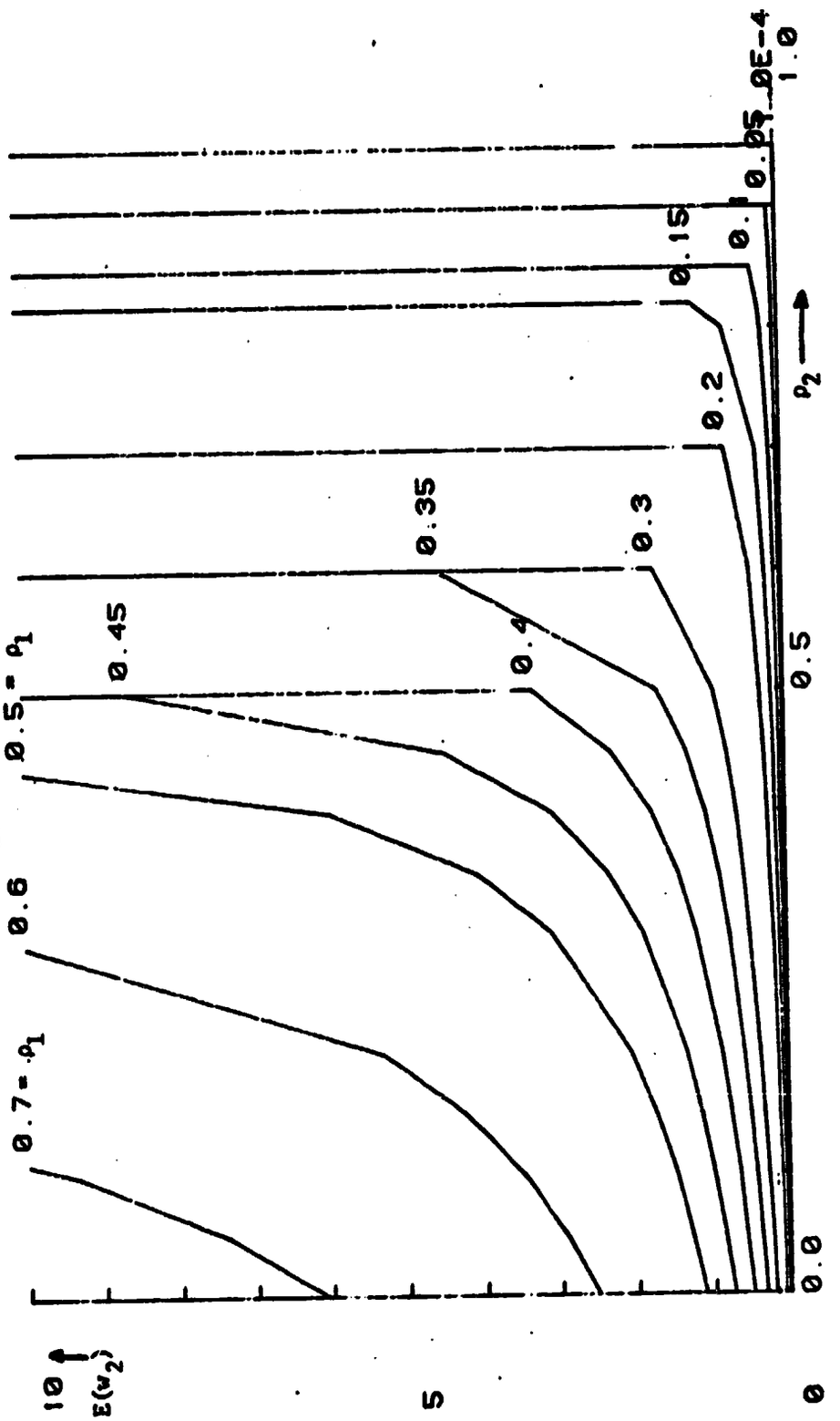


Fig. 6.24. Average Queue Size Vs. Utilization Factor at Queue 2 With ρ_1 As A Parameter.

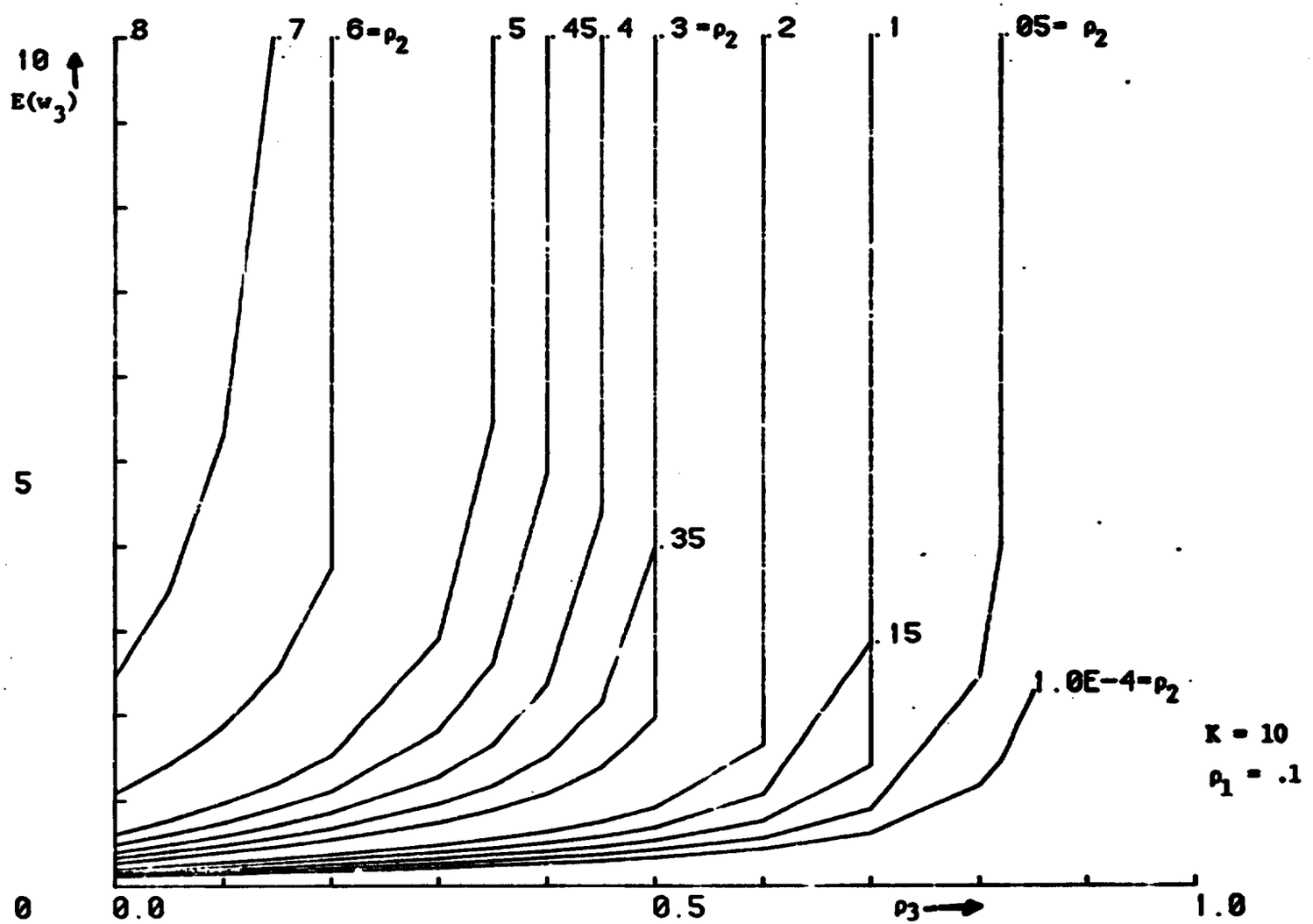


Fig. 6.25. Average Queue Size Vs. Utilization Factor At Queue 3 With ρ_1 and ρ_2 As Parameters.

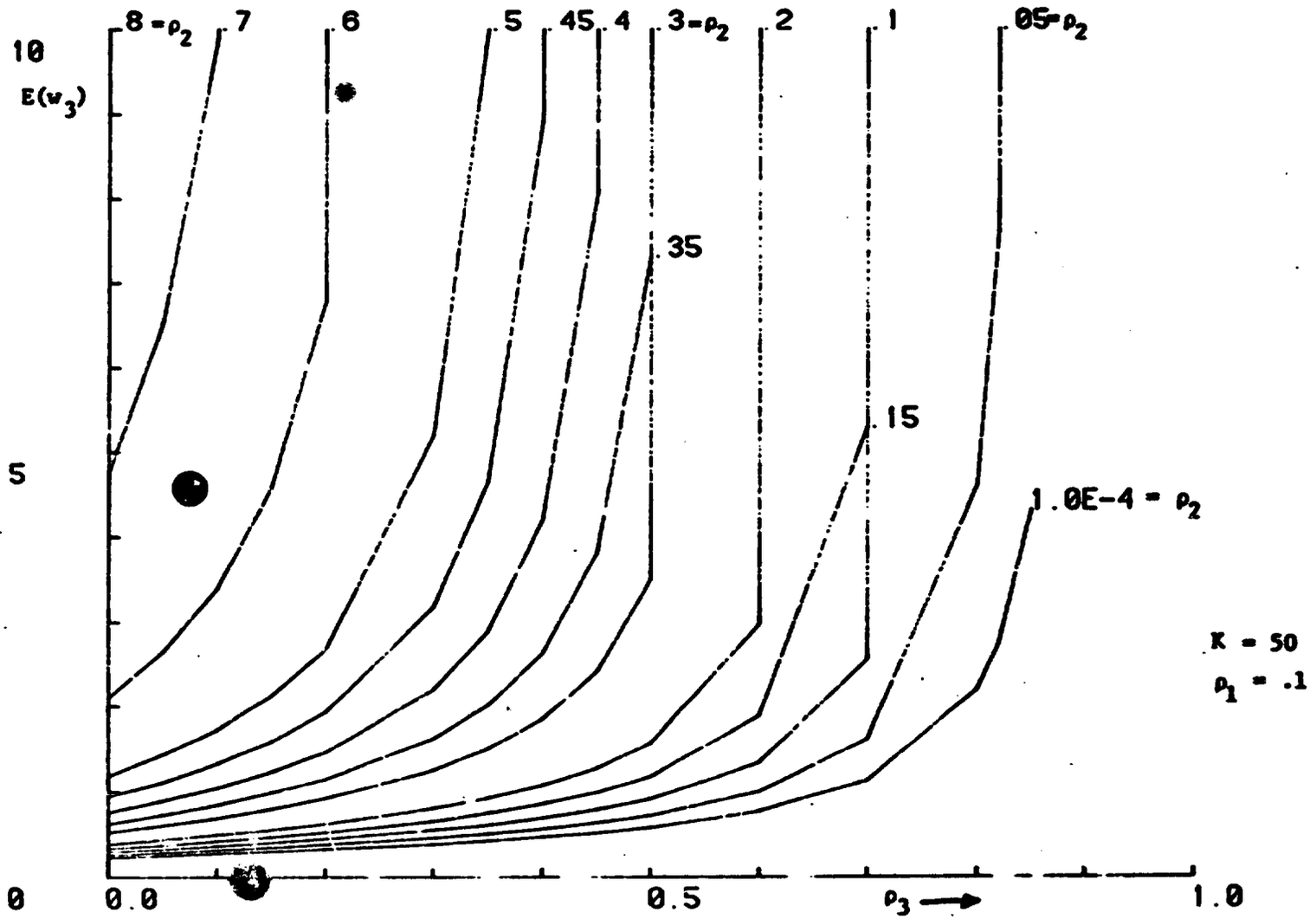


Fig. 6.26. Average Queue Size Vs. Utioization Factor At Queue 3 With ρ_1 and ρ_2 As Parameters.

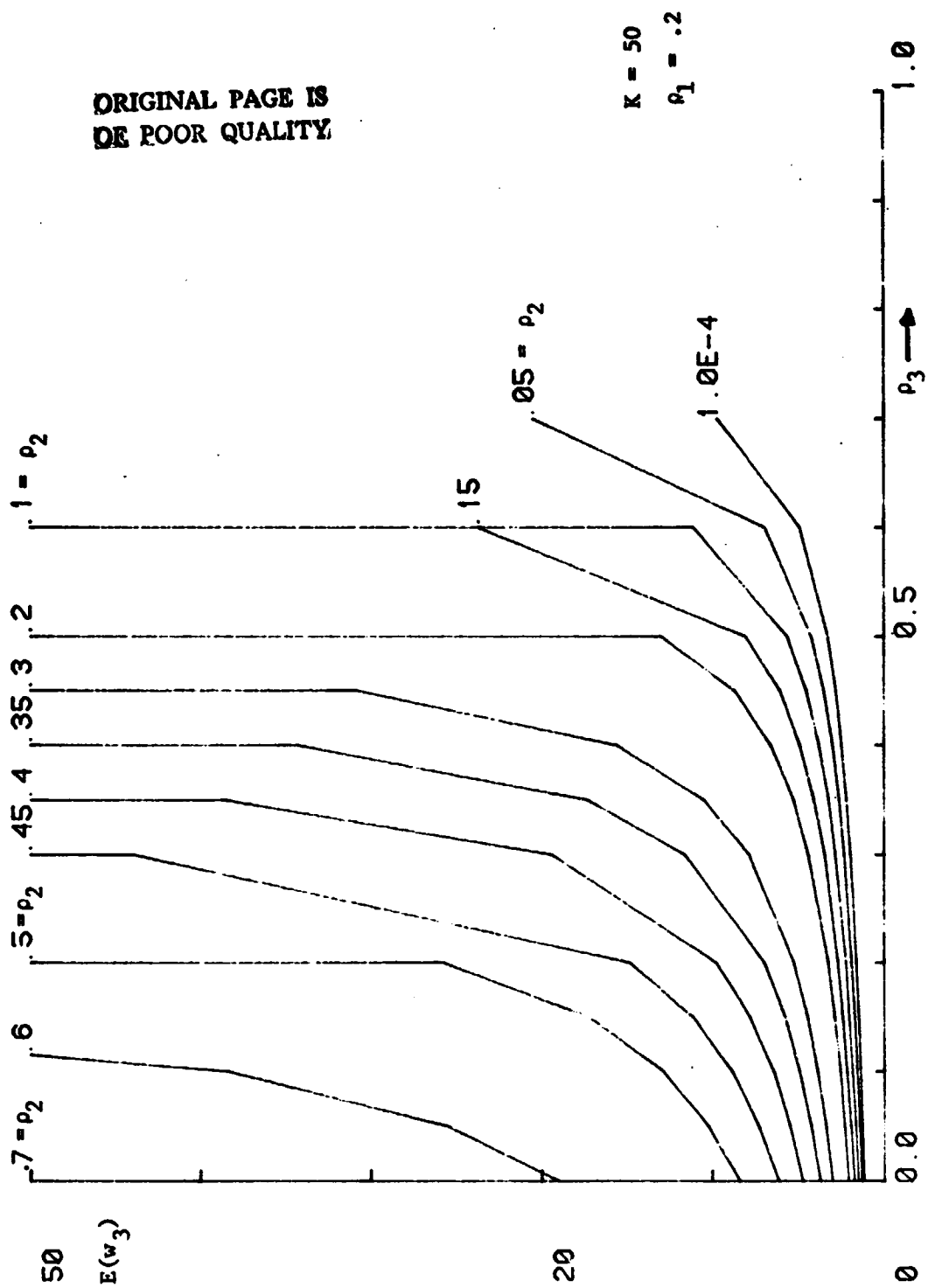


Fig. 6.27. Average Queue Size Vs. Utilization Factor At Queue 3 With ρ_1 and ρ_2 As Parameters.

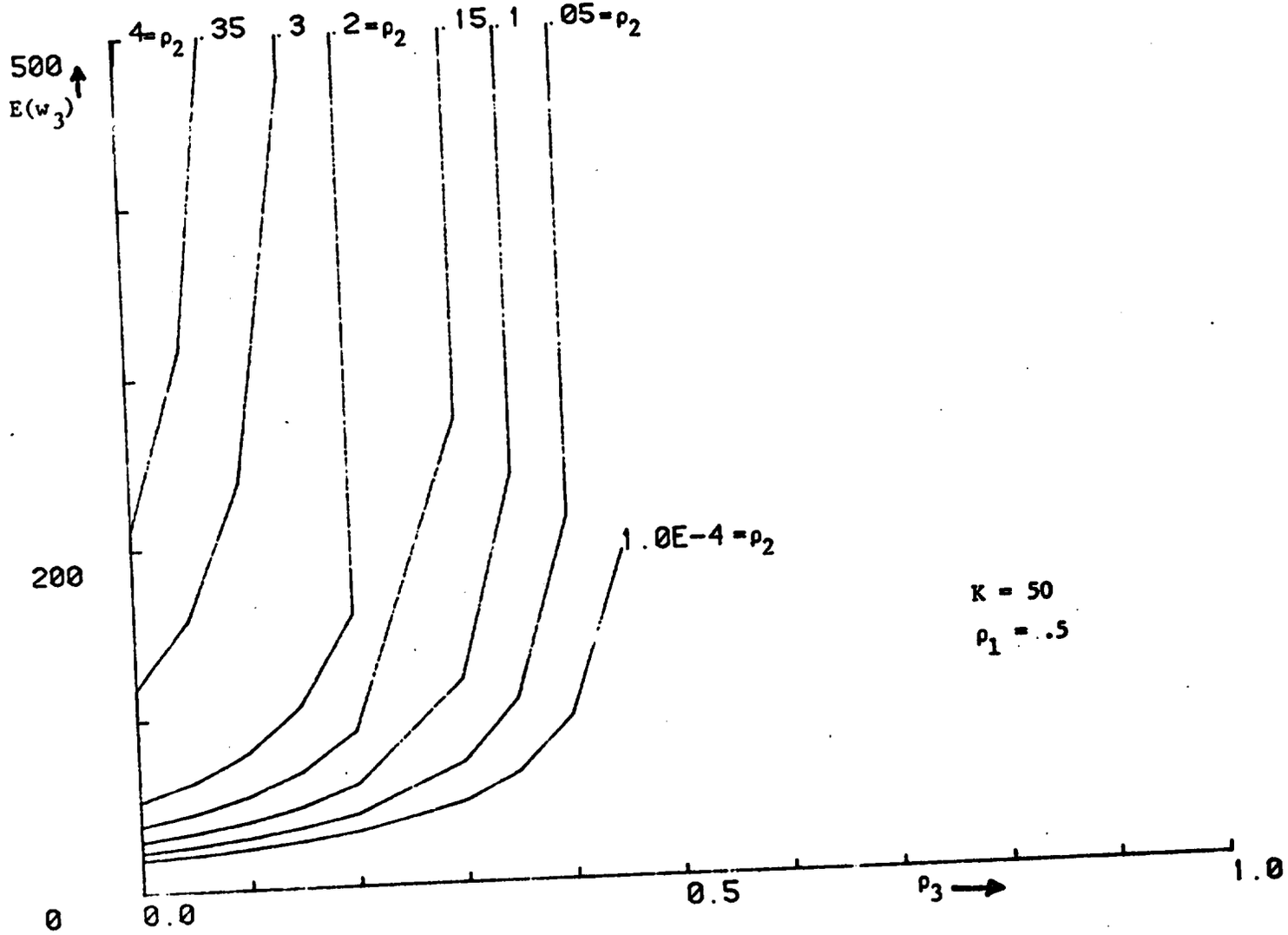


Fig. 6.28. Average Queue Size Vs. Utilization Factor At Queue 3 With ρ_1 and ρ_2 As Parameters.

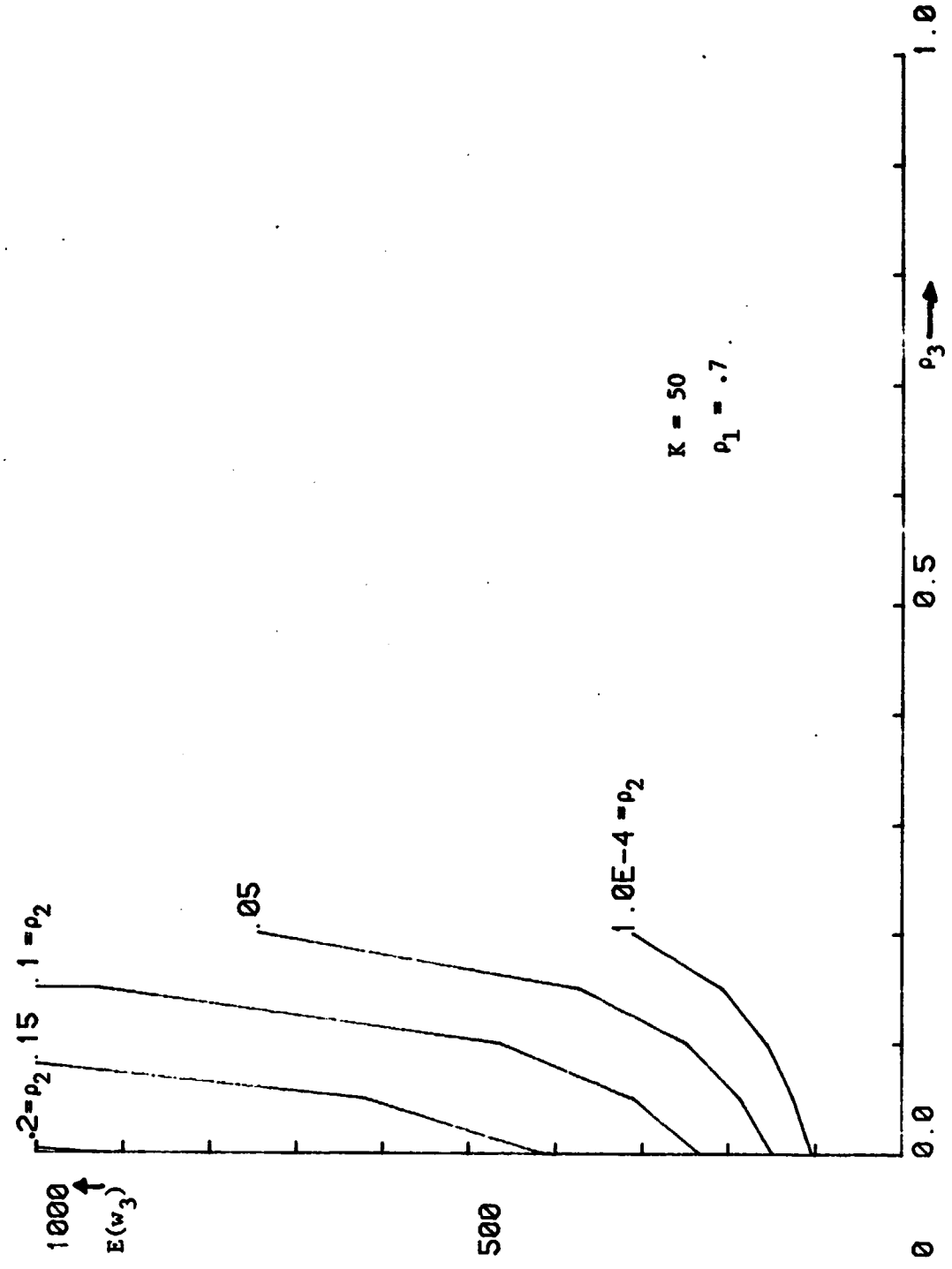
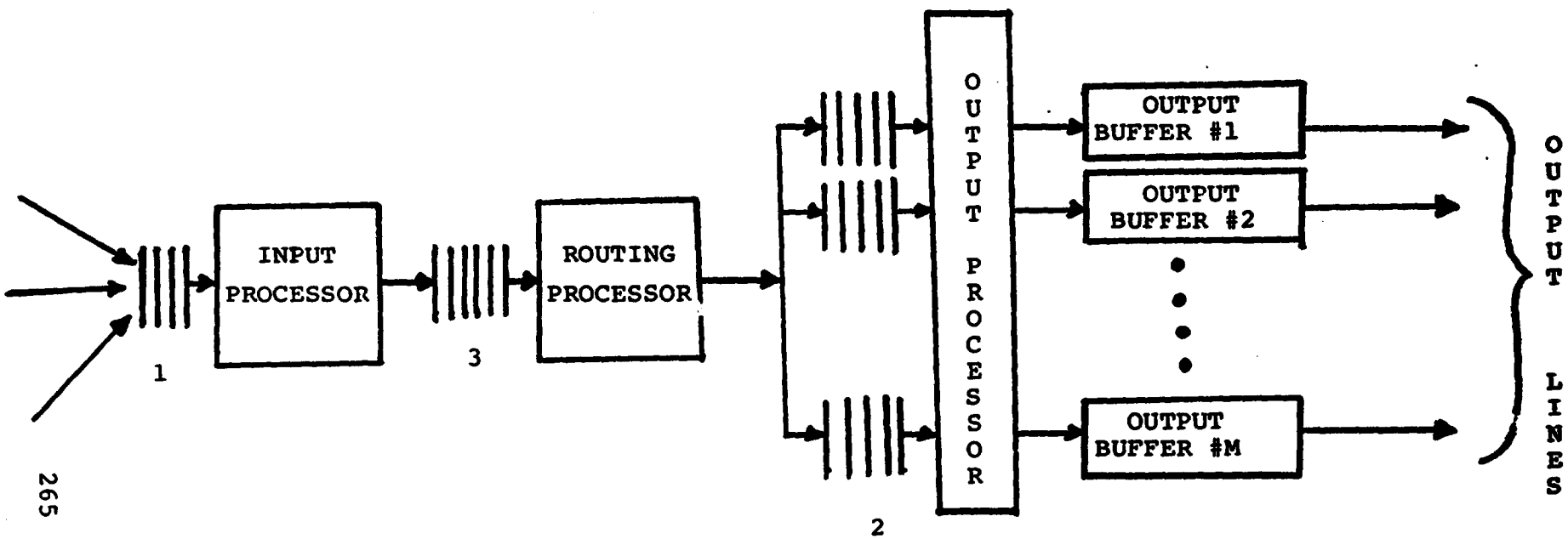


Fig. 6.29. Average Queue Size Vs. Utilization Factor At Queue 3 With ρ_1 and ρ_2 As Parameters.



LEGEND:

1 Input Queue

2 Output Queue

3 Routing Queue

Fig. 6.30. The Queueing Model for the Three Processor Design

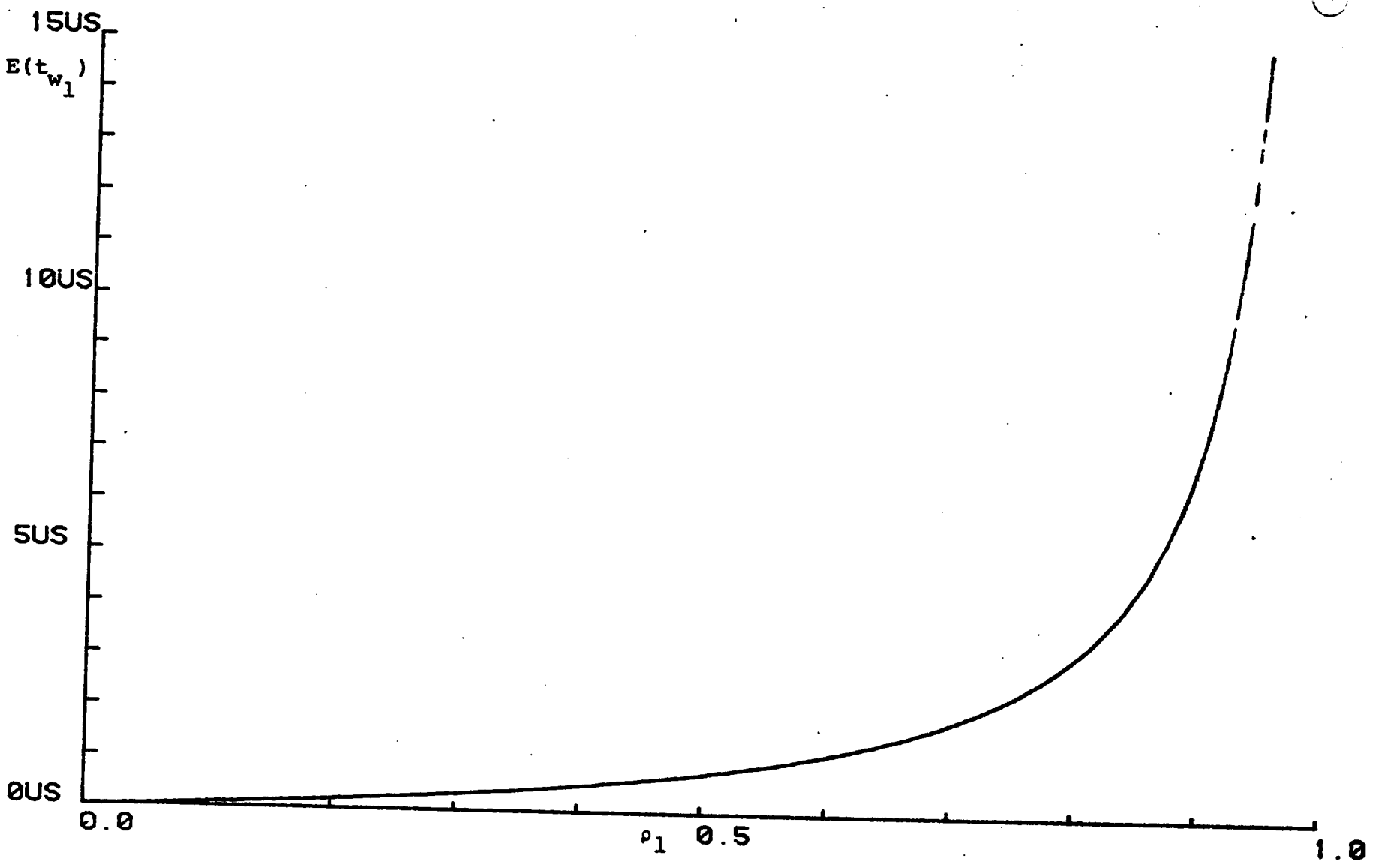


Fig. 6.31. Average Waiting Time Vs. Utilization Factor At The Input Queue.

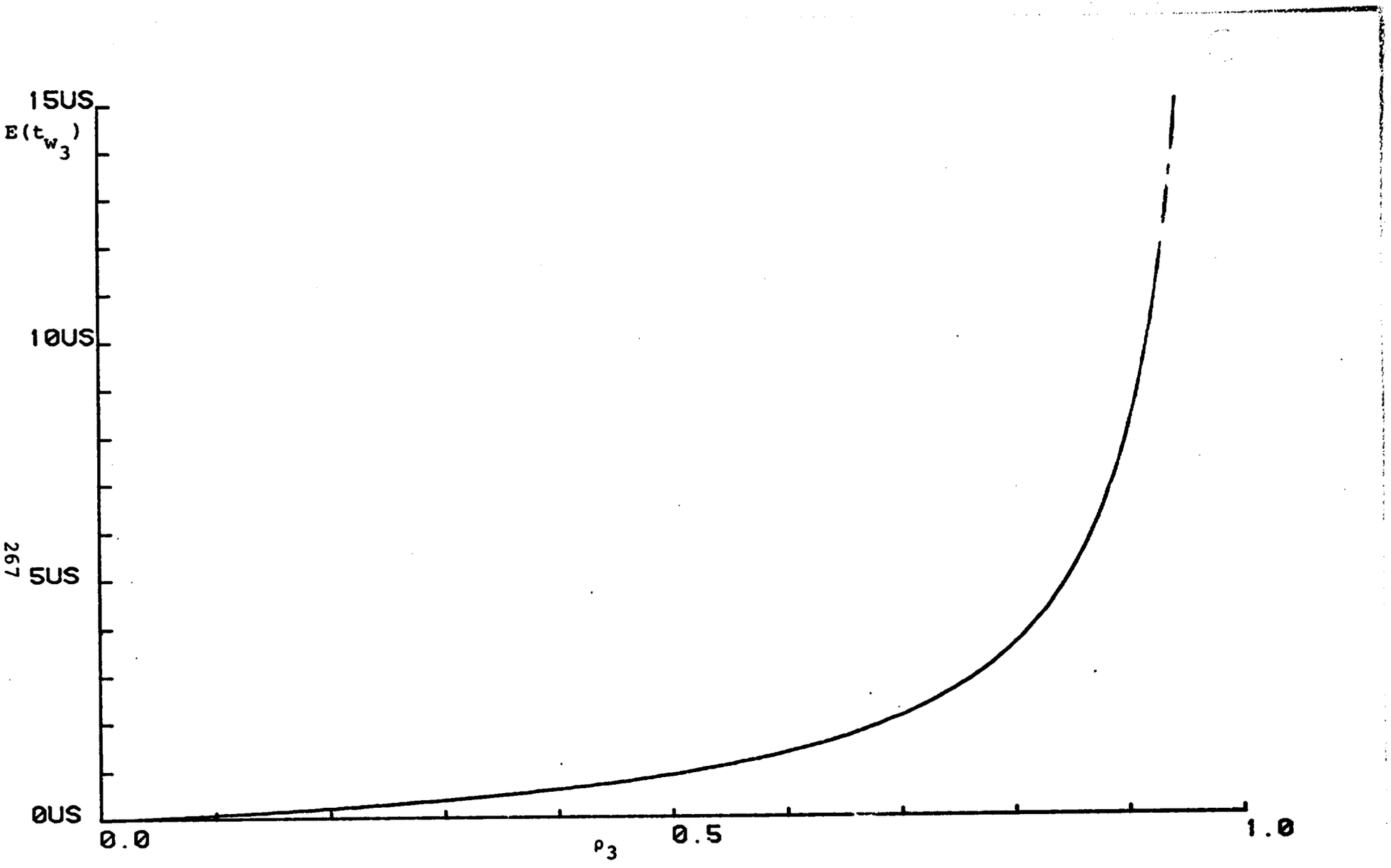


Fig. 6.32. Average Waiting Time Vs. Utilization Factor At The Routing Queue. (No contention).

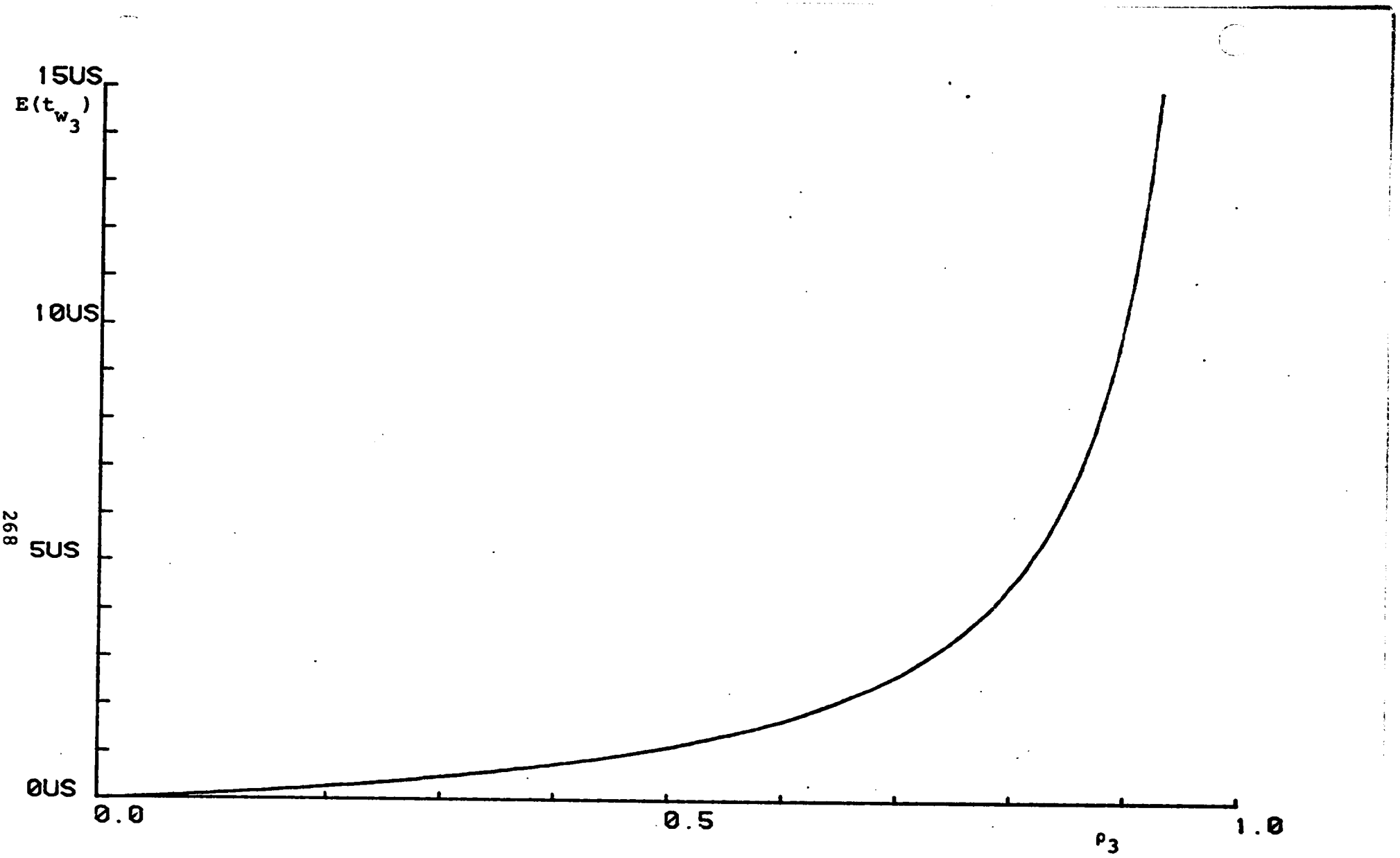


Fig. 6.33. Average Waiting Time Vs. Utilization Factor At The Routing Queue. (Contention at all times).

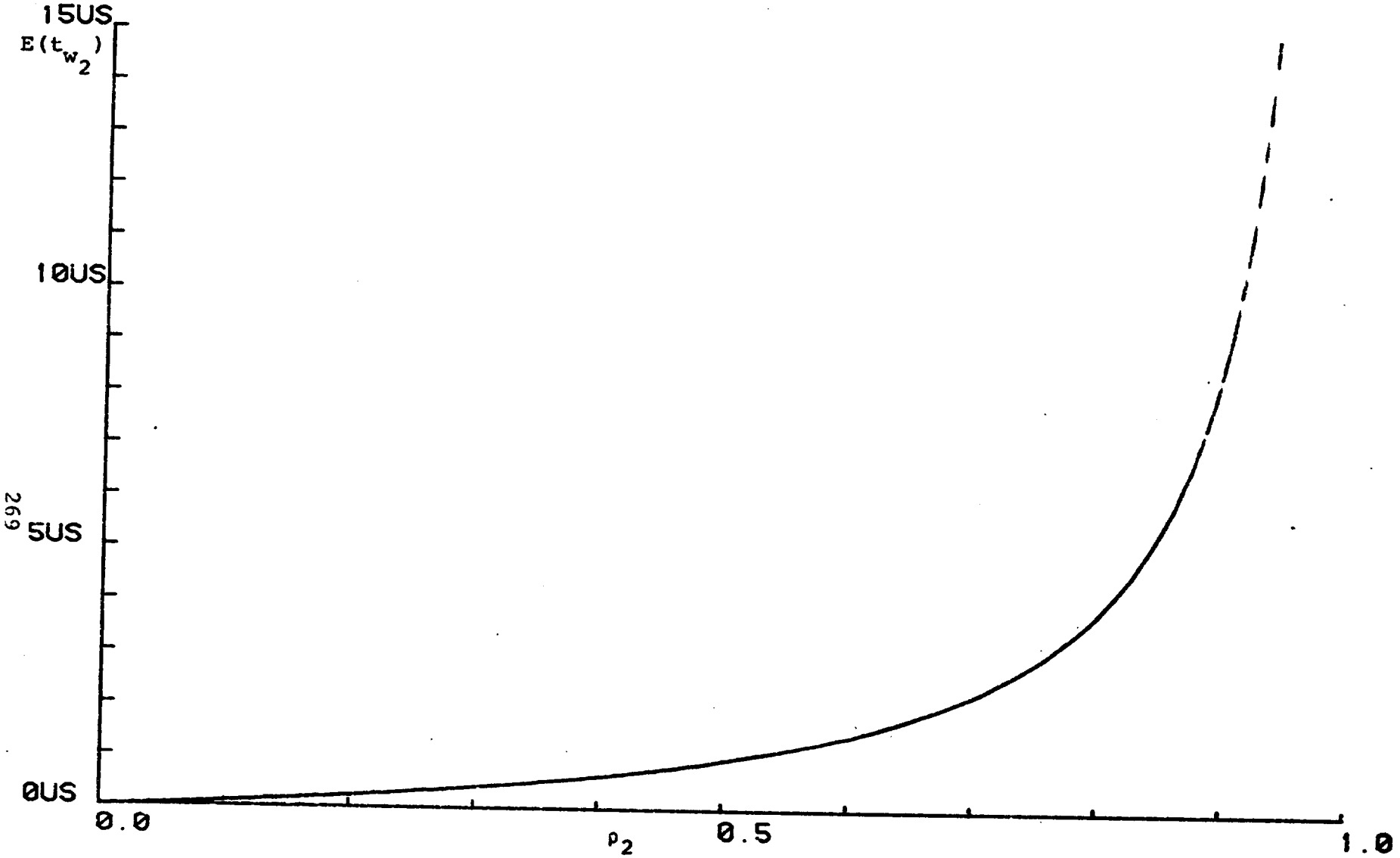


Fig. 6.34. Average Waiting Time Vs. Utilization Factor At The Output Queue. (No contention).

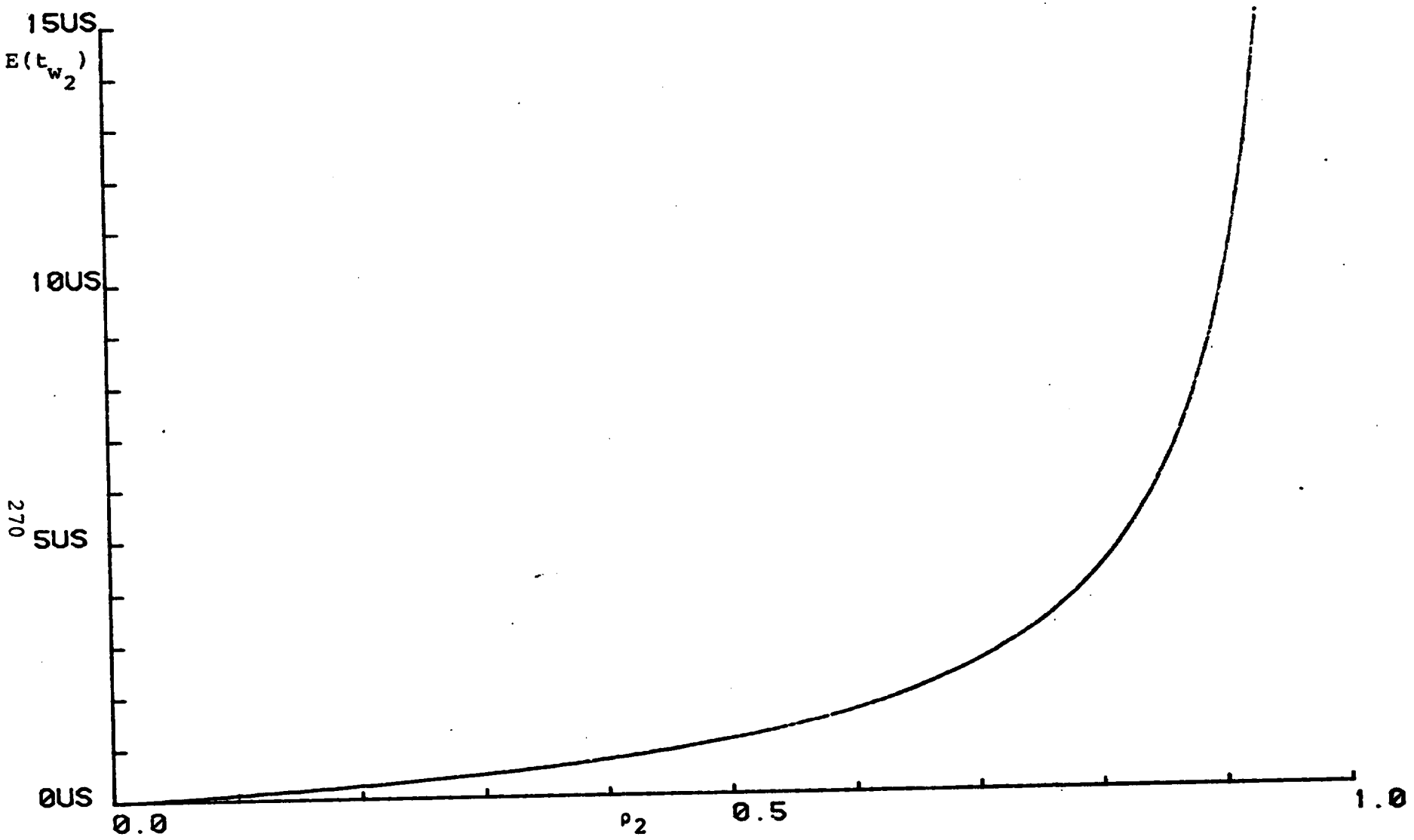


Fig. 6.35. Average Waiting Time Vs. Utilization Factor At The Output Queue. (Contention at all times).

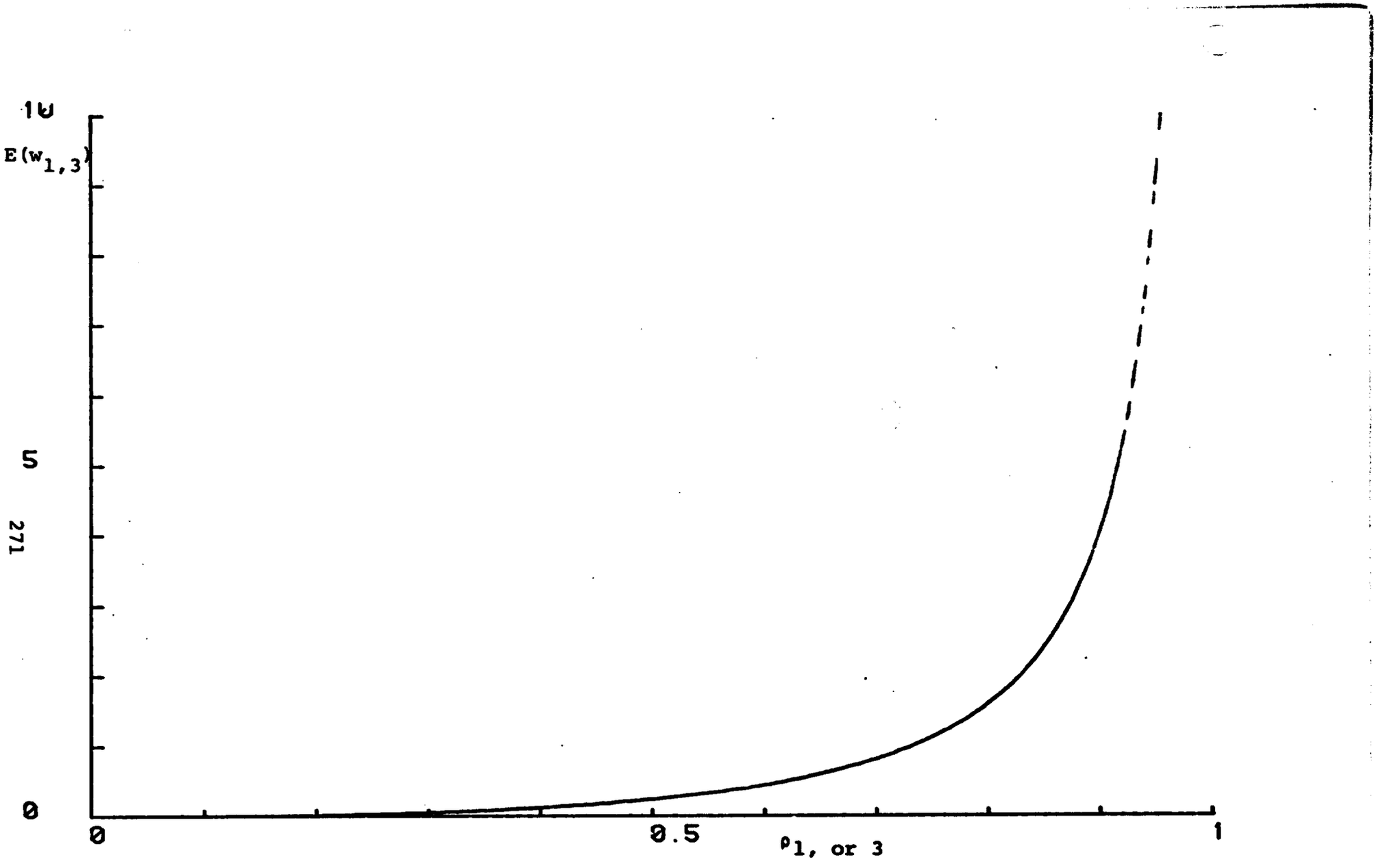


Fig. 6.36. Average Queue Size Vs. Utilization Factor At The Input And The Routing Queues. (No Contention).

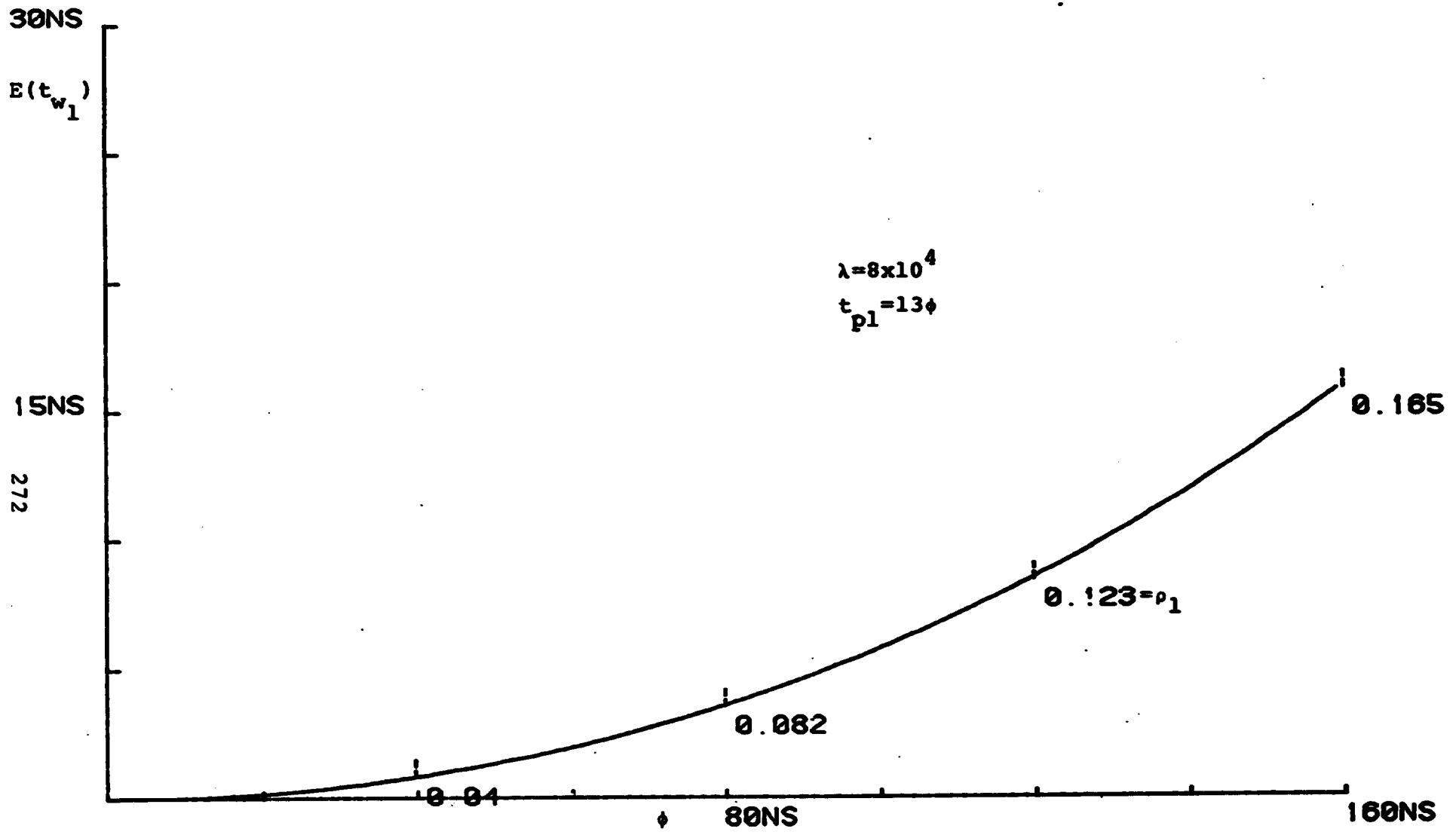


Fig. 6.37. Average Waiting Time Vs. Clock Cycle Time Of The Processor At The Input Queue. (For The Proposed Design ϕ=120ns).

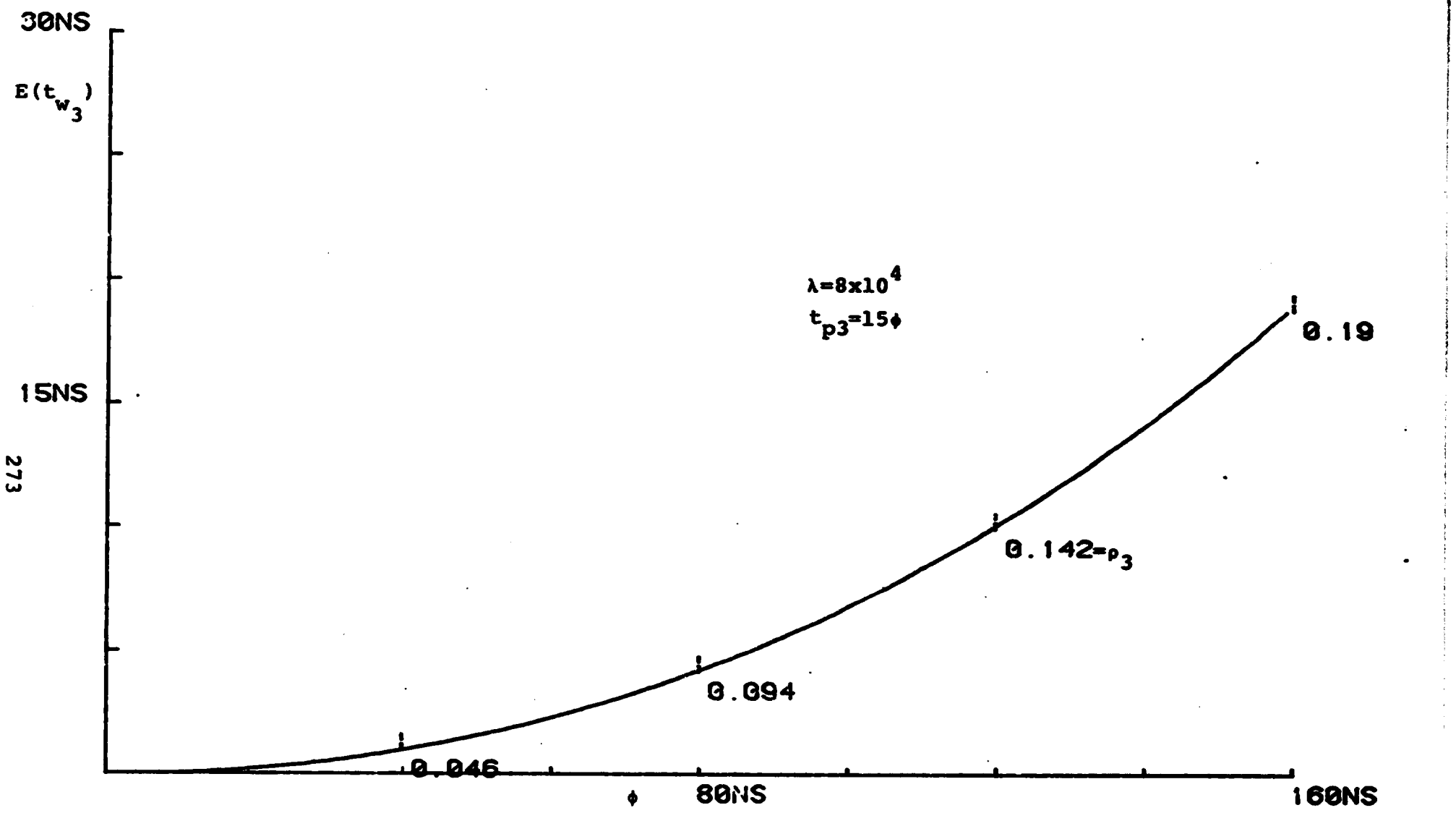


Fig. 6.38. Average Waiting Time Vs. Clock Cycle Time Of The Processor At The Routing Queue. (No Contention For The Proposed Design ϕ=120ns).

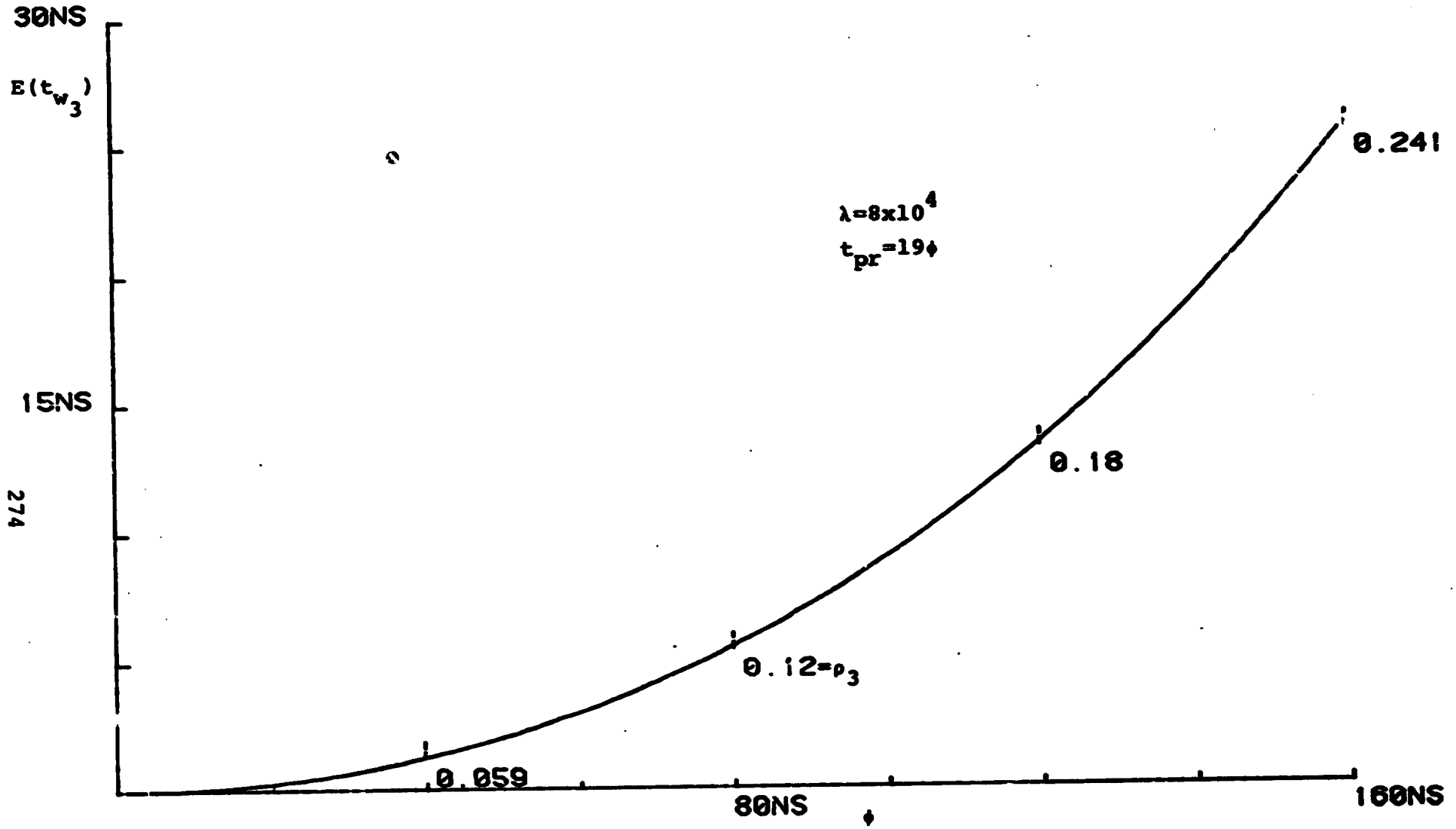


Fig. 6.39. Average Waiting Time Vs. Clock Cycle Time Of The Processor At The Routing Queue. (Contention At All Times For The Proposed Design ϕ=120ns).

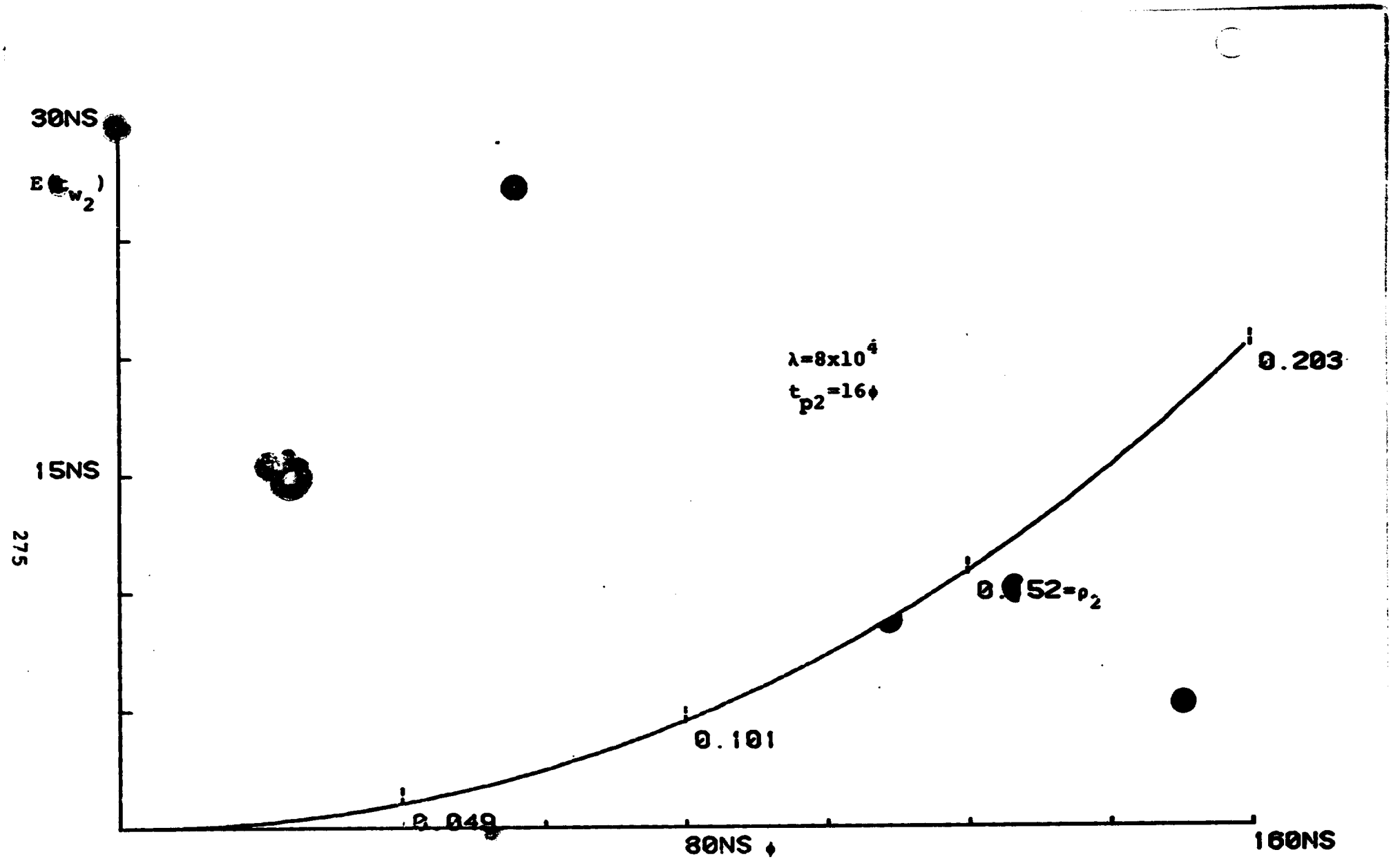


Fig. 6.40. Average Waiting Time Vs. Clock Cycle Time Of The Processor At The Output Queue. (No Contention For the Proposed Design ϕ=120ns).

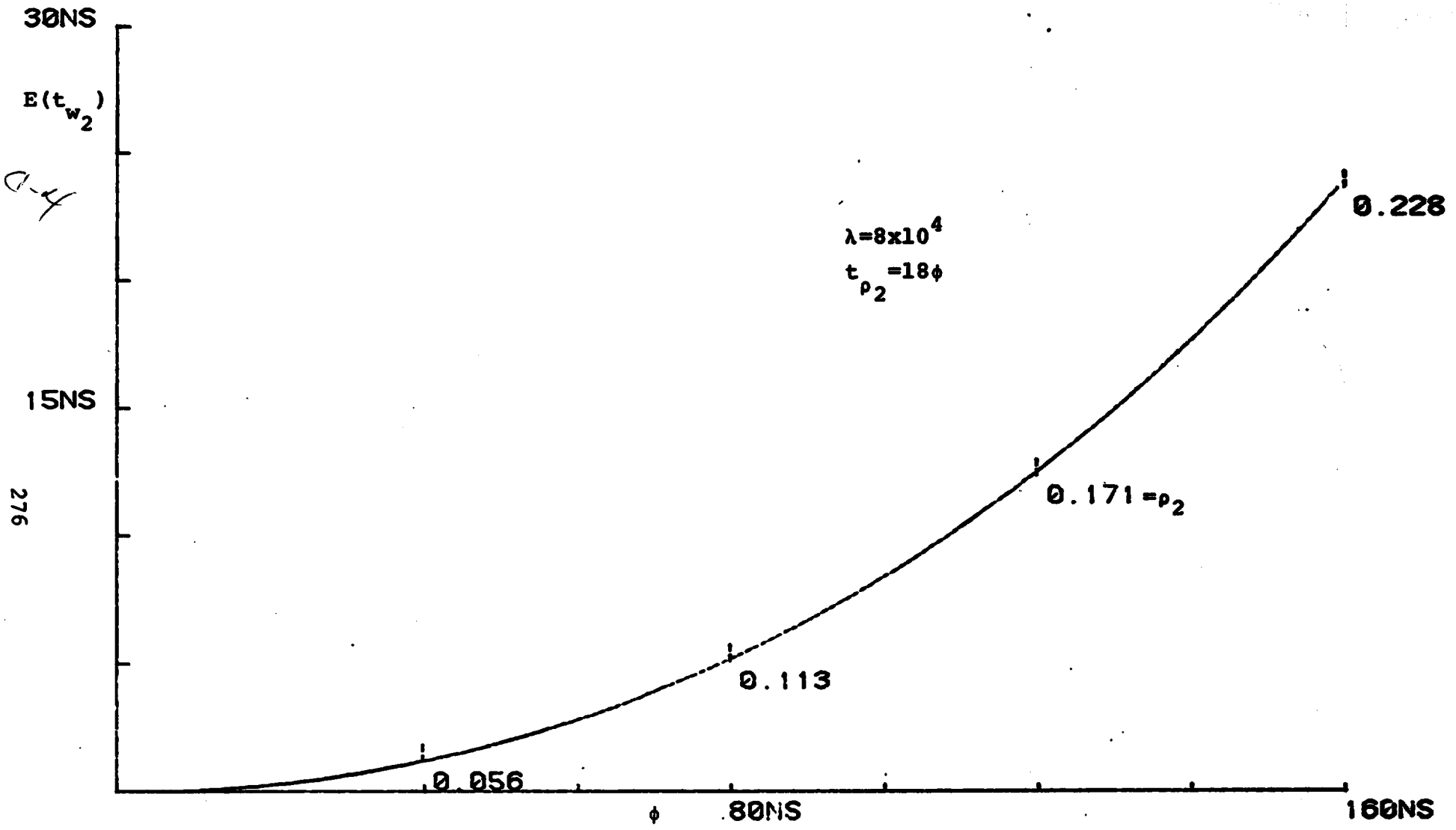


Fig. 6.41. Average Waiting Time Vs. Clock Cycle Time Of The Processor At The Output Queue. (Contention At All Times, For The Proposed Design ϕ=120ns).

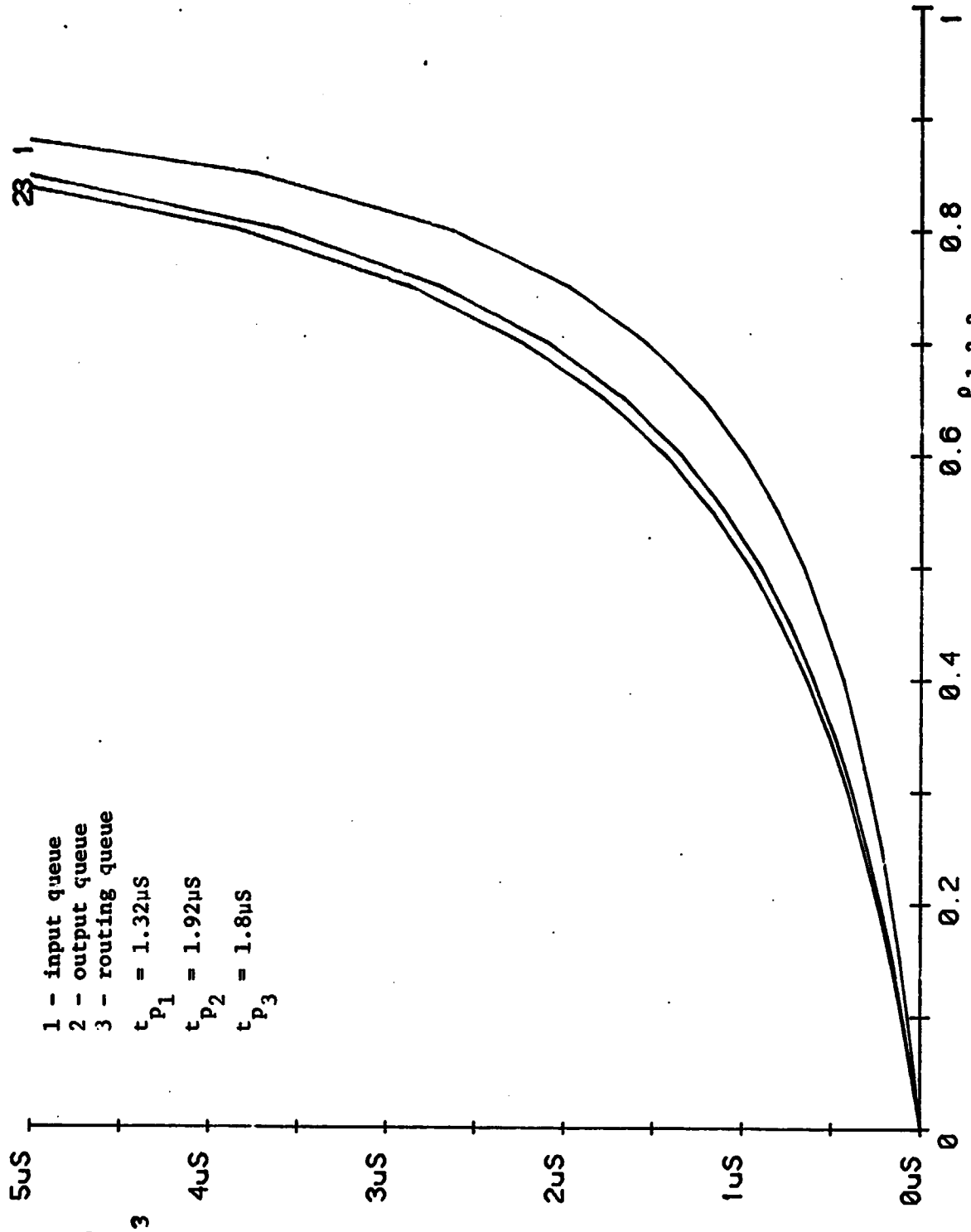


Fig. 6.42. Average Waiting Time Vs. Utilization Factor At the Input, Output and Routing Queues.

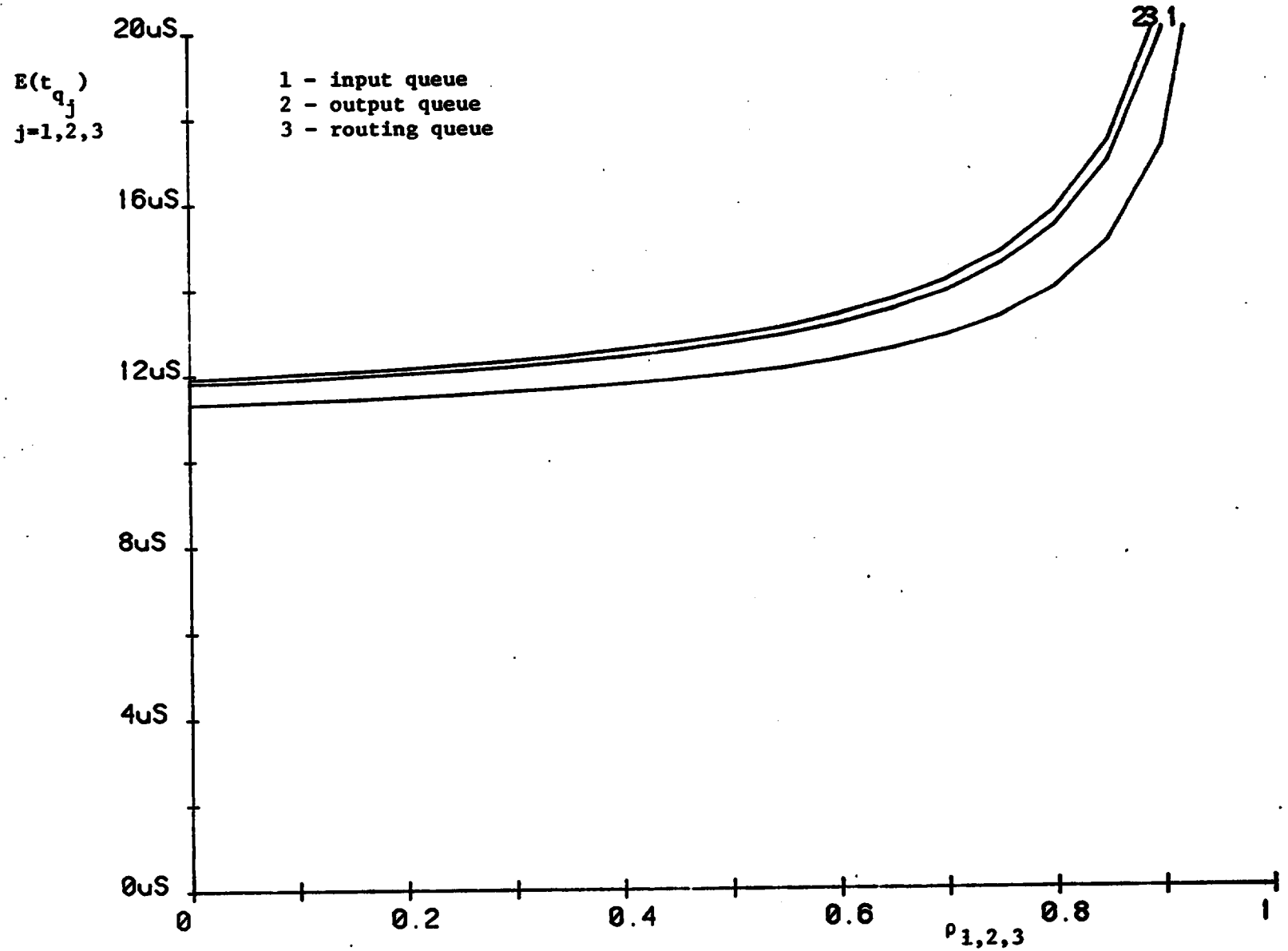


Fig. 6.43. Overall Average Response Time Vs. Utilization Factors at the Input, Output and Routing Queues.

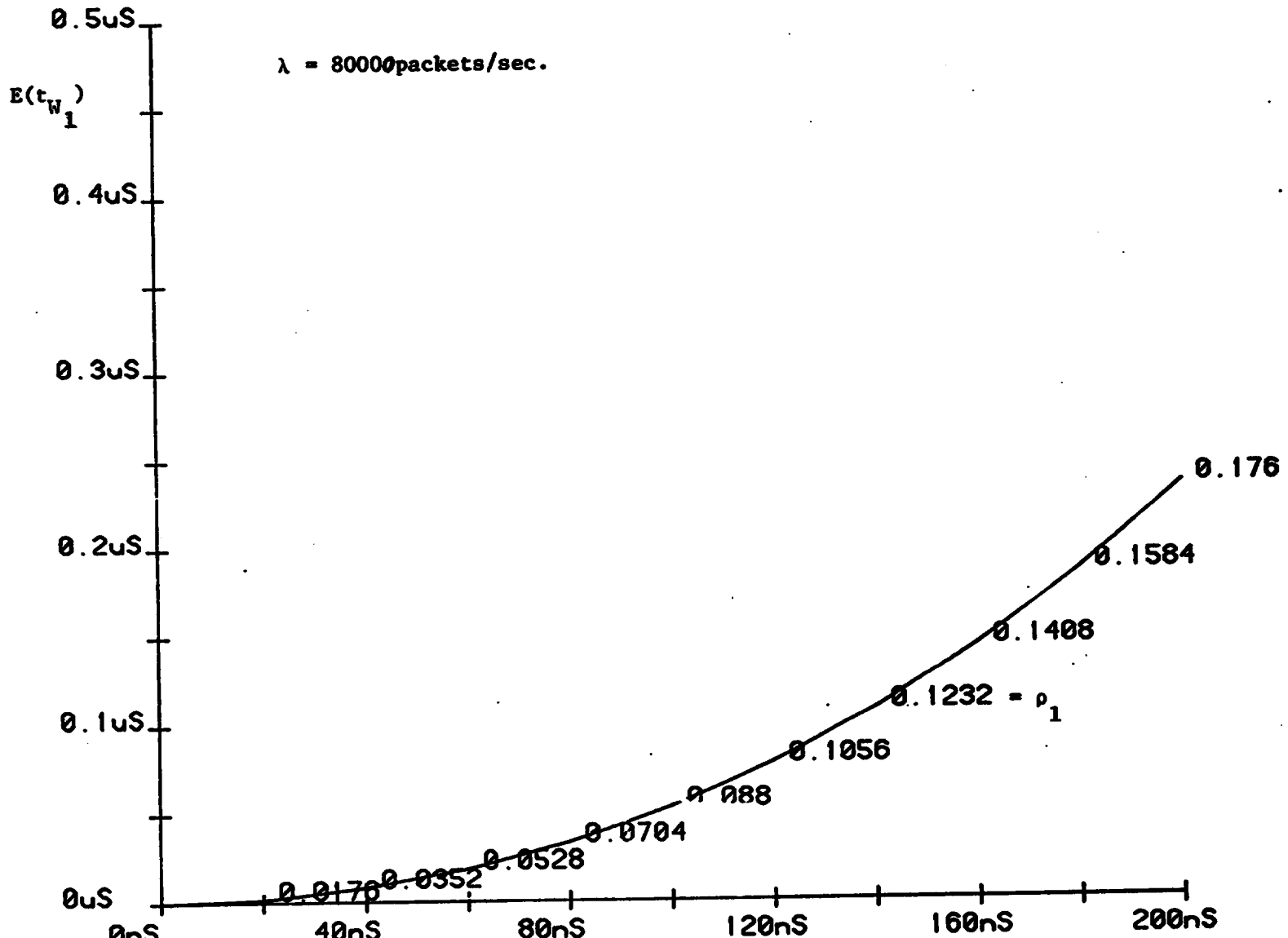


Fig. 6.44. Average Waiting Time Vs. Clock Cycle Time of the Processor at the Input Queue.

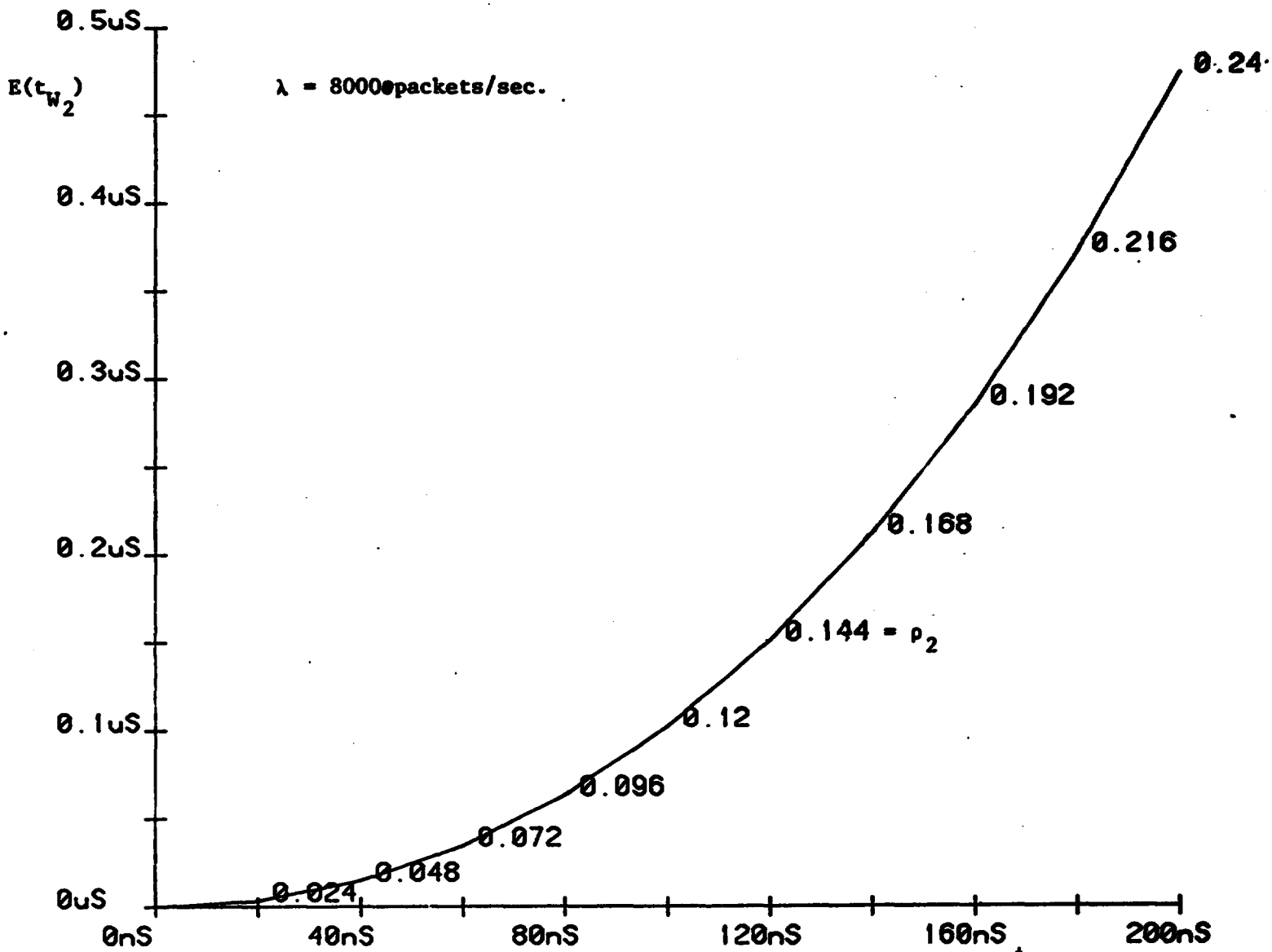


Fig. 6.45. Average Waiting Time Vs. Clock Cycle Time of the Processor at the Output Queue.

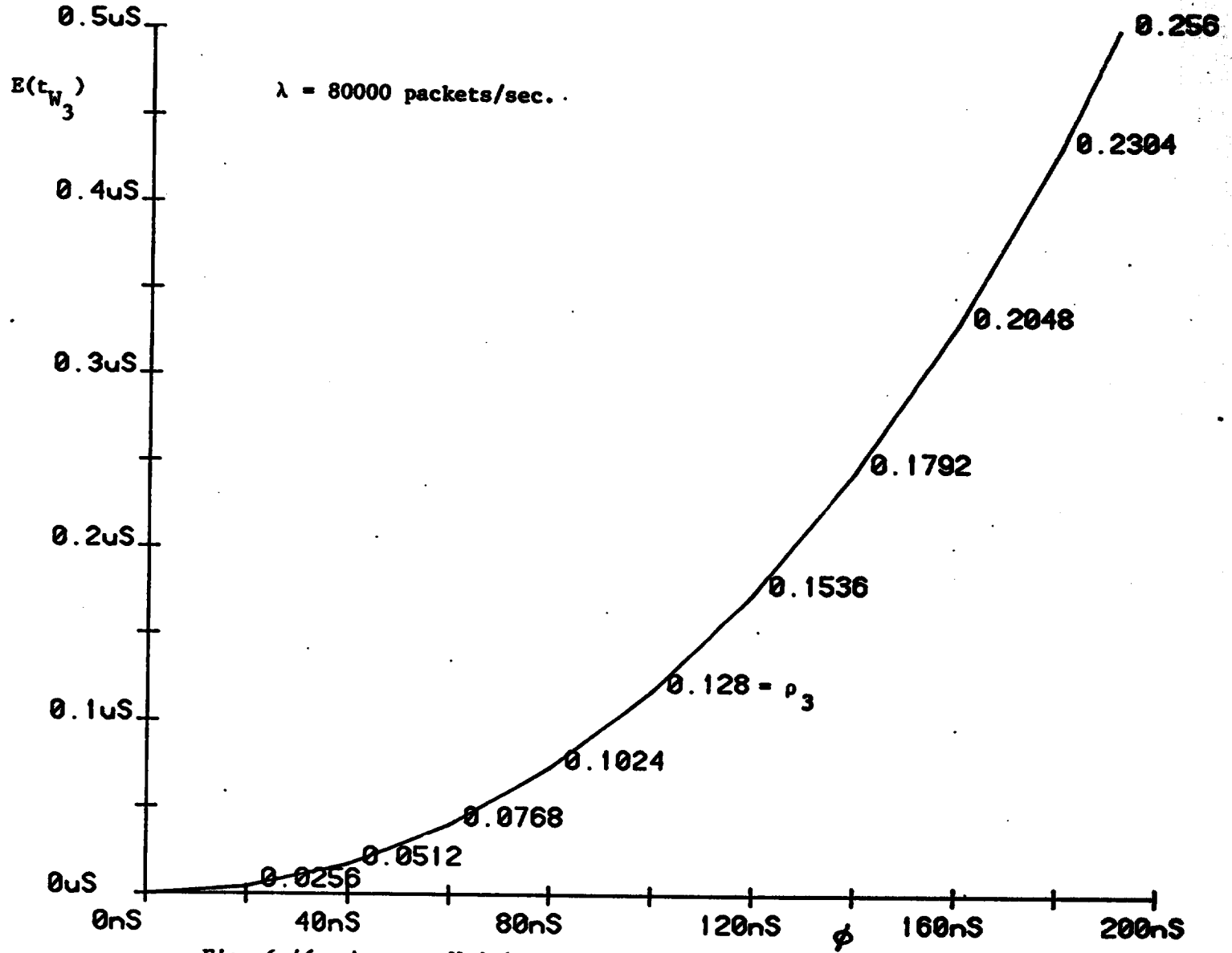


Fig. 6.46. Average Waiting Time Vs. Clock Cycle Time of the Processor at the Routing Queue.

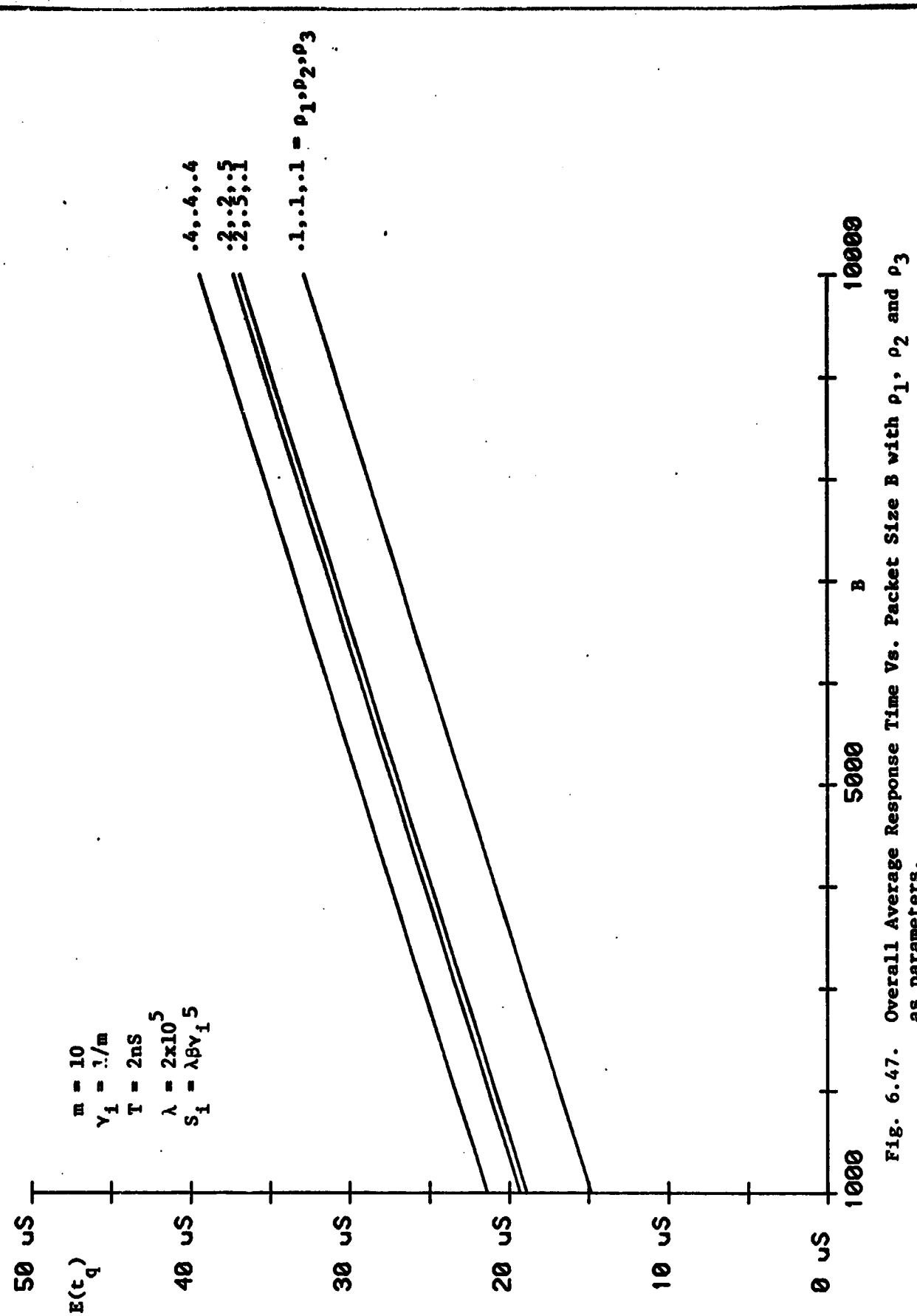
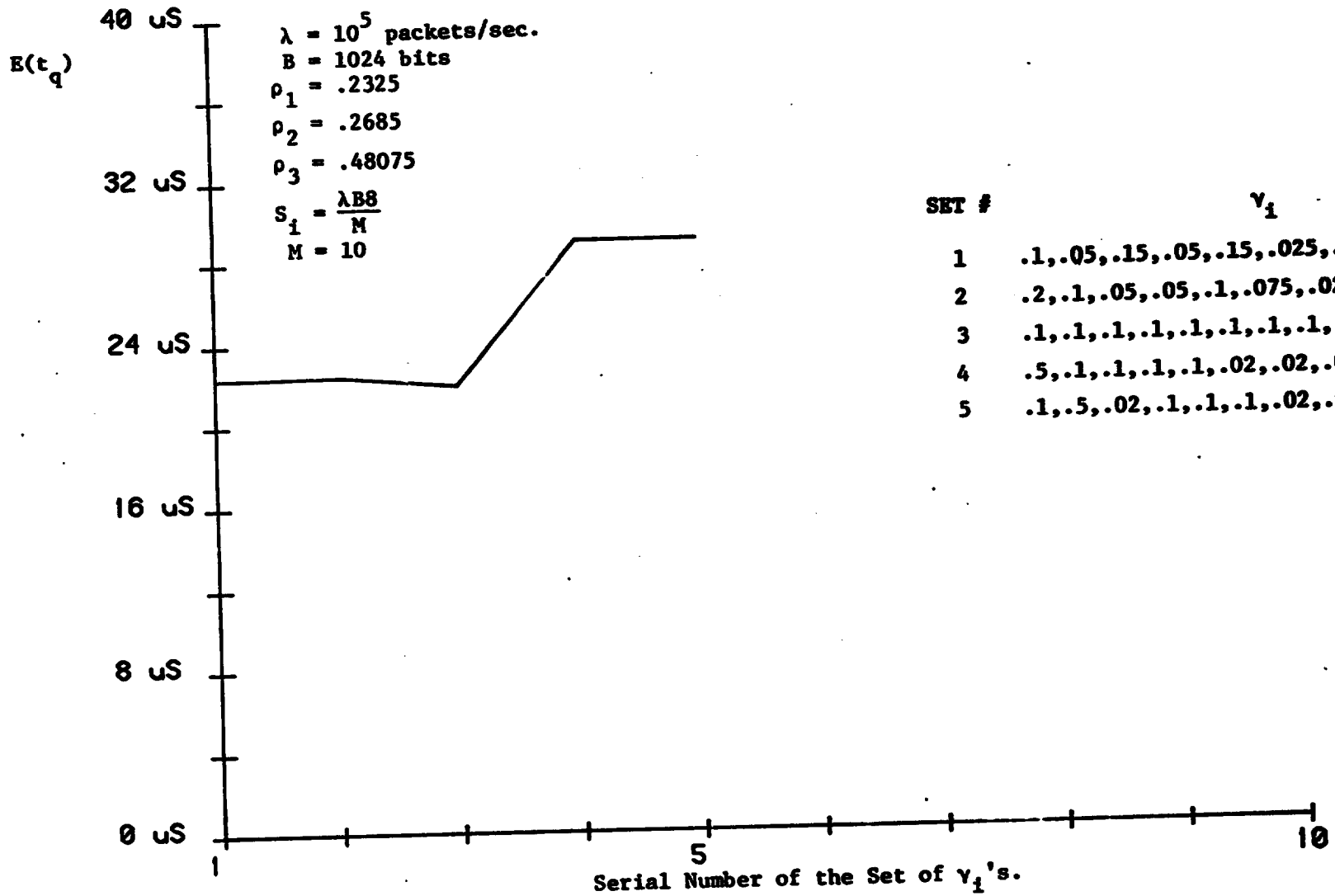


Fig. 6.47. Overall Average Response Time Vs. Packet Size B with ρ_1, ρ_2 and ρ_3 as parameters.



SET #	γ_i
1	.1,.05,.15,.05,.15,.025,.075,.2,.15,.05
2	.2,.1,.05,.05,.1,.075,.025,.2,.15,.05
3	.1,.1,.1,.1,.1,.1,.1,.1,.1,.1
4	.5,.1,.1,.1,.1,.02,.02,.02,.02,.02
5	.1,.5,.02,.1,.1,.1,.02,.02,.02,.02

Fig. 6.48. Overall Average Response Time Vs. Destination Functions.

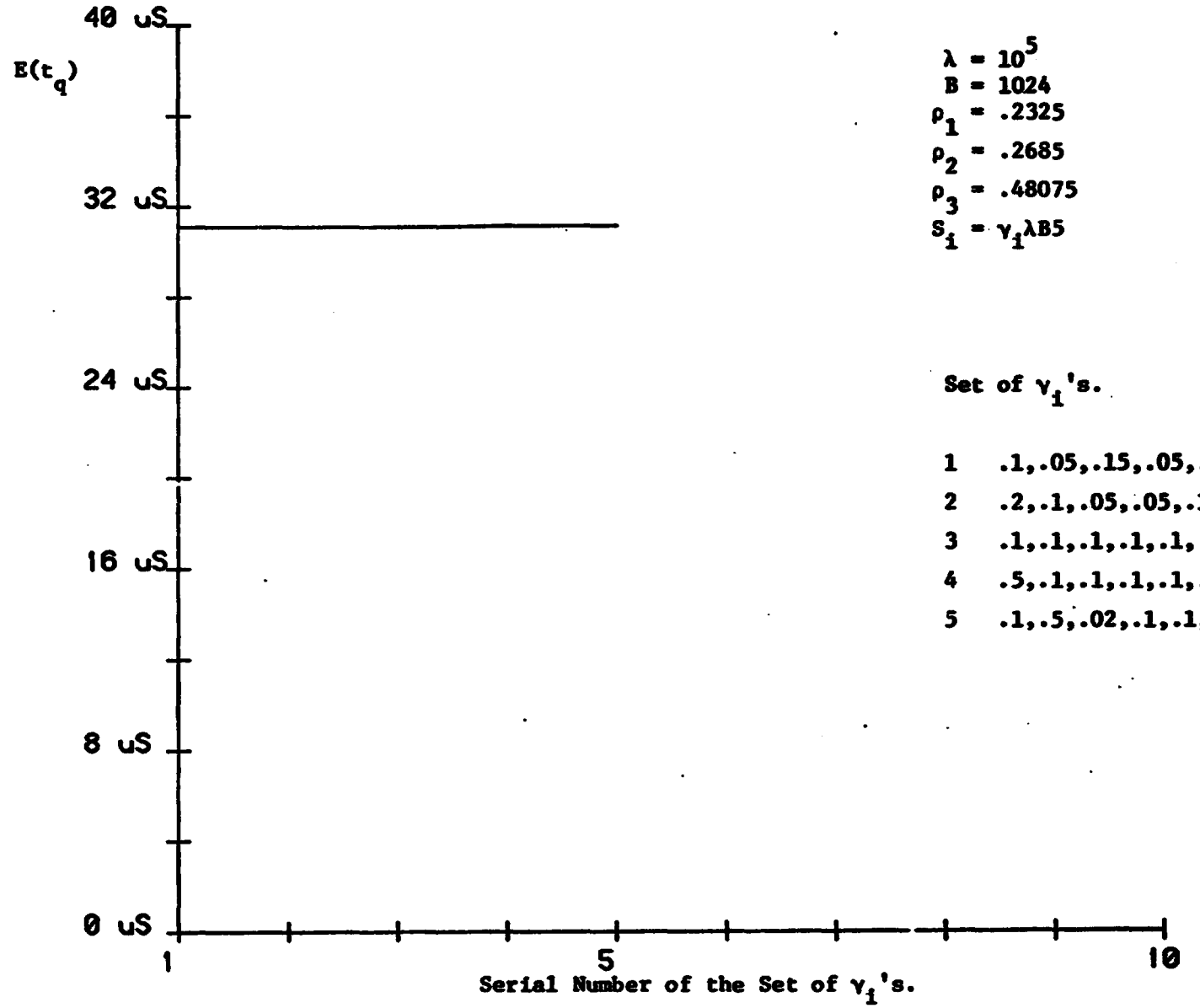


Fig. 6.49. Overall Average Response Time Vs. Destination Functions.

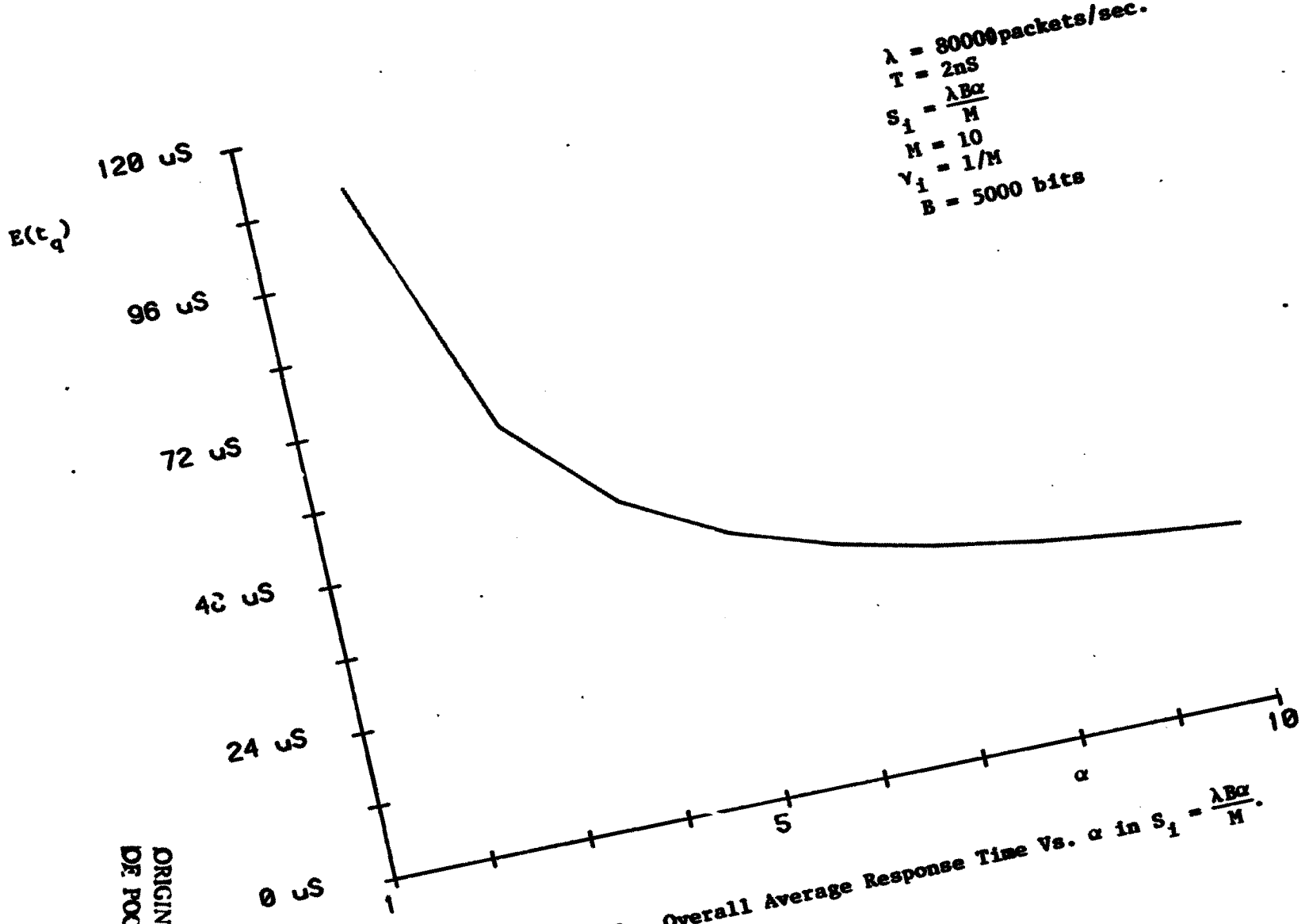


Fig. 6.50. Overall Average Response Time Vs. α in $S_i = \frac{\lambda B \alpha}{M}$.

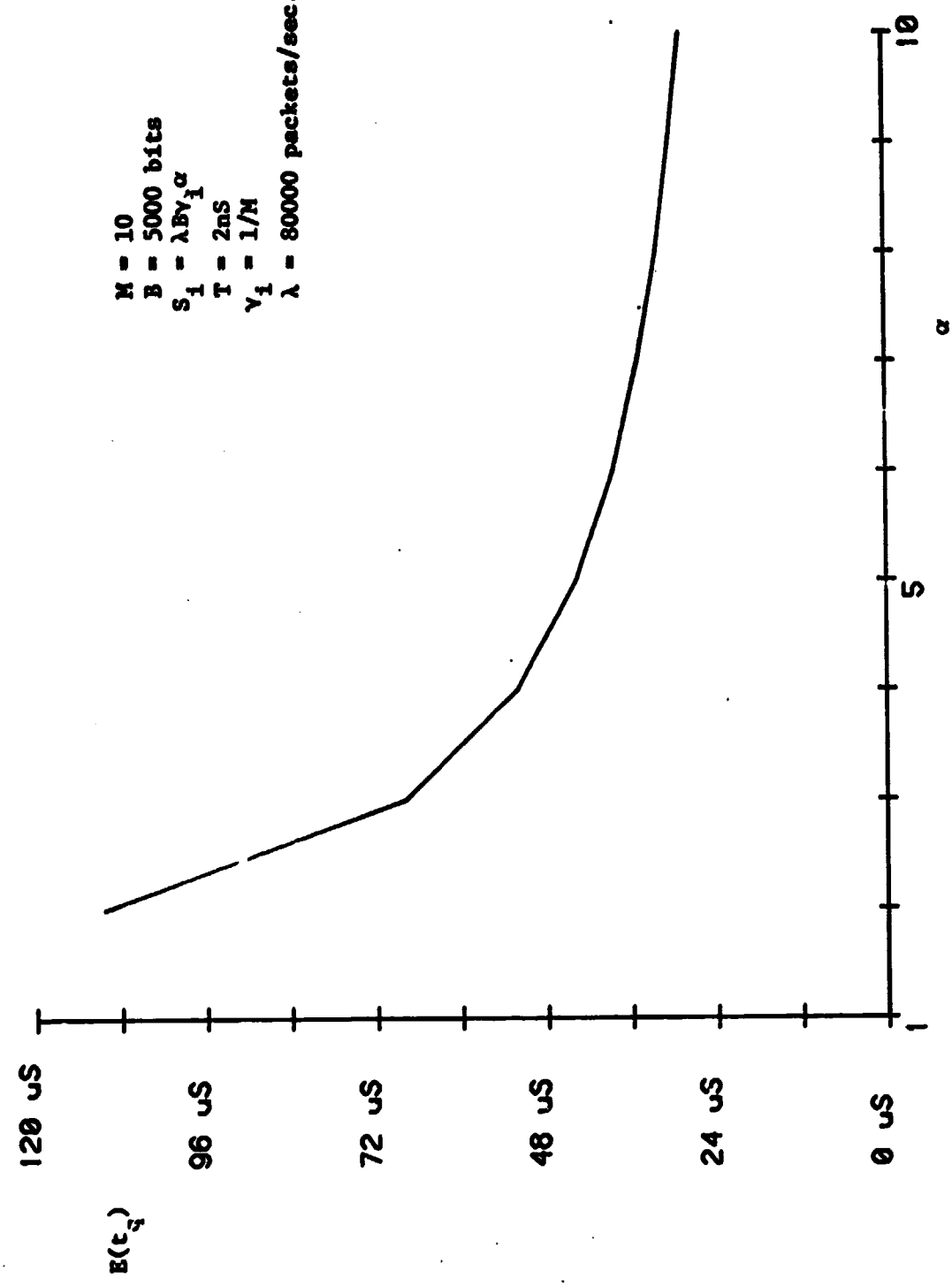


Fig. 6.51. Overall Average Response Time Vs. α in $S_i = \lambda B \gamma_i^{\alpha}$.

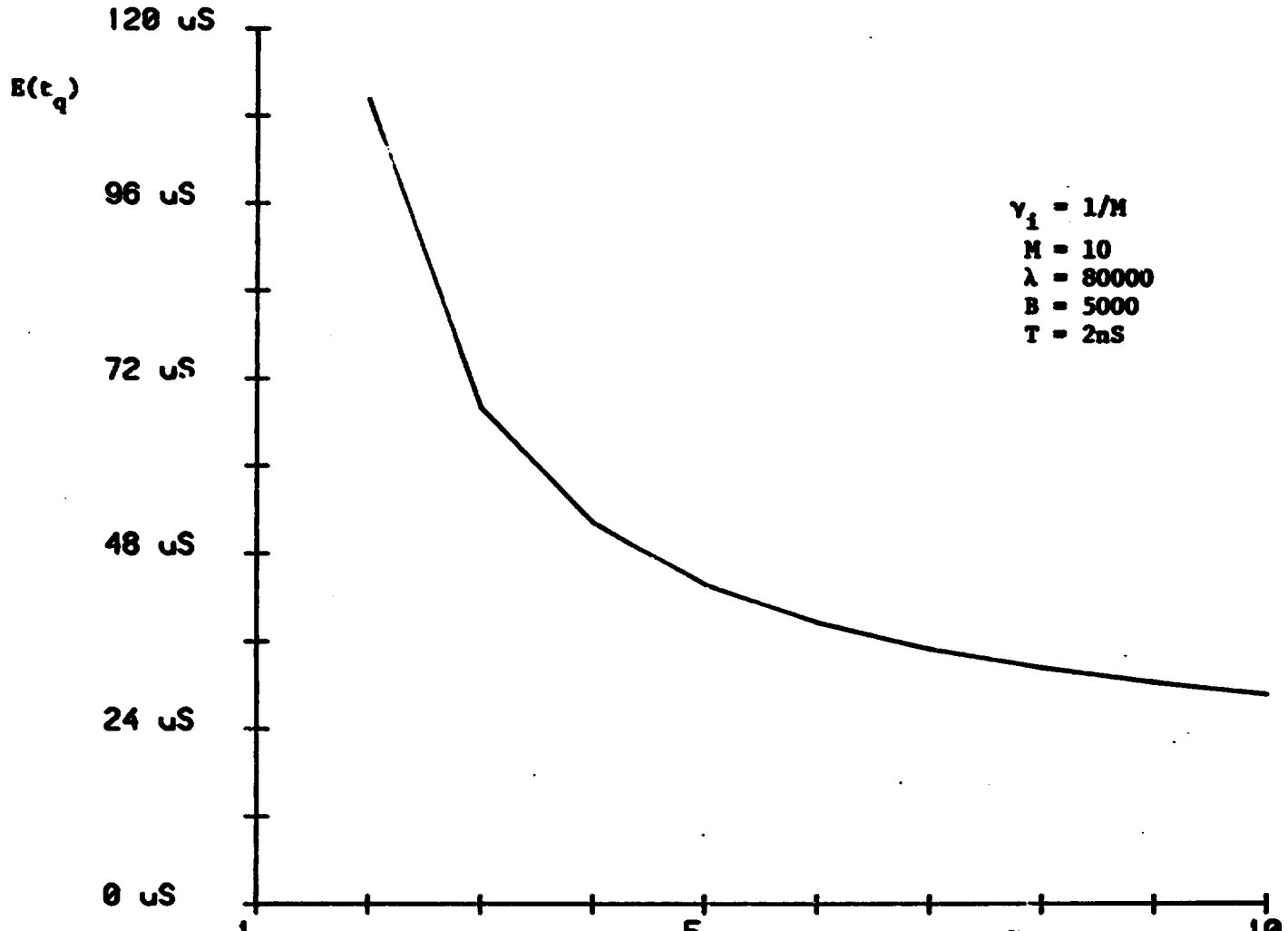


Fig. 6.52. Overall Average Response Time Vs. α in $S_i = \lambda B\gamma_i + \frac{\lambda B(\alpha-1)\sqrt{\lambda B\gamma_i}}{\sum_i \sqrt{\lambda B\gamma_i}}$

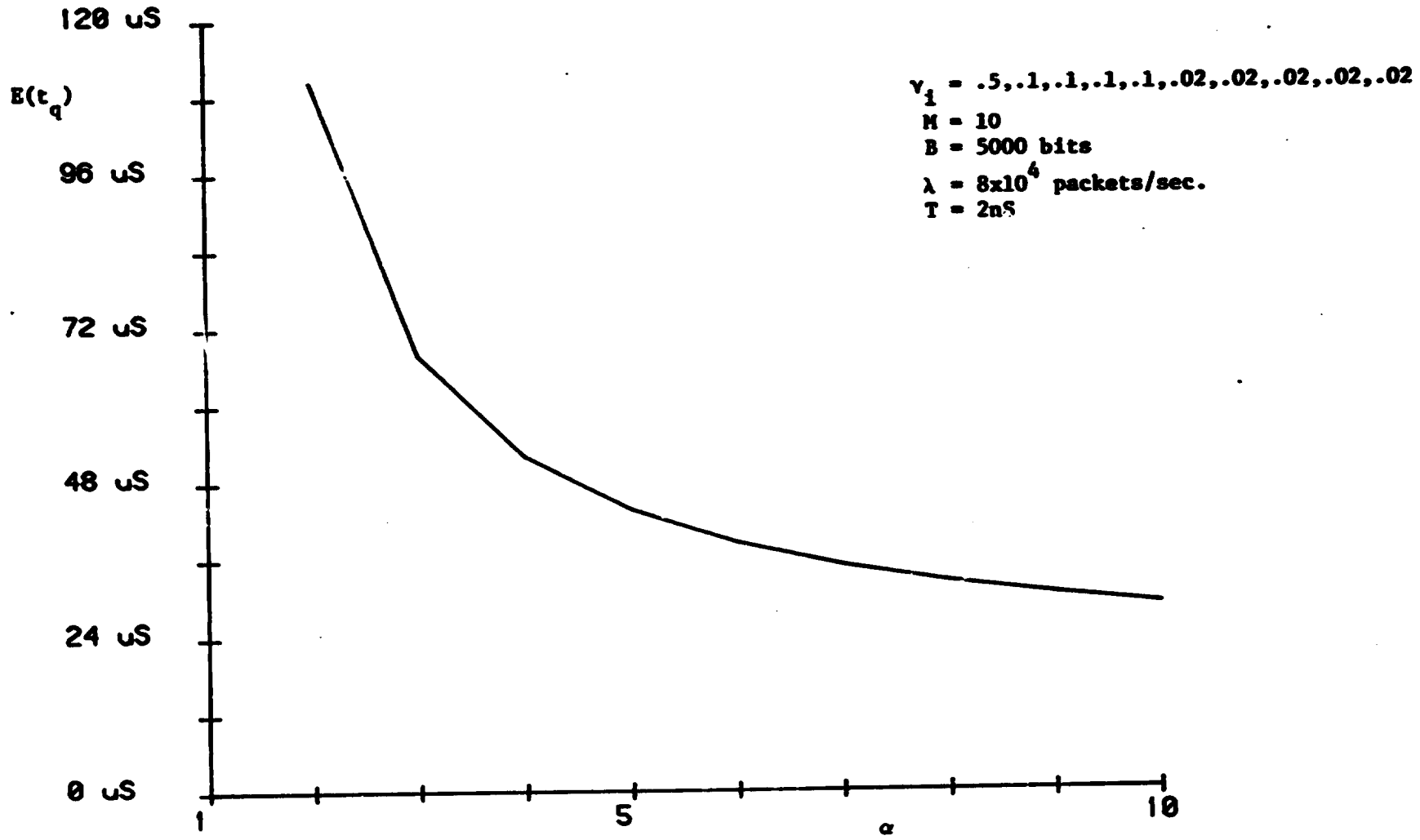


Fig. 5.53. Overall Average Response Time Vs. α in $S_i = \frac{\lambda B \alpha}{M}$.

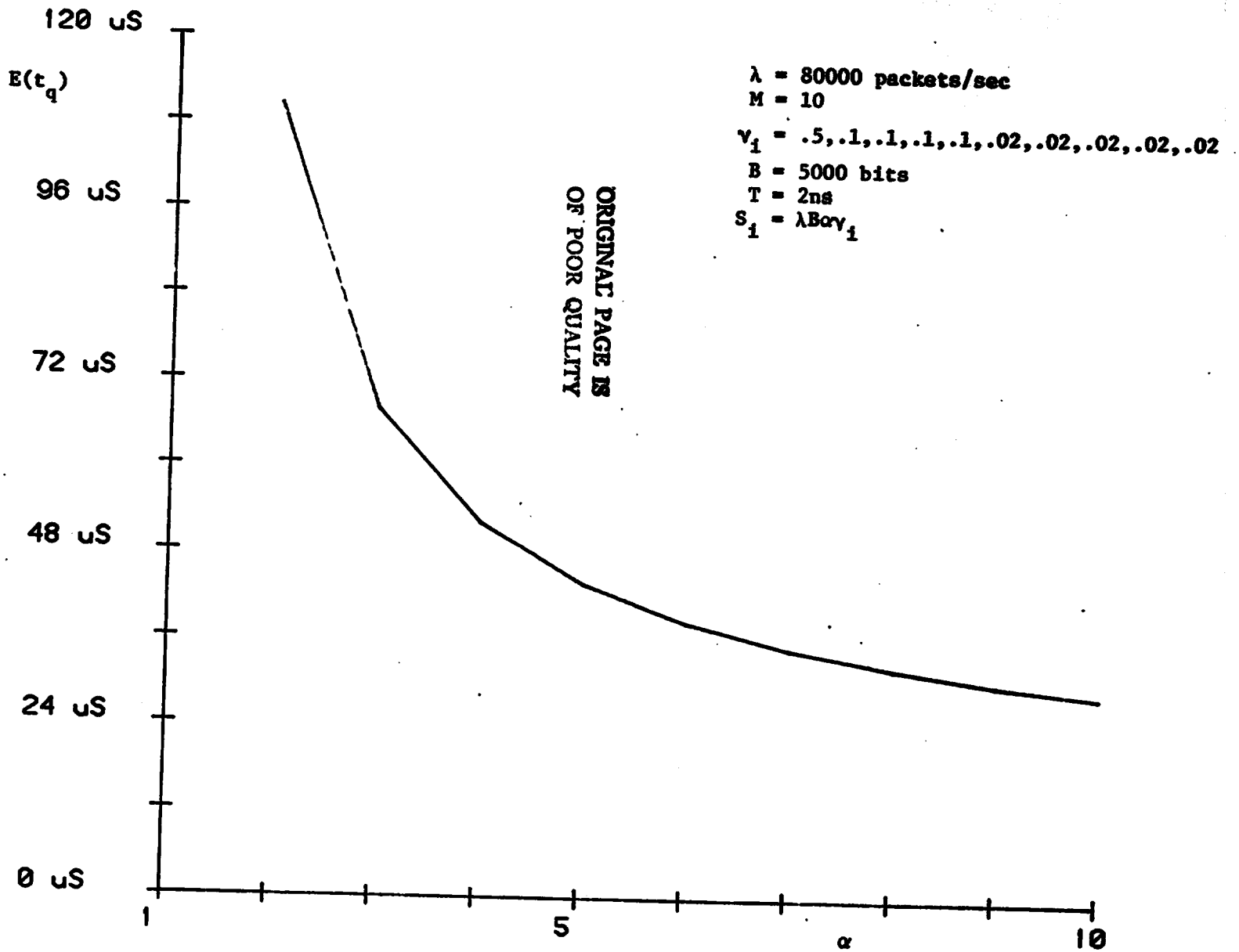


Fig. 6.54. Overall Average Response Time Vs. α in $S_i = \lambda B V_i \alpha$

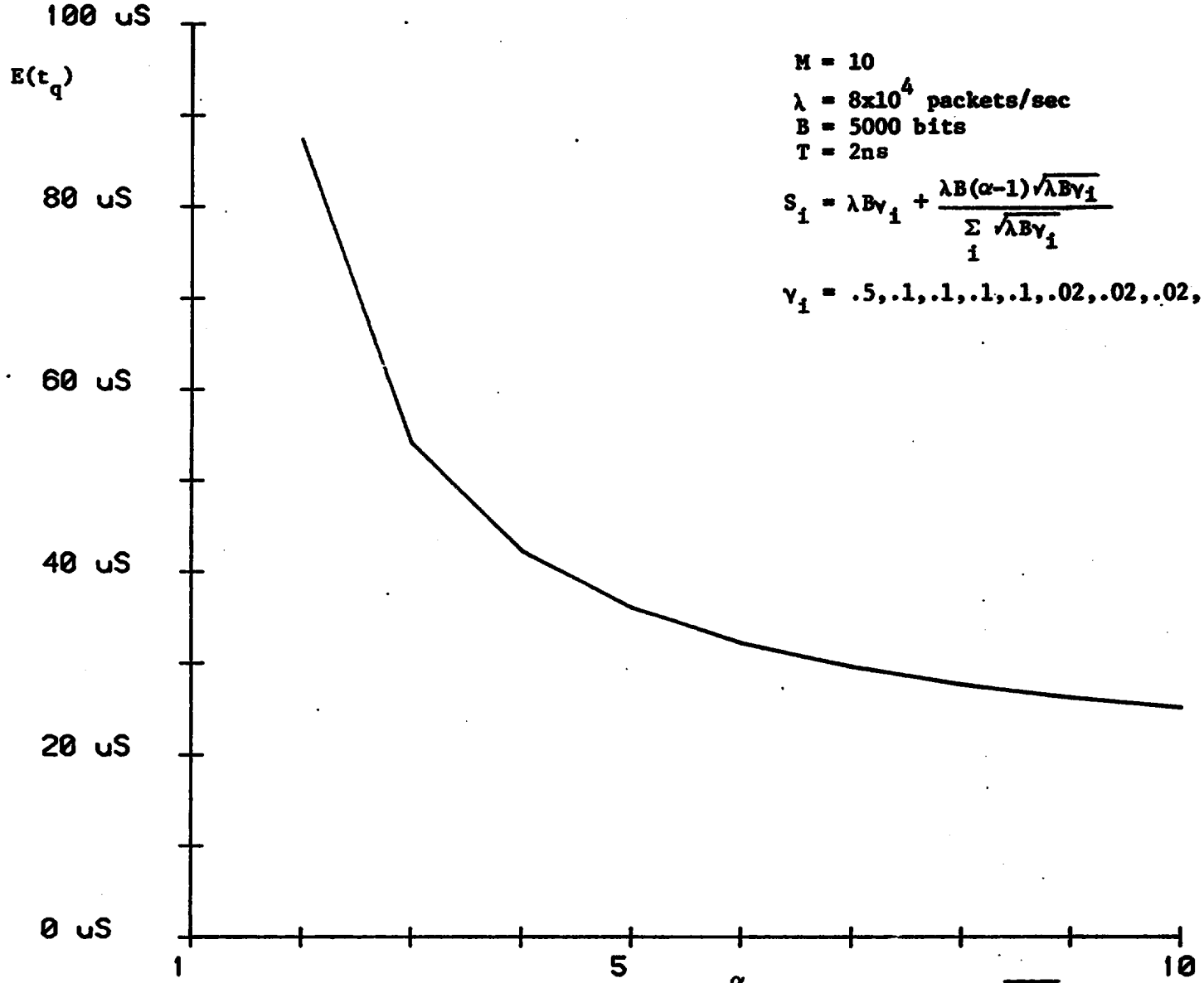


Fig. 6.55. Overall Average Response Time Vs. α in $\frac{\lambda B(\alpha-1)\sqrt{\lambda B\gamma_i}}{\sum_i \sqrt{\lambda B\gamma_i}}$

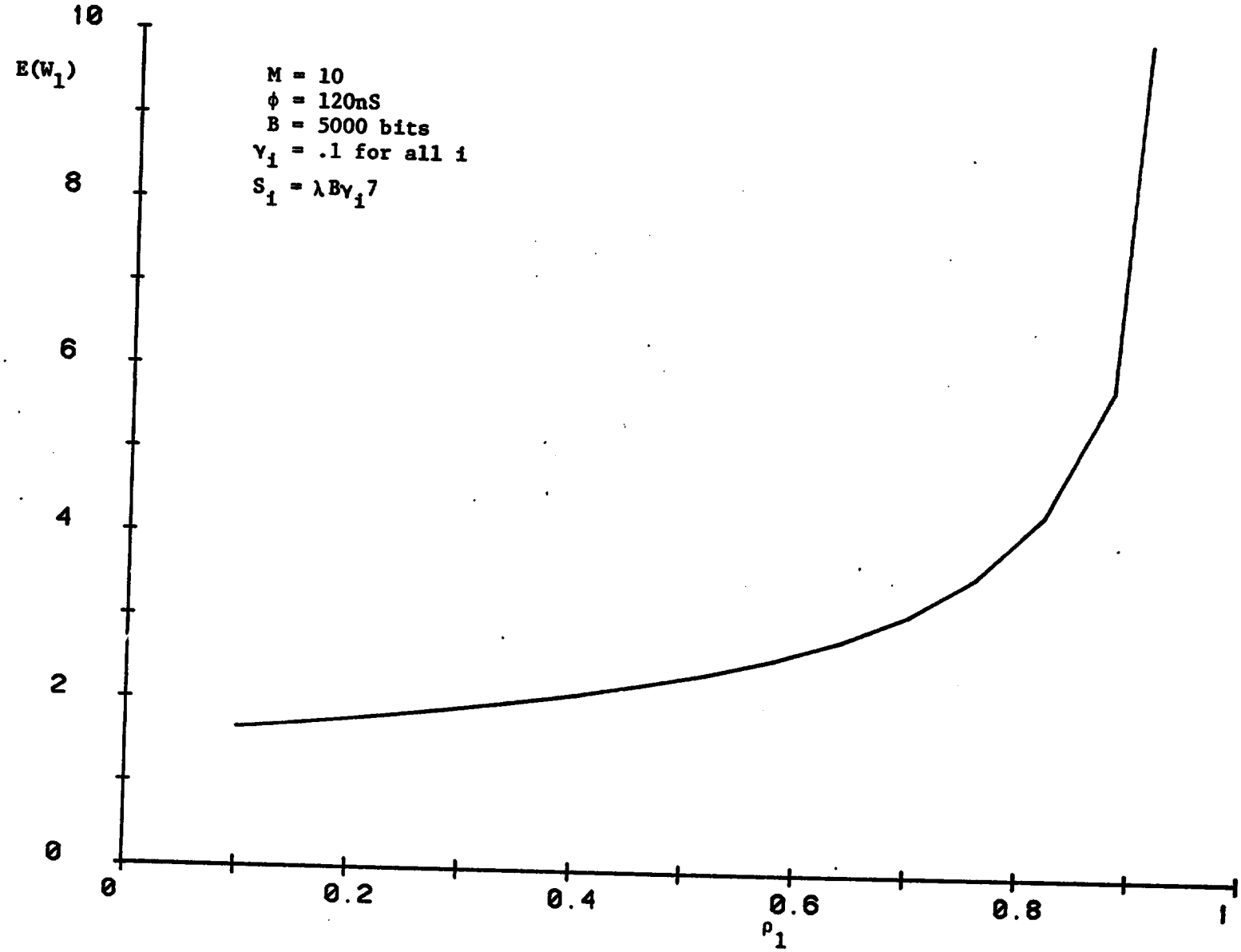


Fig. 6.56. Average Queue Size Vs. Utilization Factor At The Input Queue.

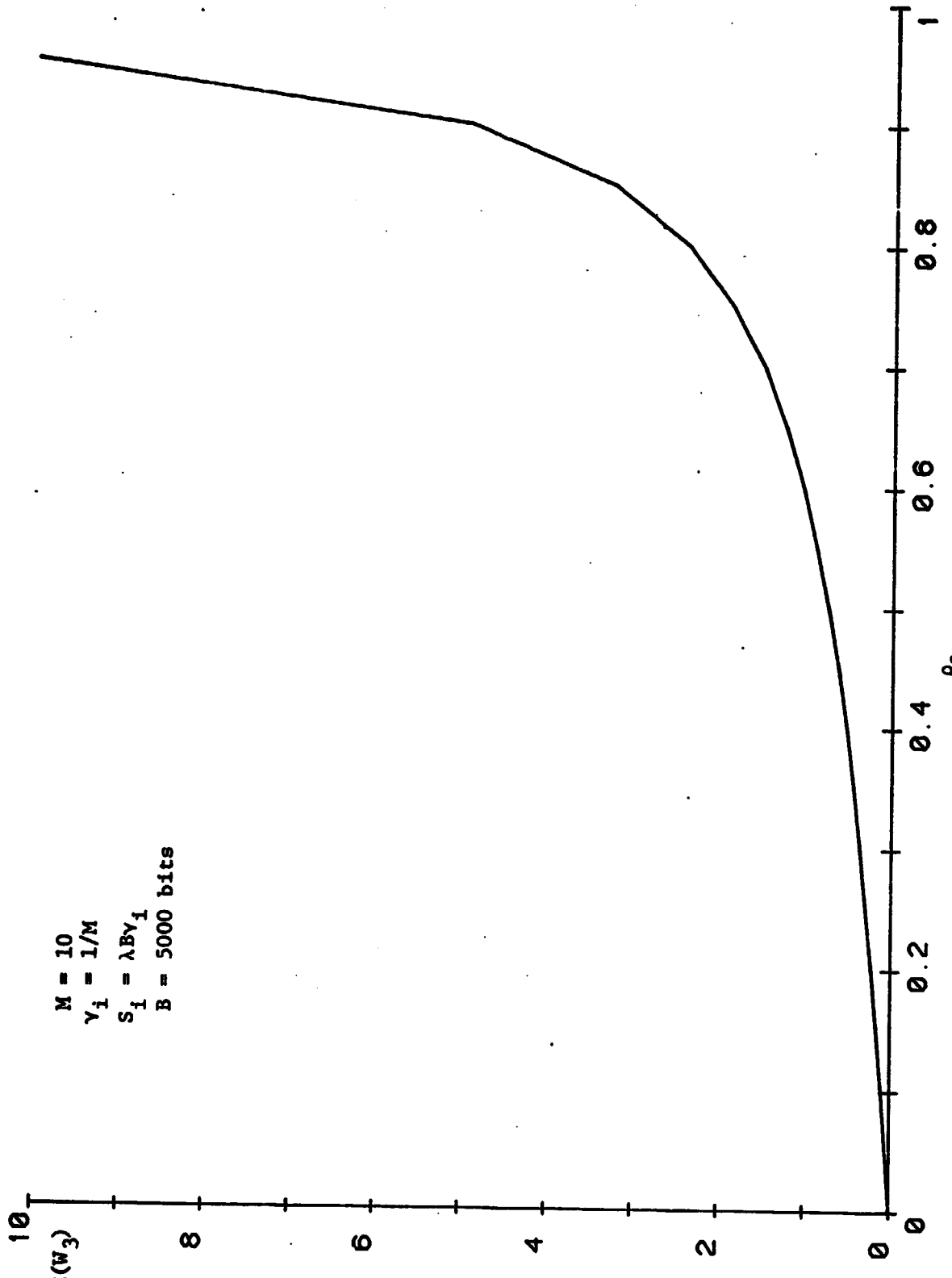


Fig. 6.57. Average Queue Size Vs. Utilization Factor At The Routing Queues.

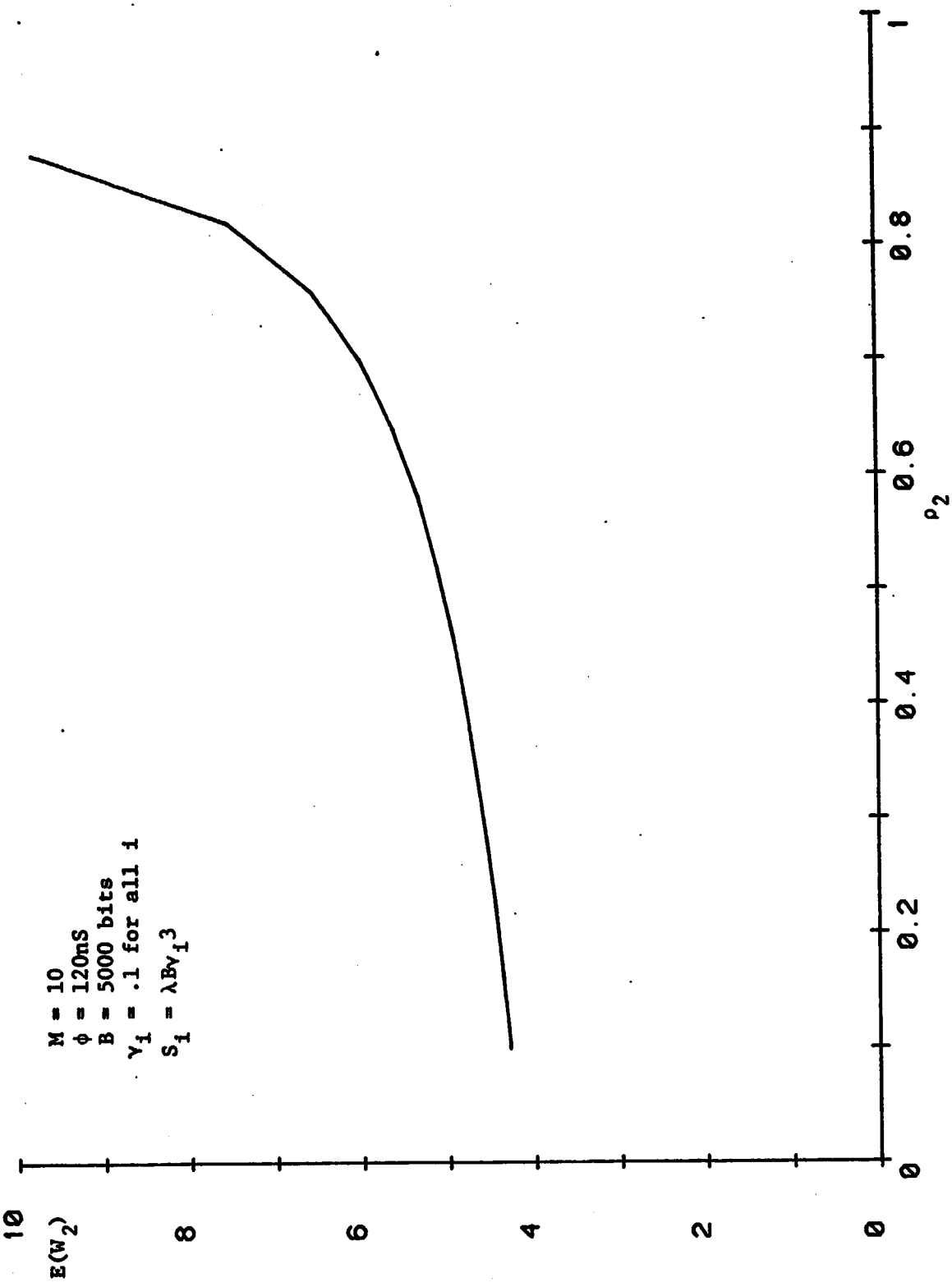


Fig. 6.58. Average Queue Size Vs. Utilization Factor At The Output Queue.

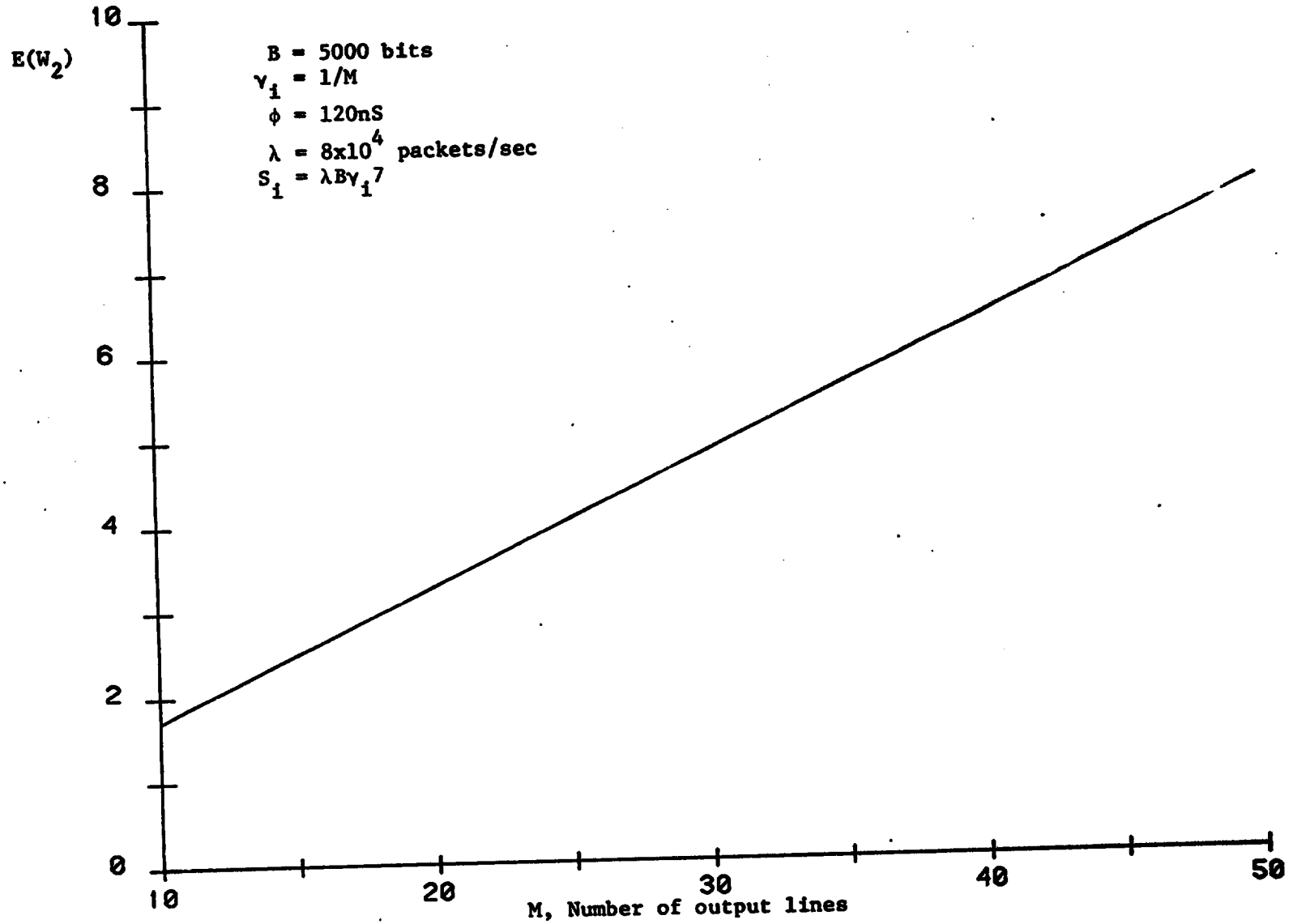


Fig. 6.59. Average Queue Size At The Output Queue Vs. The Number of Output Lines.

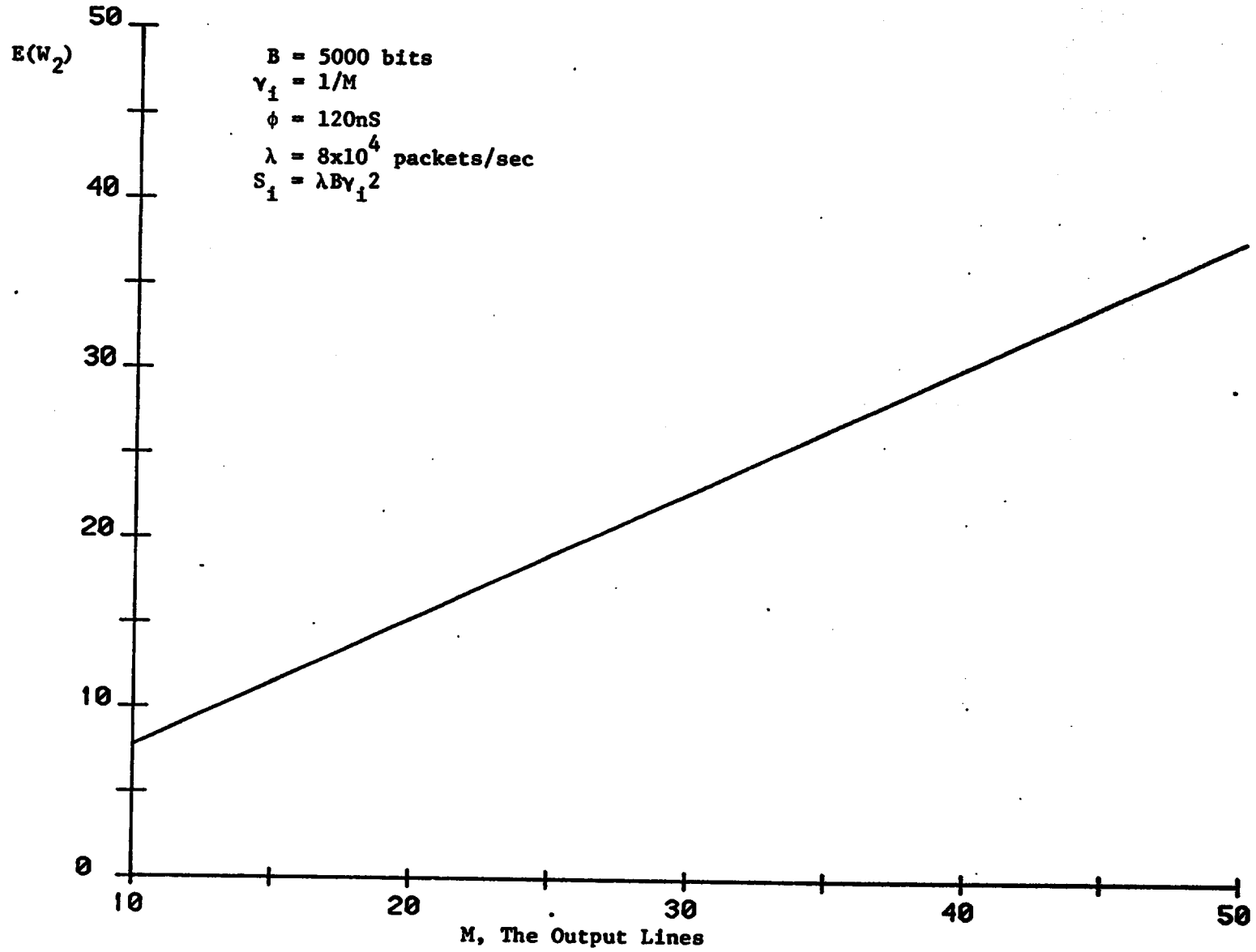


Fig. 6.60. Average Queue Size At The Output Queue Vs. The Number of Output Lines.

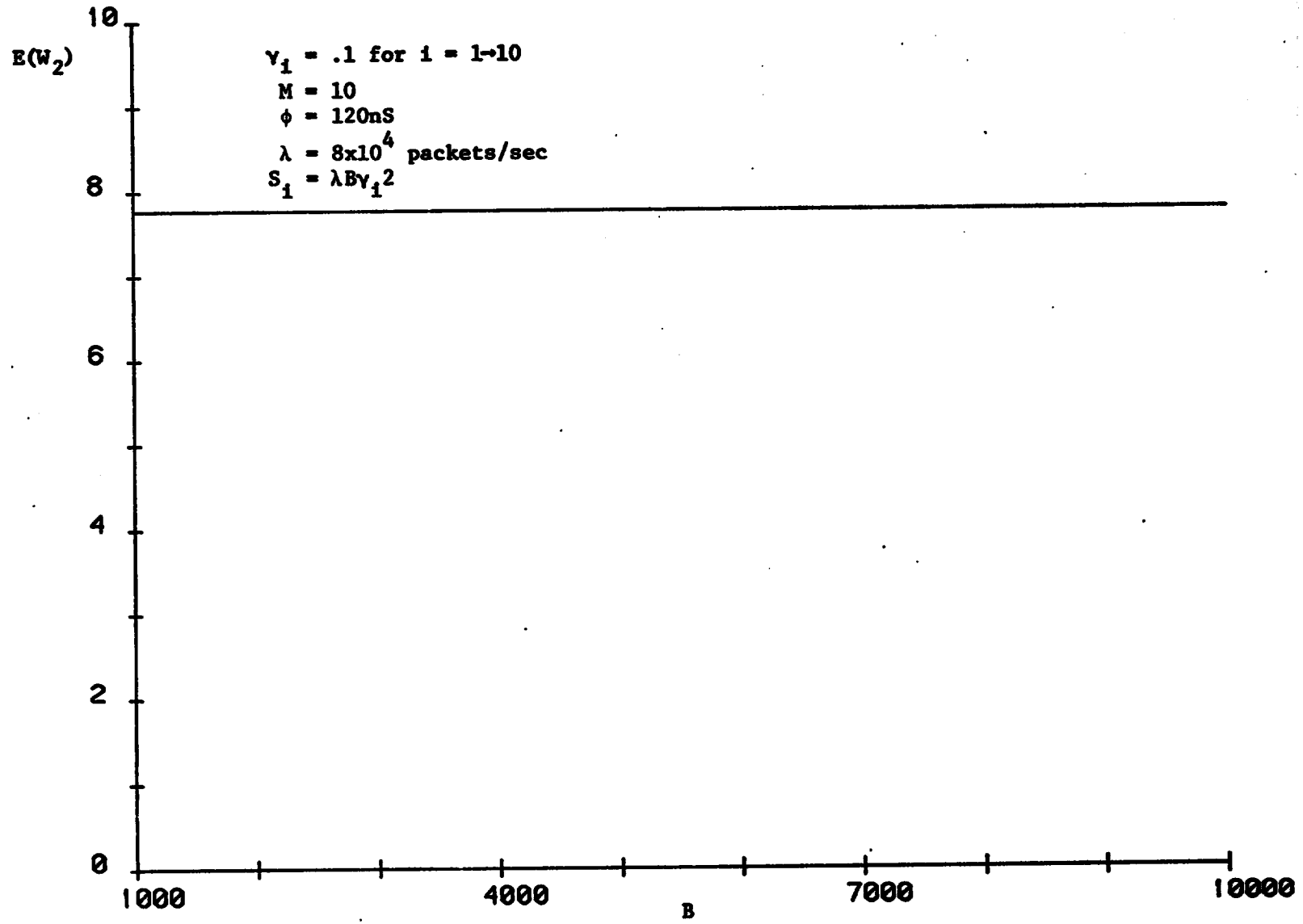


Fig. 6.61. Average Queue Size At The Output Queue Vs. The Packet Size.

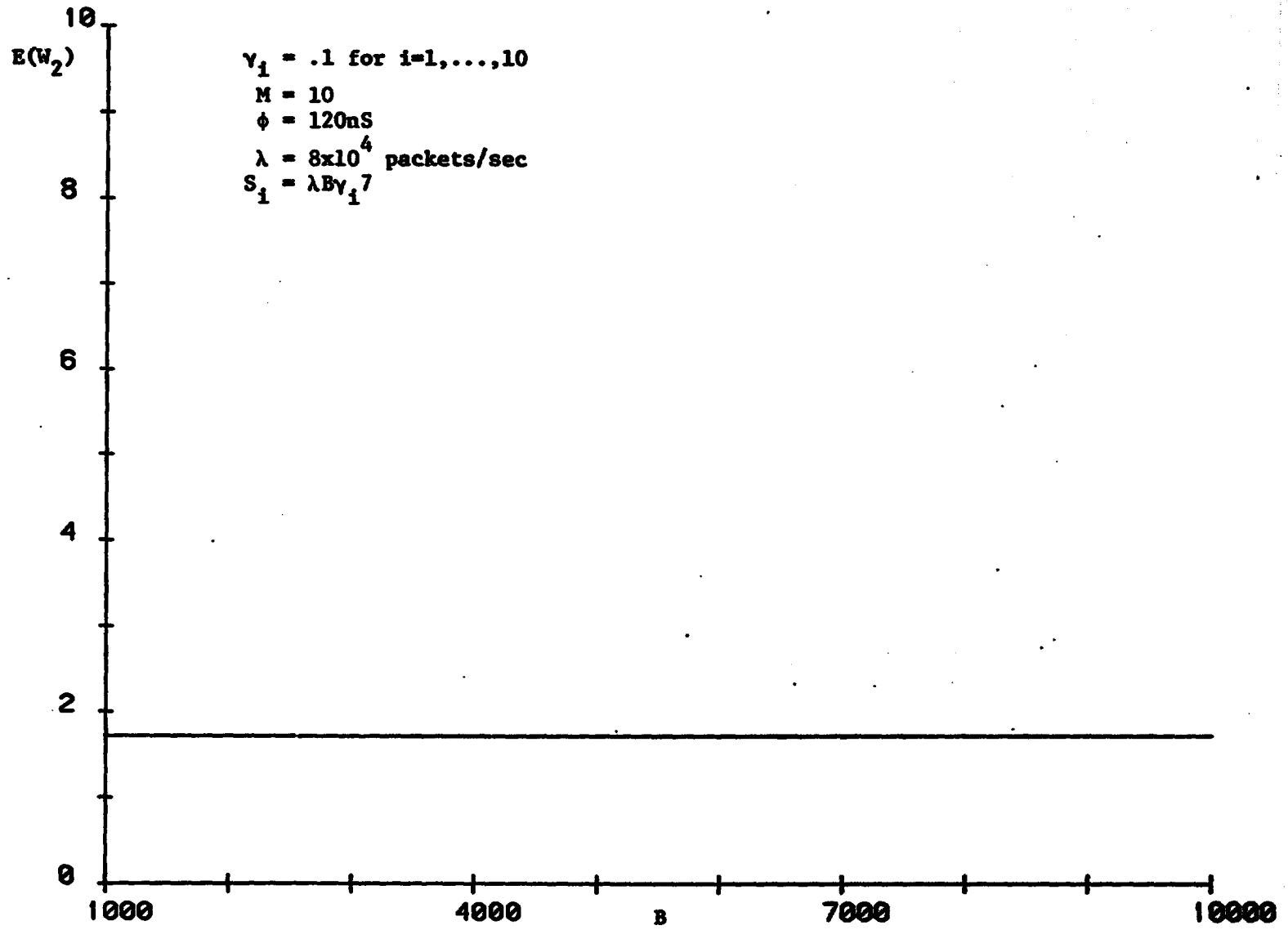


Fig. 6.62. Average Queue Size At The Output Queue Vs. Packet Size.

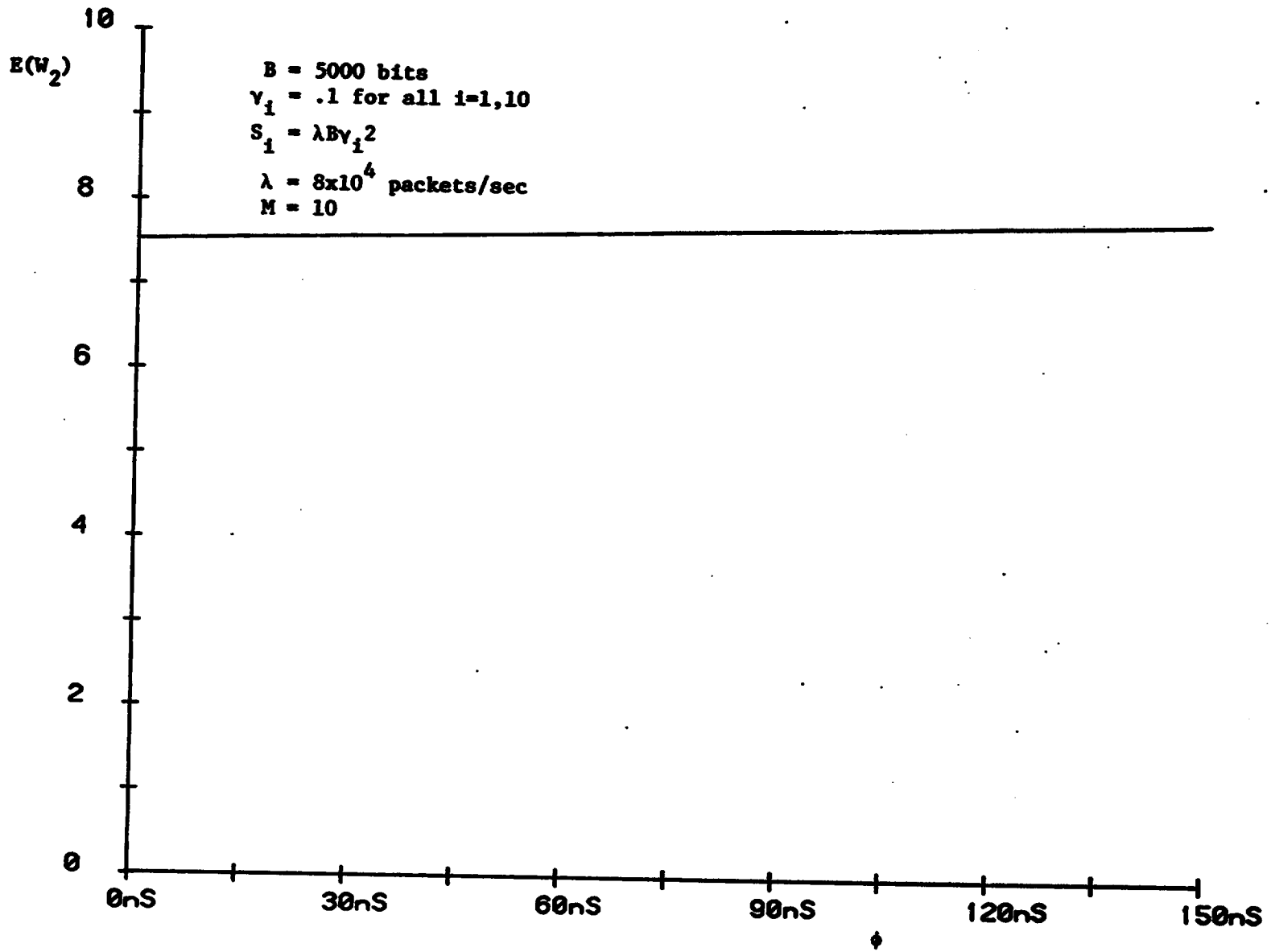


Fig. 6.63. Average Queue Size At The Output Queue Vs. The Clock Cycle Time.

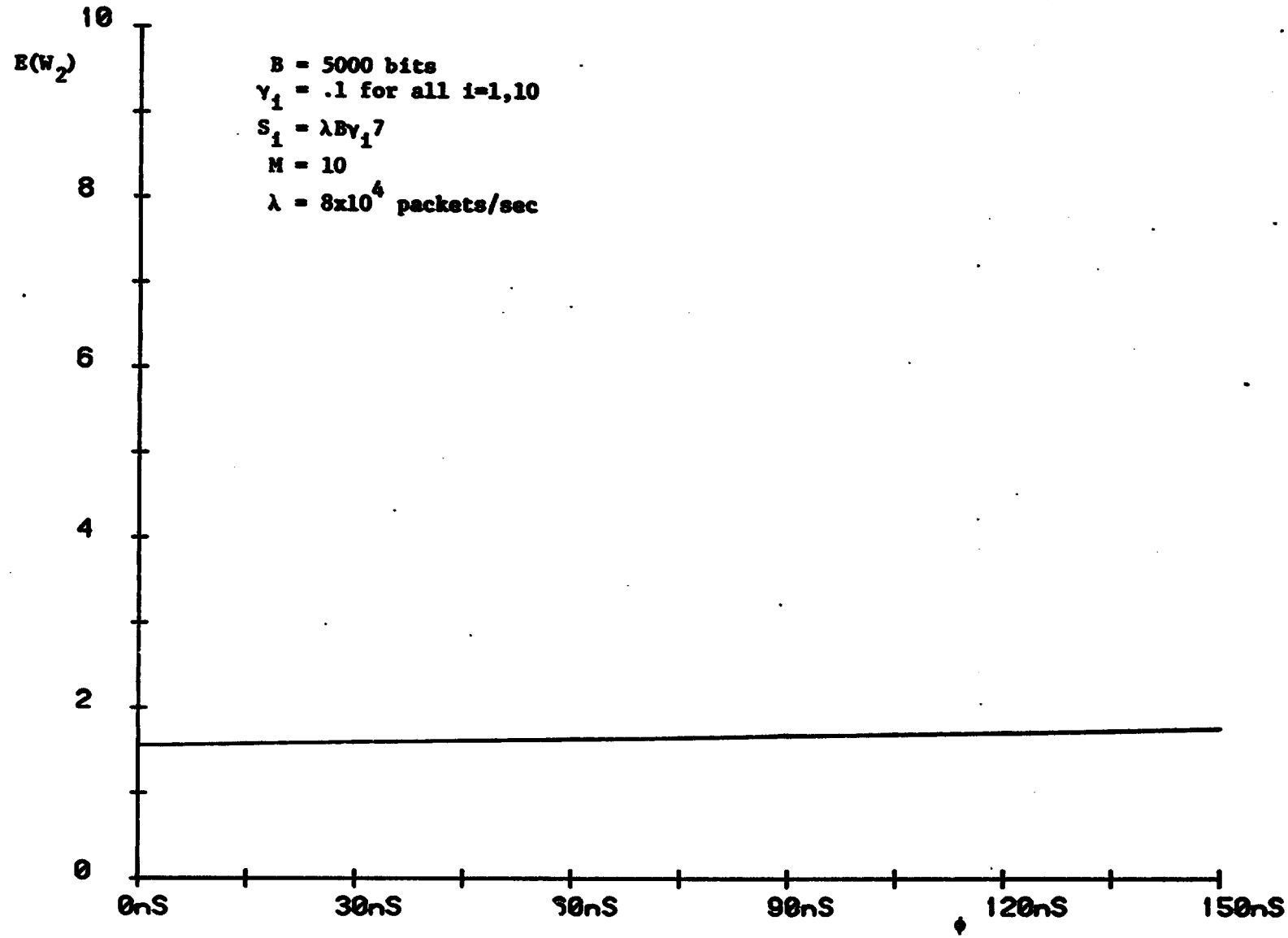
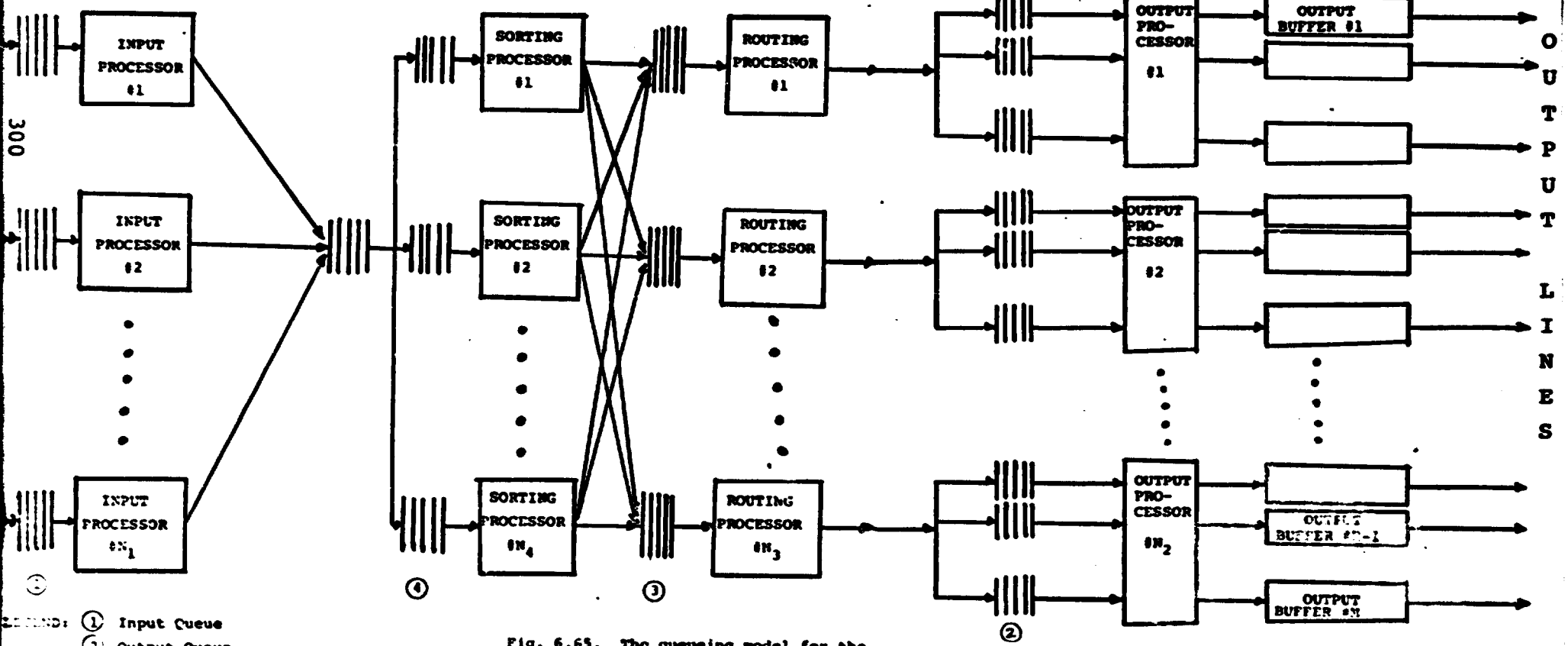


Fig. 6.64. Average Queue Size At The Output Queues Vs. The Clock Cycle Time.



LEGEND: (1) Input Queue
(2) Output Queue
(3) Routing Queue

Fig. 6.65. The queueing model for the multiple processor design.

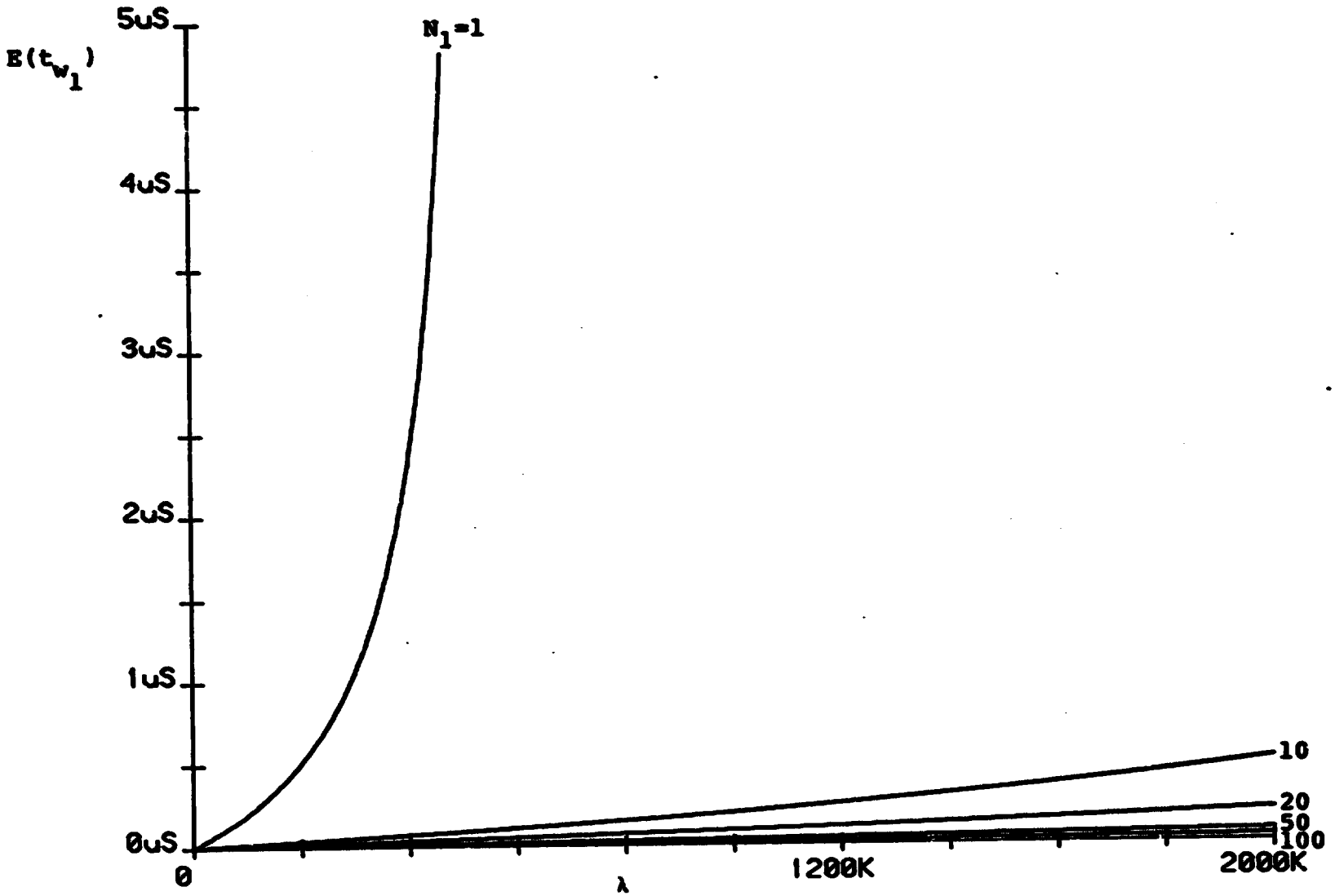


Fig. 6.66. Average Waiting Time Vs. Packet Arrival Rate At The Input Queue.

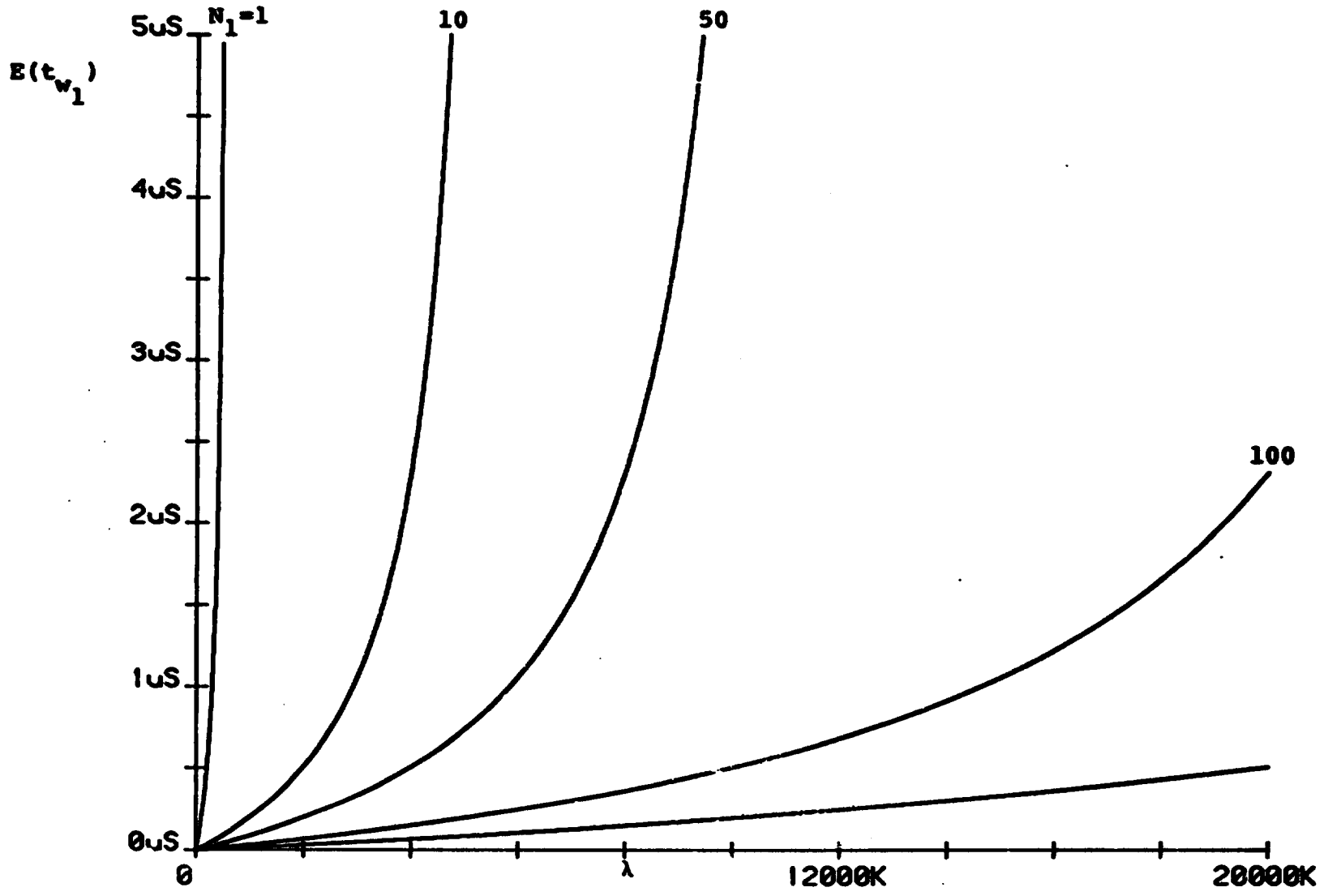


Fig. 6.67. Average Waiting Time Vs. Packet Arrival Rate At The Input Queue.

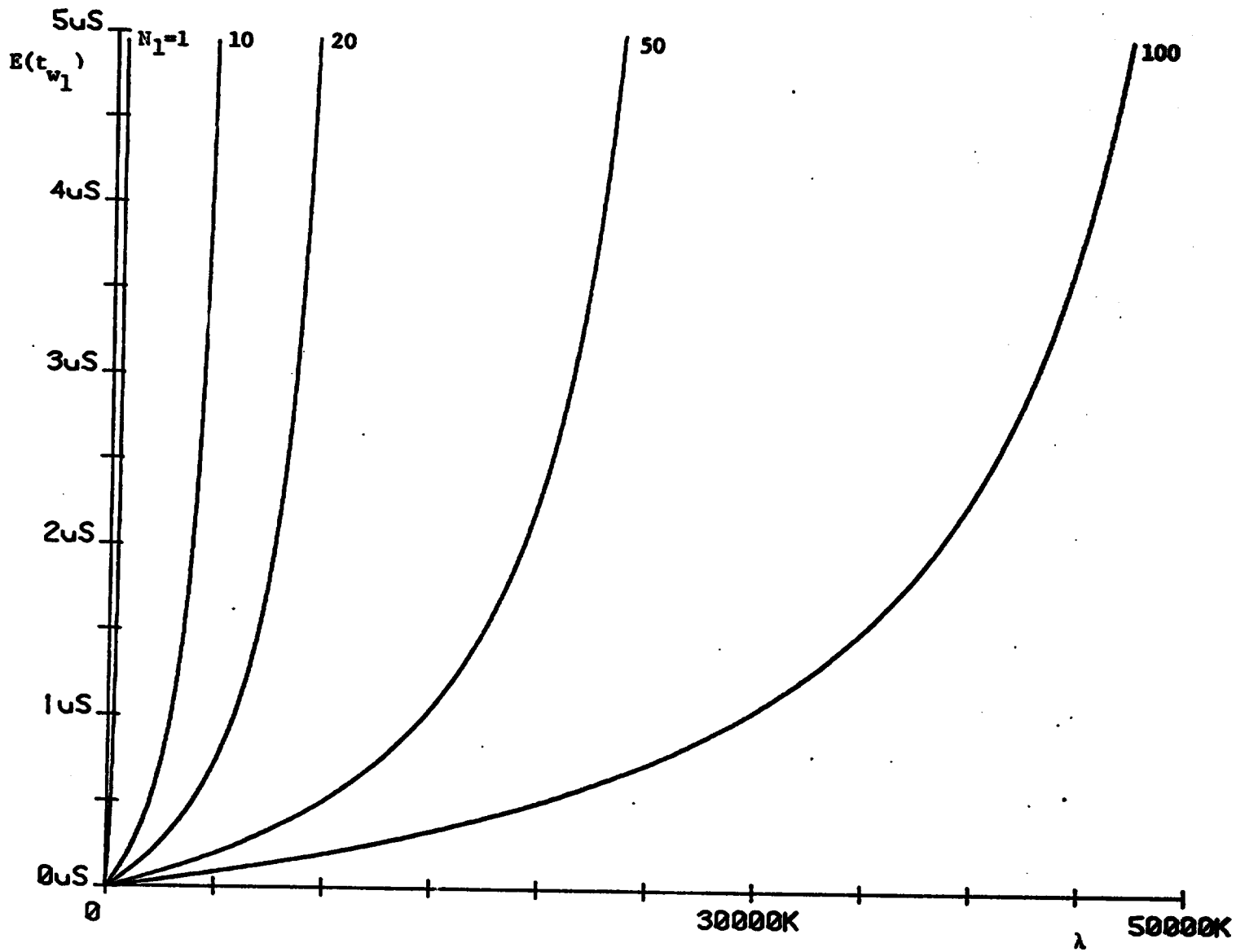


Fig. 6.68. Average Waiting Time Vs. Packet Arrival Rate At The Input Queue.

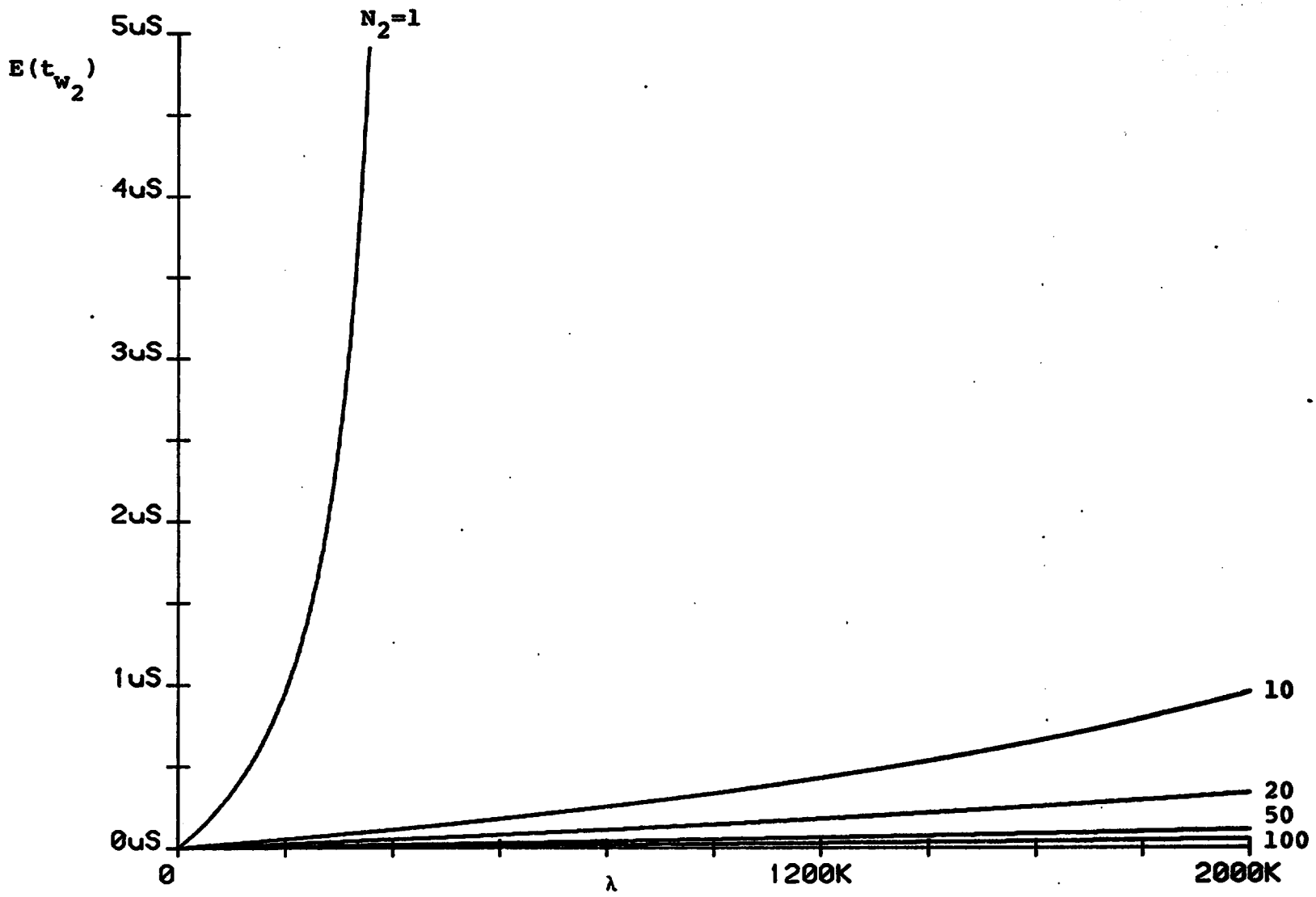


Fig. 6.69. Average Waiting Time Vs. Packet Arrival Rate At The Output Queue.

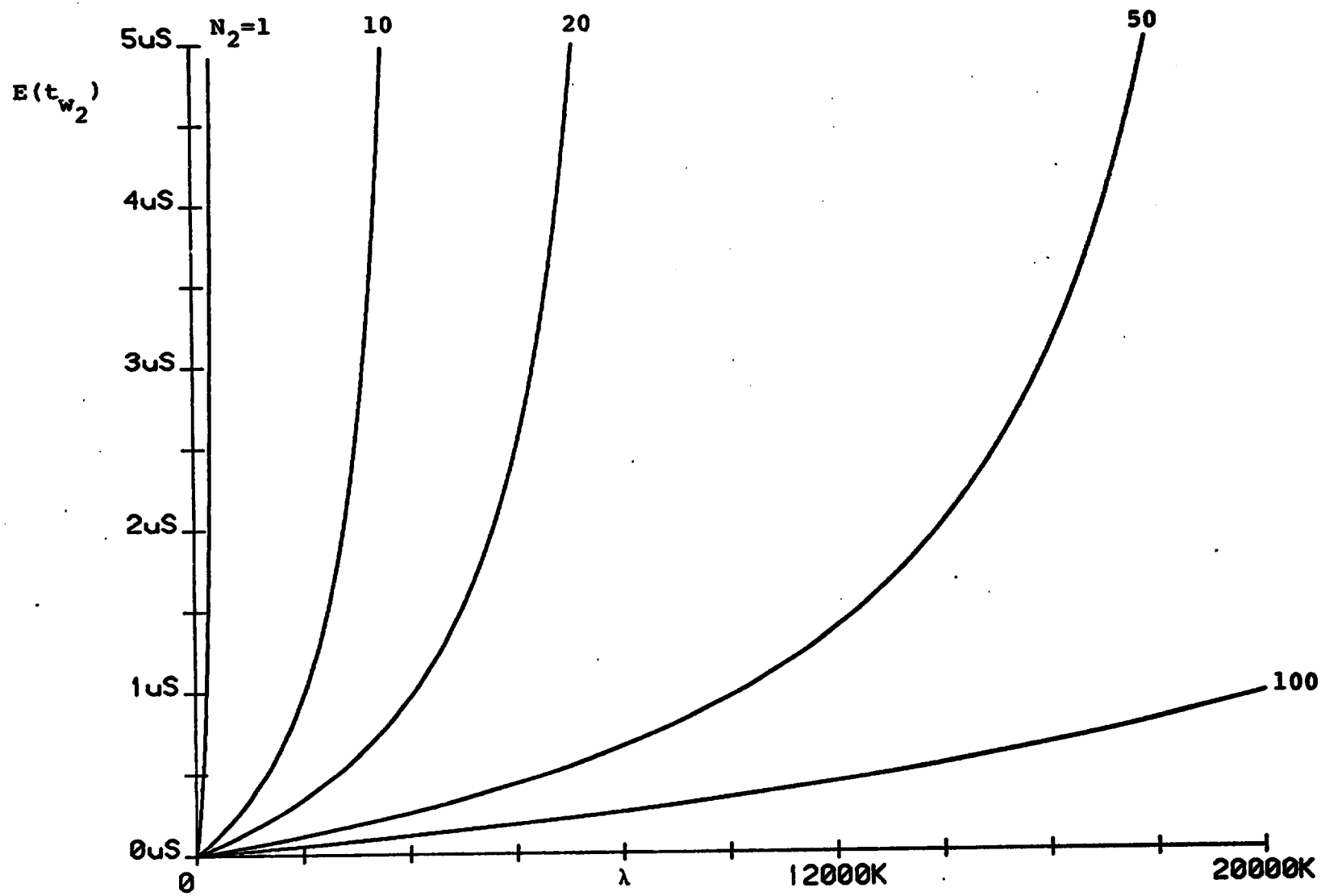


Fig. 6.70. Average Waiting Time Vs. Packet Arrival Rate At The Output Queue.

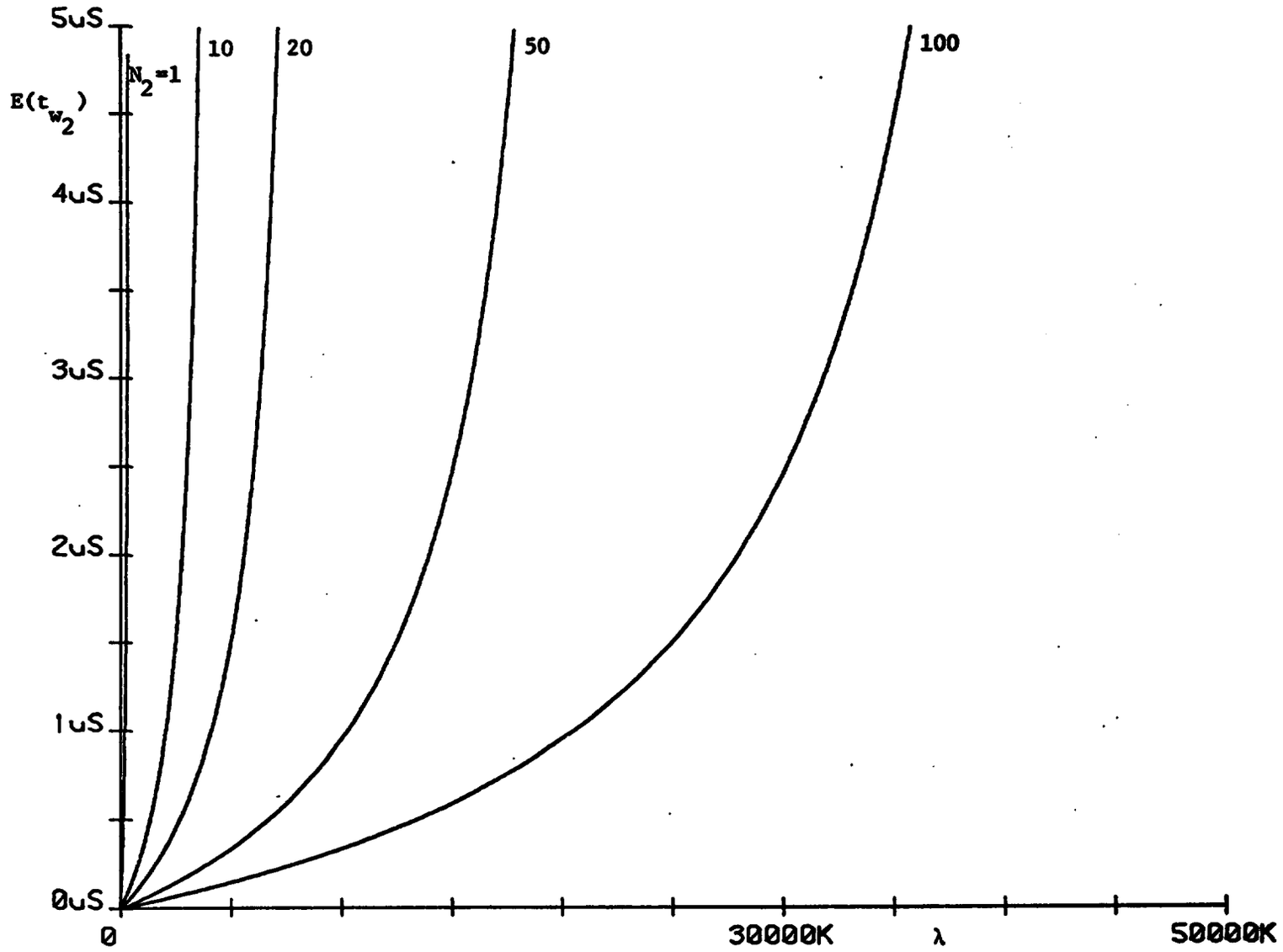


Fig. 6.71. Average Waiting Time Vs. Packet Arrival Rate At The Output Queue.

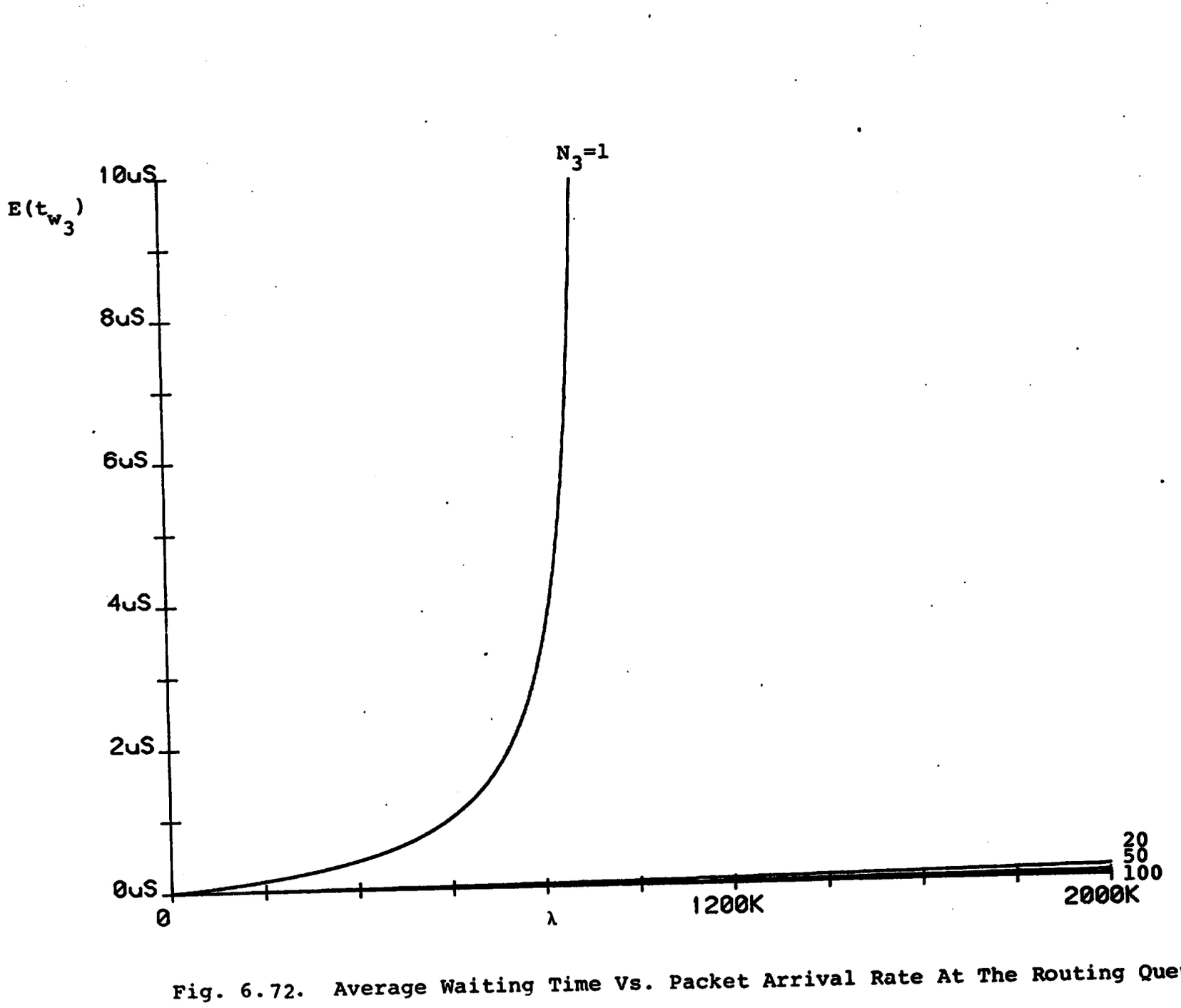


Fig. 6.72. Average Waiting Time Vs. Packet Arrival Rate At The Routing Queue.

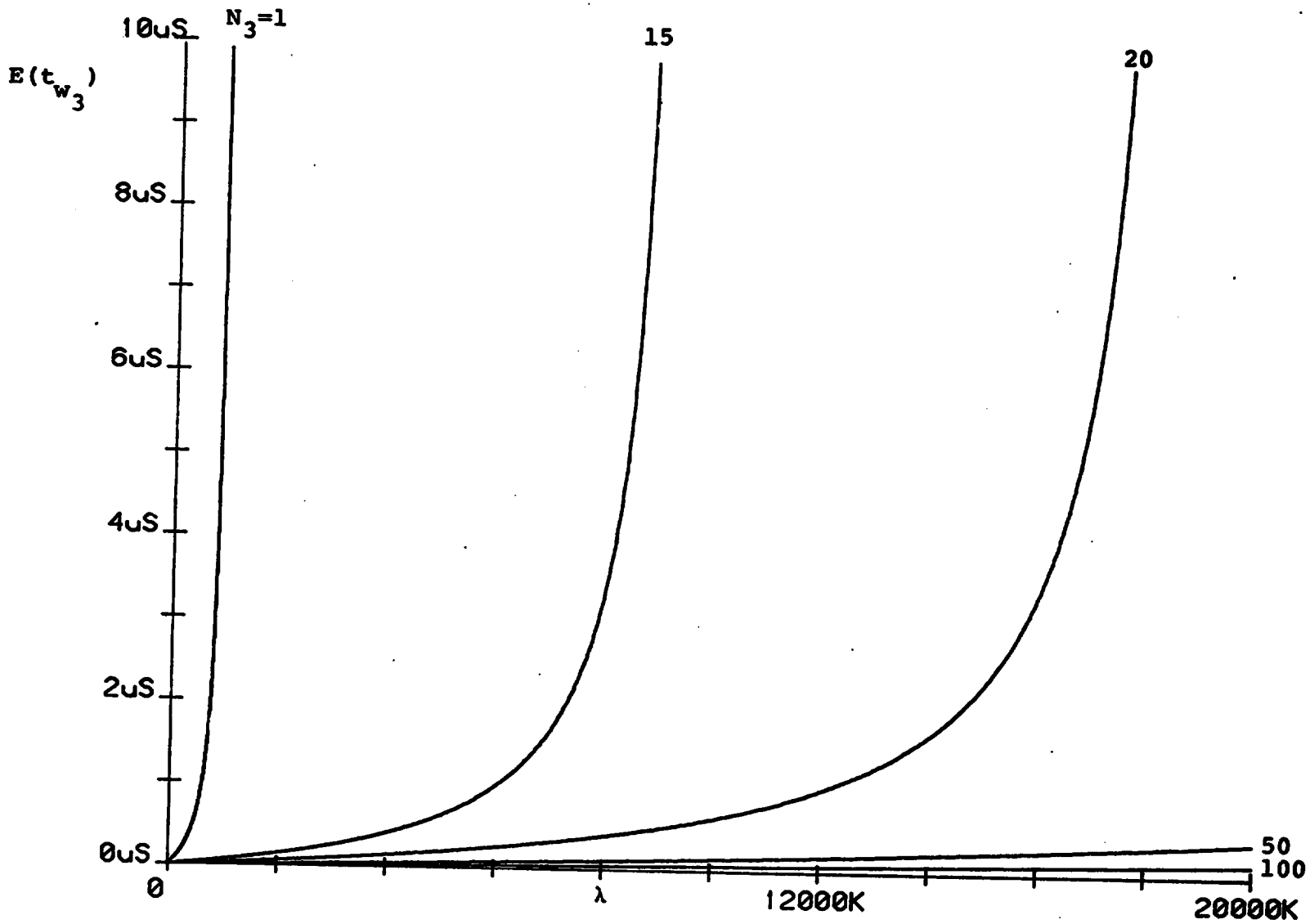


Fig. 6.73. Average Waiting Time Vs. Packet Arrival Rate At The Routing Queue.

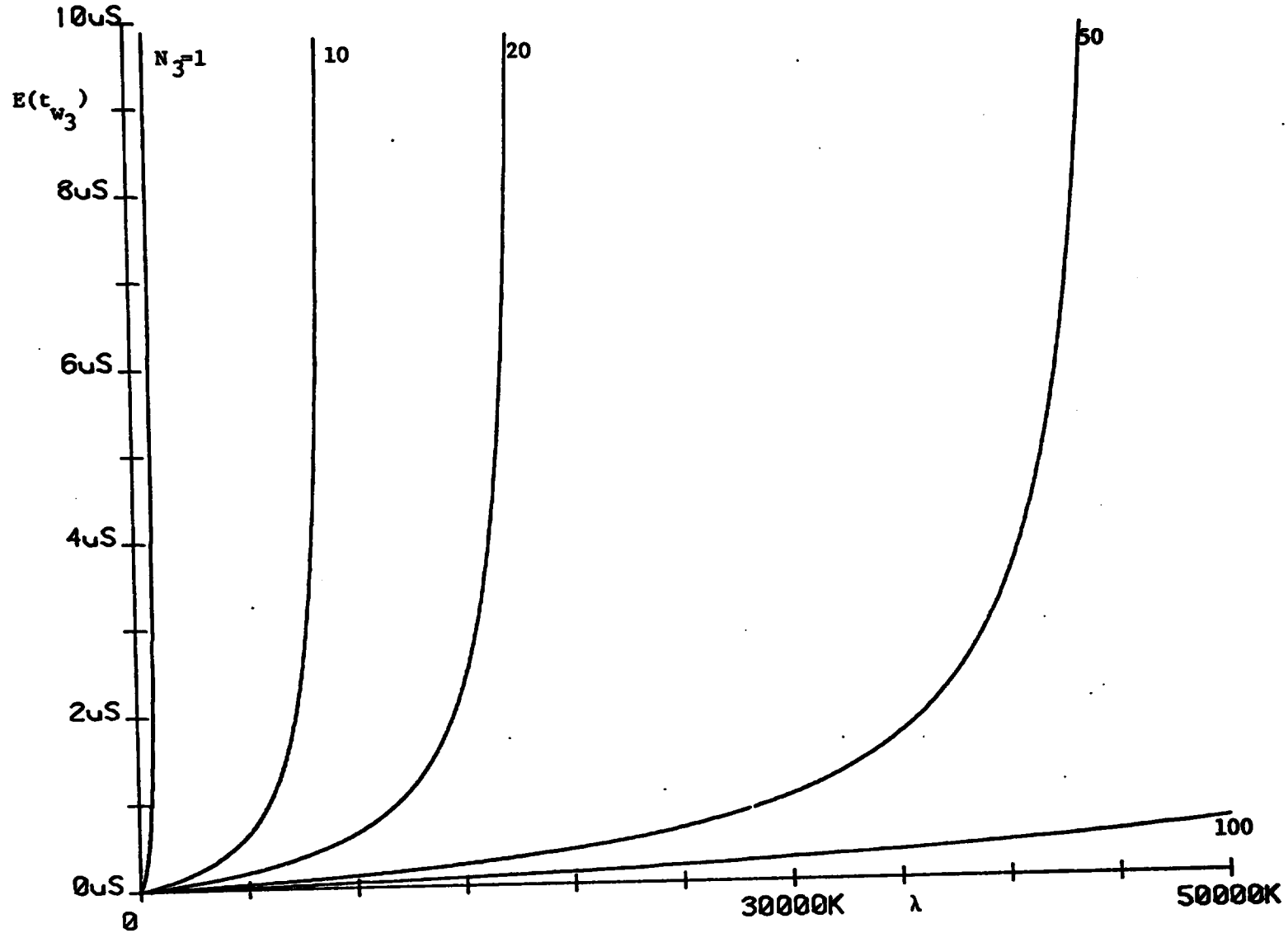


Fig. 6.74. Average Waiting Time Vs. Packet Arrival Rate At The Routing Queue.

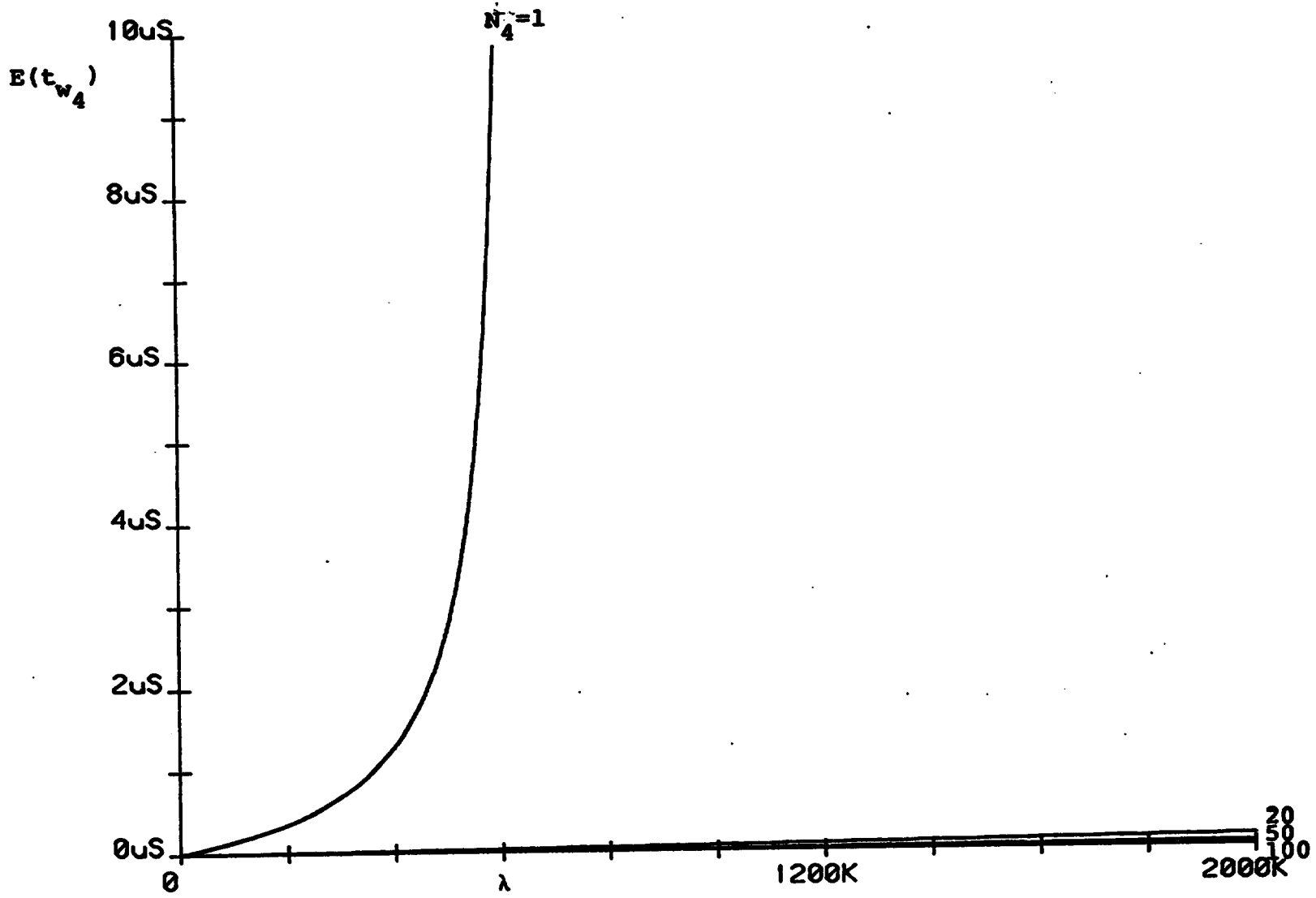


Fig. 6.75. Average Waiting Time Vs. Packet Arrival Rate At The Sorting Queue.

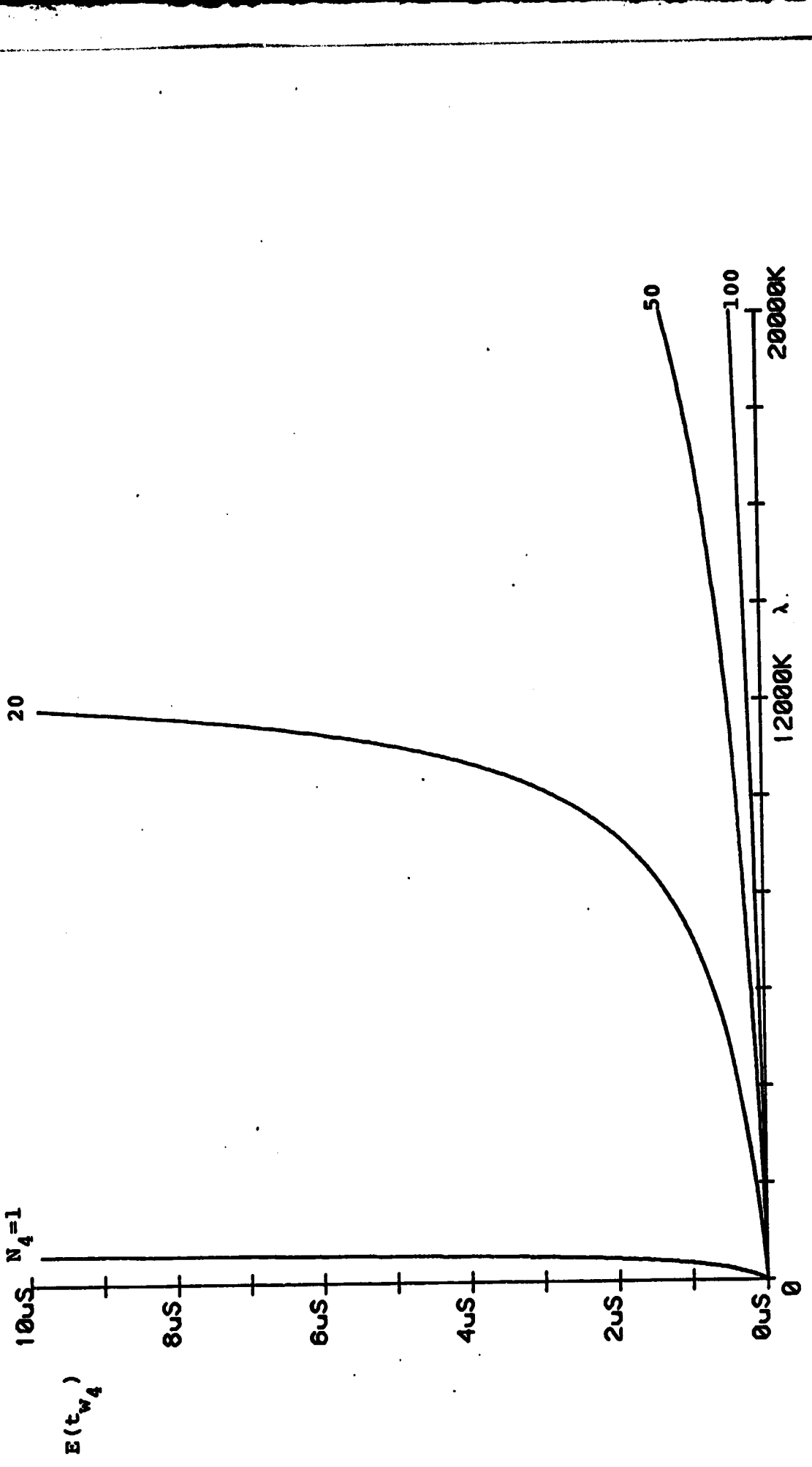


Fig. 6.76. Average Waiting Time Vs. Packet Arrival Rate At The Sorting Queue.

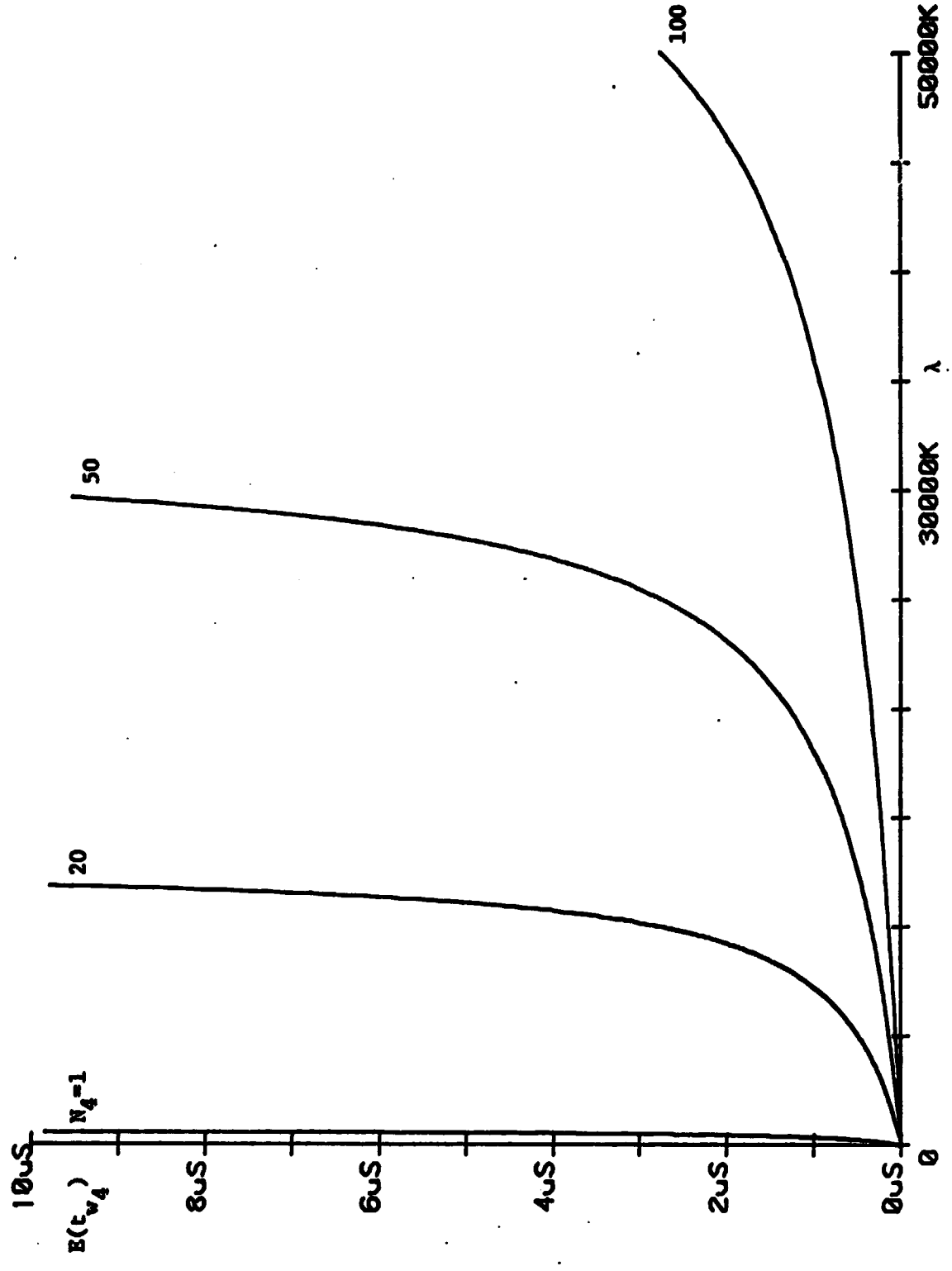


Fig. 6.7.7. Average Waiting Time Vs. Packet Arrival Rate At The Sorting Queue.

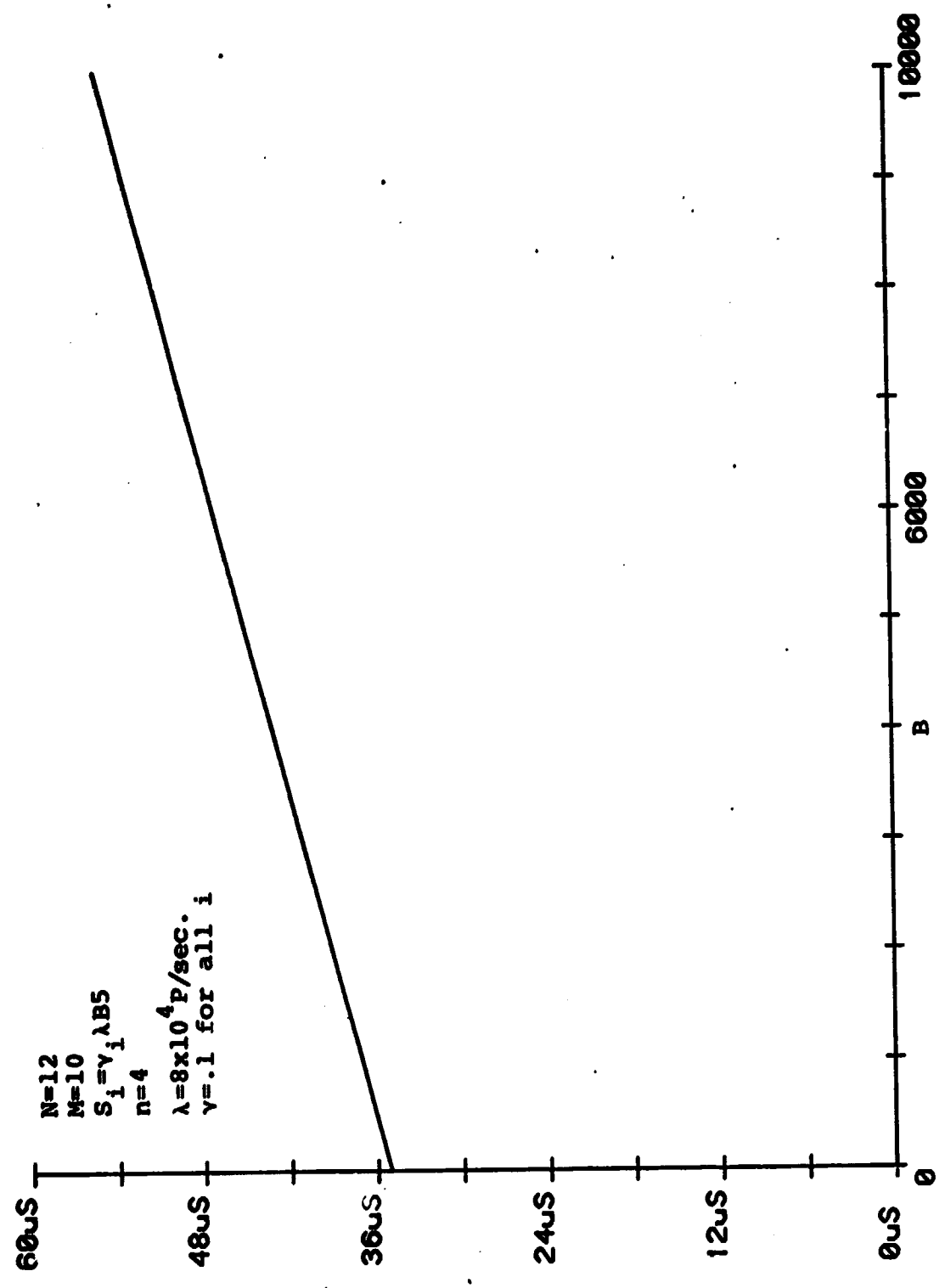


Fig. 6.78. Overall Average Waiting Time Vs. Packet Size.

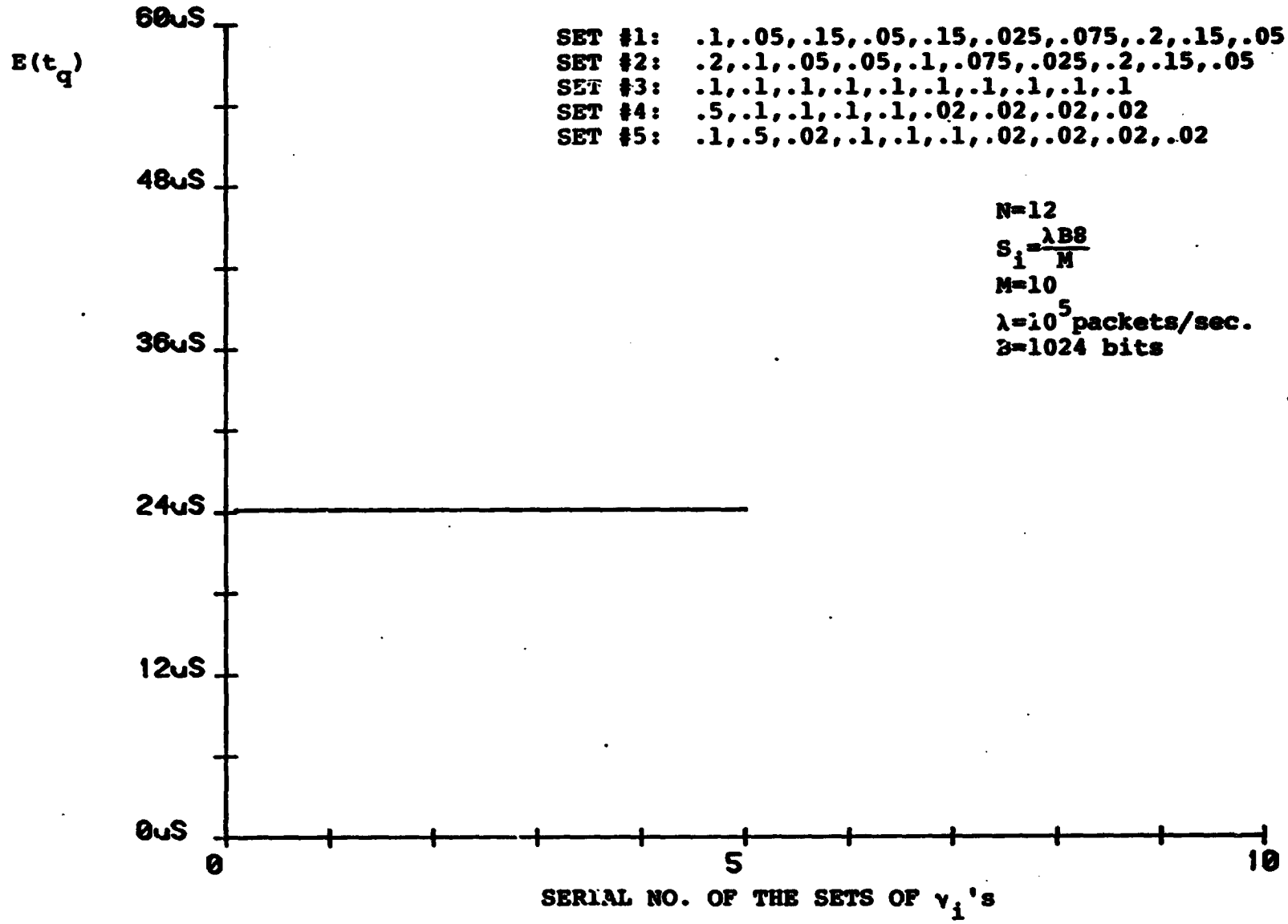


Fig. 6.79. Overall Average Waiting Time Vs. Destination Functions.

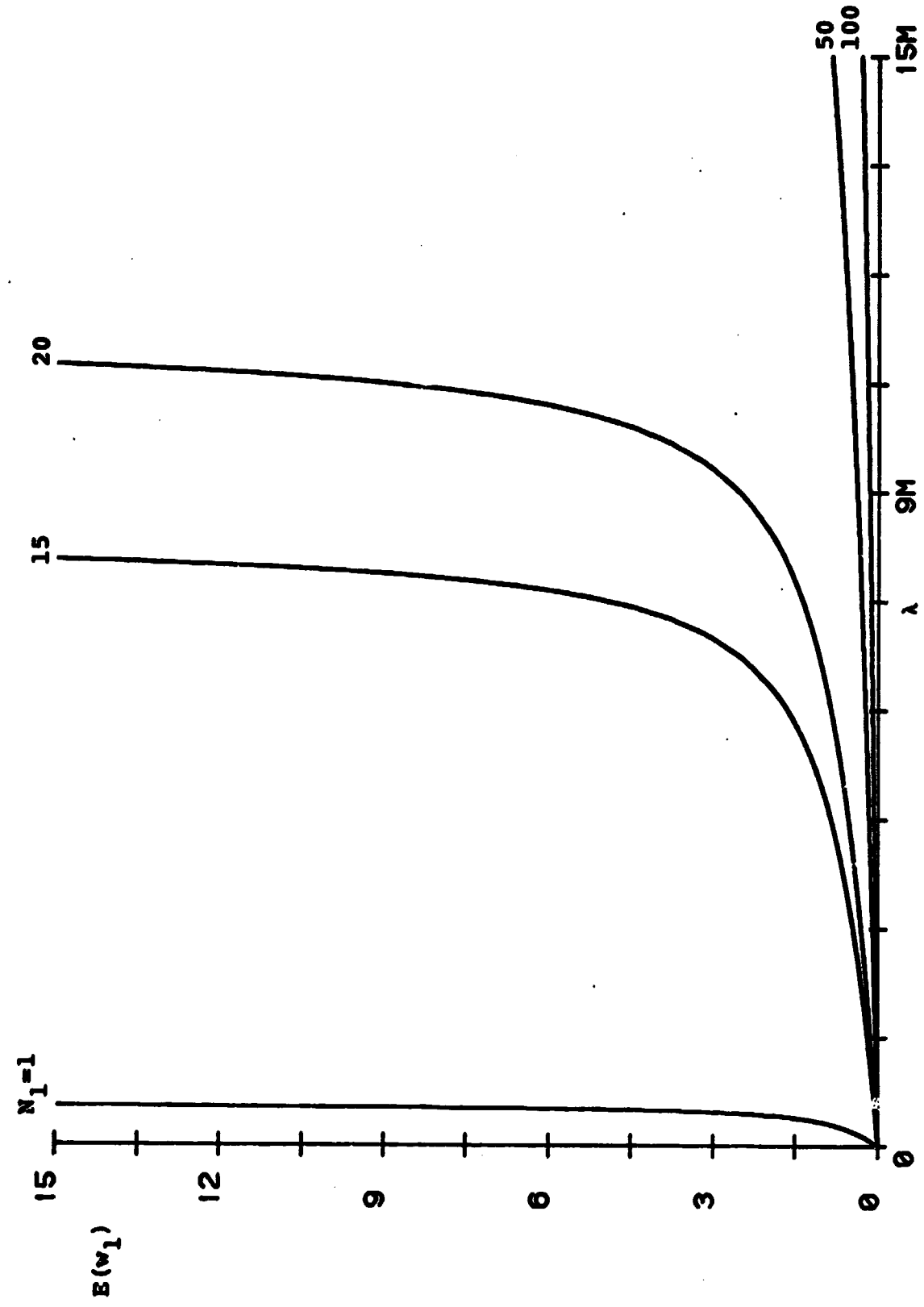


Fig. 6.80. Average Queue Size Vs. Packet Arrival Rate At The Input Queue.

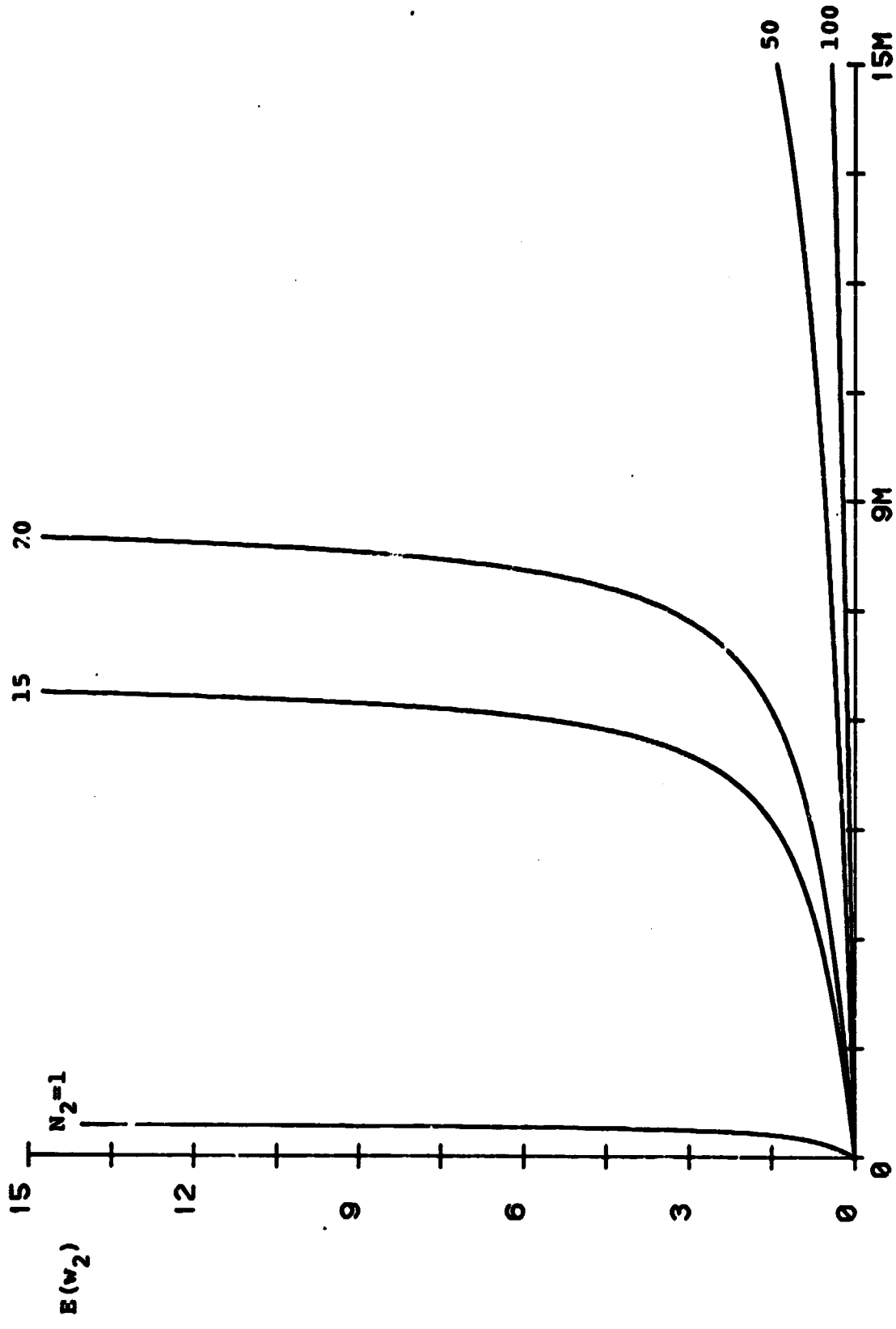


Fig. 6.81. Average Queue Size Vs. Packet Arrival Rate At The Output Queue.

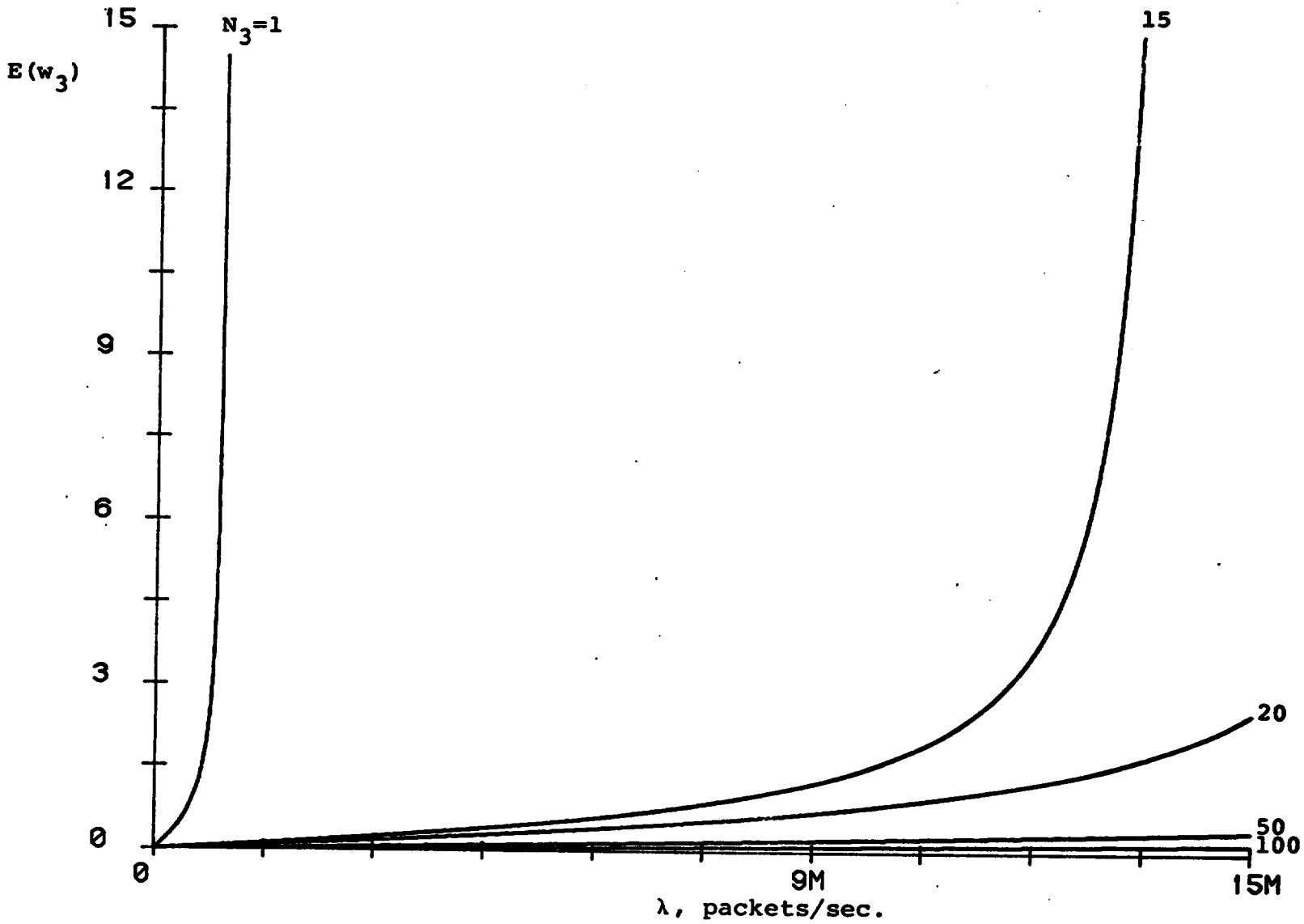


Fig. 6.82. Average Queue Size Vs. Packet Arrival Rate At The Routing Queue.

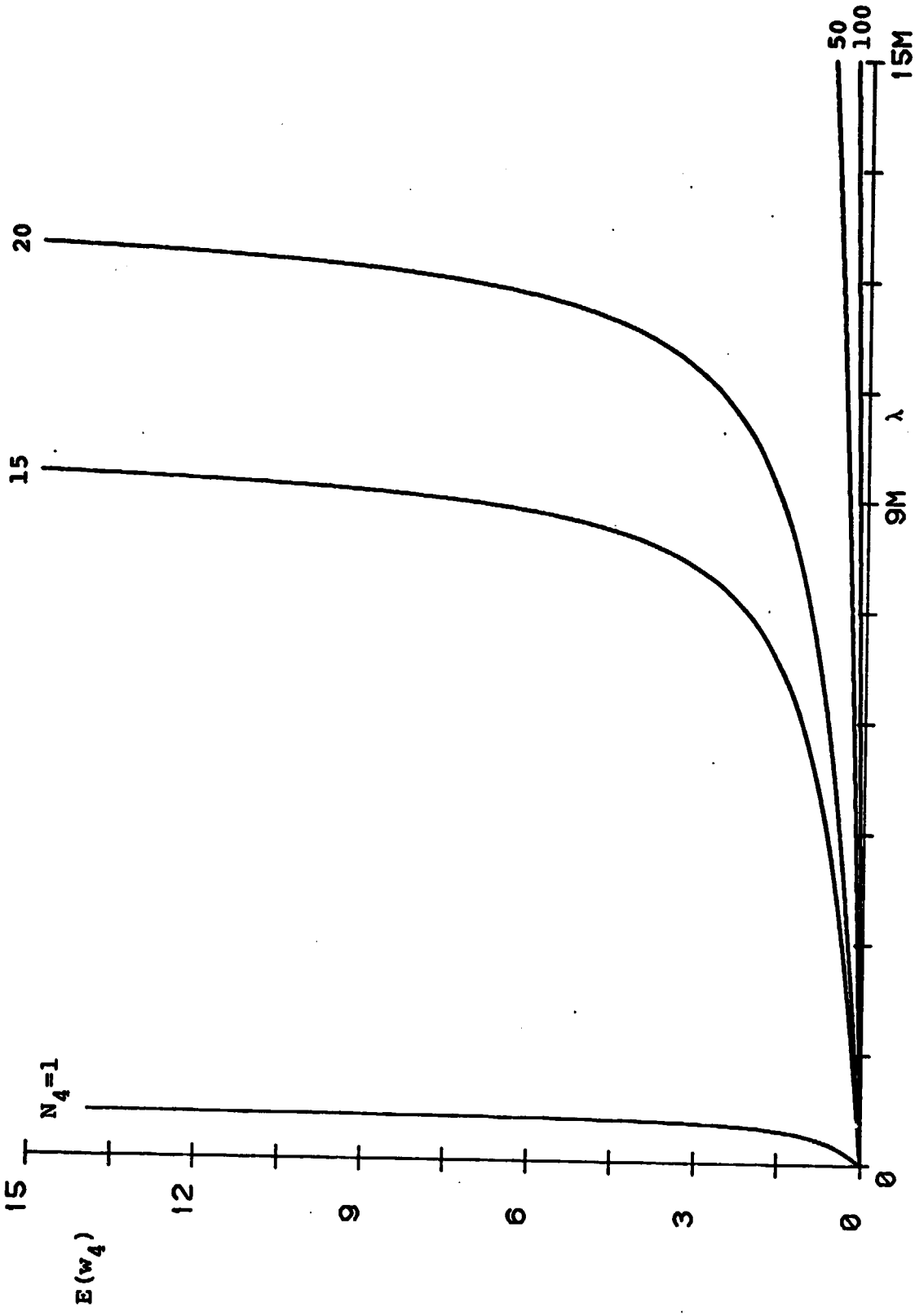


Fig. 6.83. Average Queue Size Vs. Packet Arrival Rate At The Sorting Queue.

APPENDIX A

INPUT SERVICE ROUTINE MICROCODE

IMMEDIATE OPERAND	ALU SOURCE		B ADDRESS	ALU FUNCTION	CARRY IN	ALU DESTINATION	BUS LATCH	OUTPUT ENABLE	DECODER ENABLE	WRITE ENABLE	ELIST READ	ELIST UPDATE	POLLER RESET	NEXT ADDRESS SELECT	BRANCH ADDRESS
	R SOURCE	S SOURCE													

INPUT: If IBSR-A = Ø, JMP to INPUT

	MW	Polling Port	scratch1	F=R+Cn	Ø	F→Y	NONE	YES	NO	NO	NO	NO	NO	Jump on IESR-A = Ø	INPUT
	2		Ø	6	Ø	4	Ø	Ø	Ø	1	1	Ø	Ø	2	

Input Polling Port → Q

	MW	Polling Port	scratch1	F=S+Cn	Ø	F→Q F→Y	NONE	NO	NO	NO	NO	NO	NO	mic+1	
	2		Ø	4	Ø	6	Ø	1	Ø	1	1	Ø	Ø	Ø	

[ELIST] @ EPTR$\emptyset$ → Scratch1 ; Reset Poller

	IBUS	Reg@B	scratch1	F = R+Cn	$\emptyset$	F → Y	NONE	YES	NO	YES	YES	NO	YES	μPC+1	
	4		$\emptyset$	6	$\emptyset$	4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	1	$\emptyset$	

Input Data Path Status Port Address → Address Latch

Data Path status Port address	μW	Reg@B	scratch2	F = R+Cn	$\emptyset$	F → Y	ADDRESS LATCH	YES	NO	NO	NO	NO	NO	μPC+1	
	$\emptyset$		1	6	$\emptyset$	4	1	$\emptyset$	$\emptyset$	1	1	$\emptyset$	$\emptyset$	$\emptyset$	

Data Path Busy Status Port → scratch2 ; update EPTR$\emptyset$

	IBUS	Reg@B	scratch2	F = R+Cn	$\emptyset$	F → Y	NONE	YES	NO	YES	NO	YES	NO	μPC+1	
	4		1	6	$\emptyset$	4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	1	1	$\emptyset$	$\emptyset$	

Scratch 2 + Data Path Latch A Base Address → Address Latch

Data Path Latch A Base Address	MW	Reg@B	Scratch2	F=R+S+Cn	Ø	F→Y	ADDRESS LATCH	YES	NO	NO	NO	NO	NO	μPC+1	
	Ø		1	3	Ø	4	1	Ø	Ø	1	1	Ø	Ø	Ø	

Q → Data Path mux select Latch A (D)

	IBUS	Q	Scratch2	F=S+Cn	Ø	F→Y	NONE	YES	NO	NO	NO	NO	NO	μPC+1	
	7		1	4	Ø	4	Ø	Ø	Ø	1	1	Ø	Ø	Ø	

Scratch2 + Data Path Latch B Base Address → Address Latch

Data Path Latch B Base Address	MW	Reg@B	Scratch2	F=R+S+Cn	Ø	F→Y	ADDRESS LATCH	YES	NO	NO	NO	NO	NO	μPC+1	
	Ø		1	3	Ø	4	1	Ø	Ø	1	1	Ø	Ø	Ø	

Scratch 1 → Data Path Demux Select Latch B (0)

	MW	Reg@B	Scratch1	F = S + C_n	Ø	F → Y	NONE	YES	NO	NO	NO	No	NO	μPc + 1	
	Ø		Ø	4	Ø	4	Ø	Ø	Ø	1	1	Ø	Ø	Ø	

Data Path Transmit Control Address → Address Latch

Data Path Transmit Control Address	MW	Reg@B	Scratch2	F = R + C_n	Ø	F → Y	ADDRESS LATCH	YES	NO	NO	NO	NO	NO	μPc + 1	
	Ø		1	6	Ø	4	1	Ø	Ø	1	1	Ø	Ø	Ø.	

Scratch 2 → Data Bus Decoder ; Jmp to INPUT

	MW	Reg@B	Scratch2	F = S + C_n	Ø	F → Y	NONE	YES	YES	NO	NO	NO	NO	UNCONDITIONAL Jump	INPUT
	Ø		1	4	Ø	4	Ø	Ø	1	1	1	Ø	Ø	1	

APPENDIX B

PROCESSOR-CONTROLLED ELIST

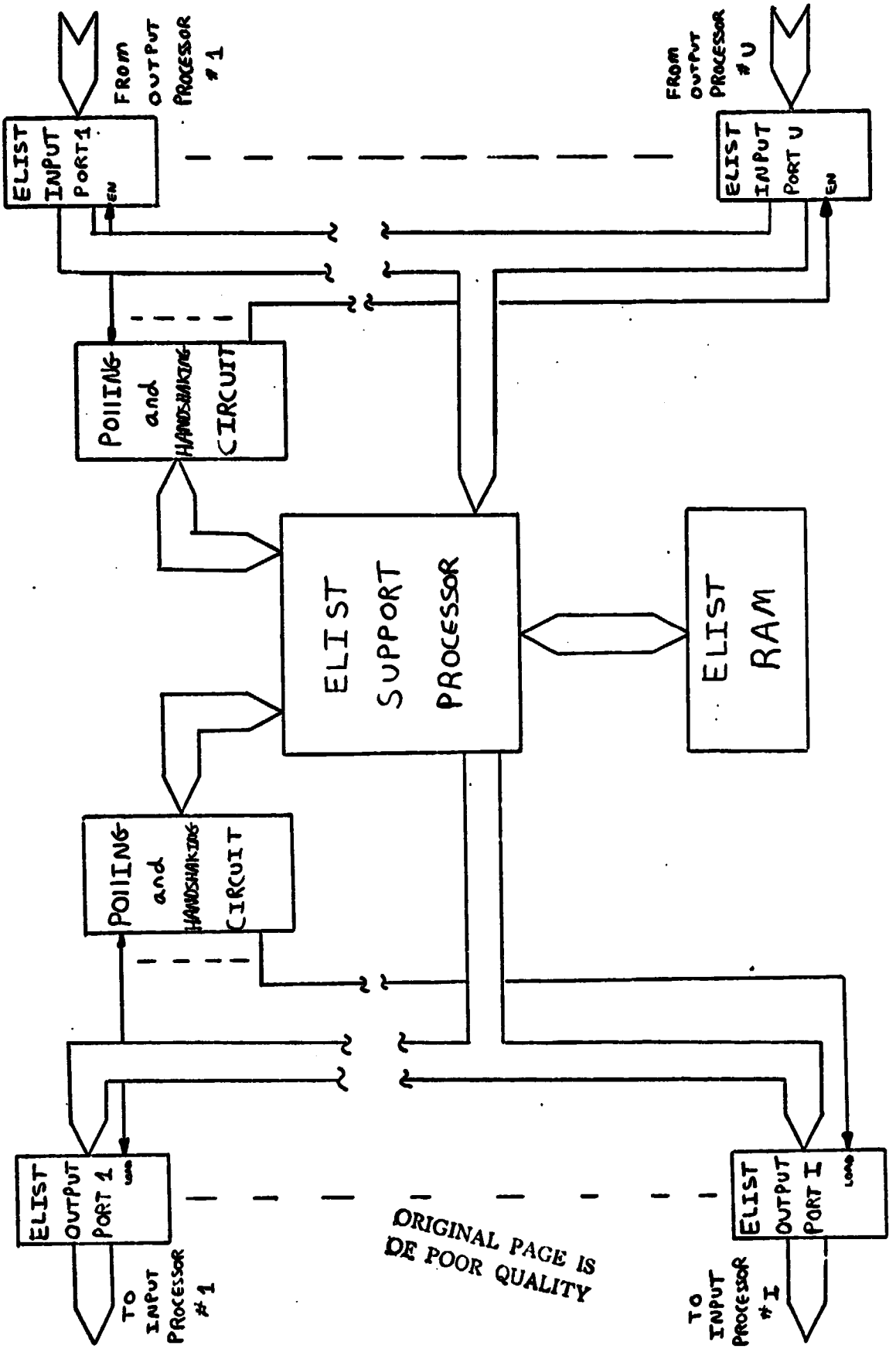


Fig. B1. Processor-Controlled ELIST Architecture.

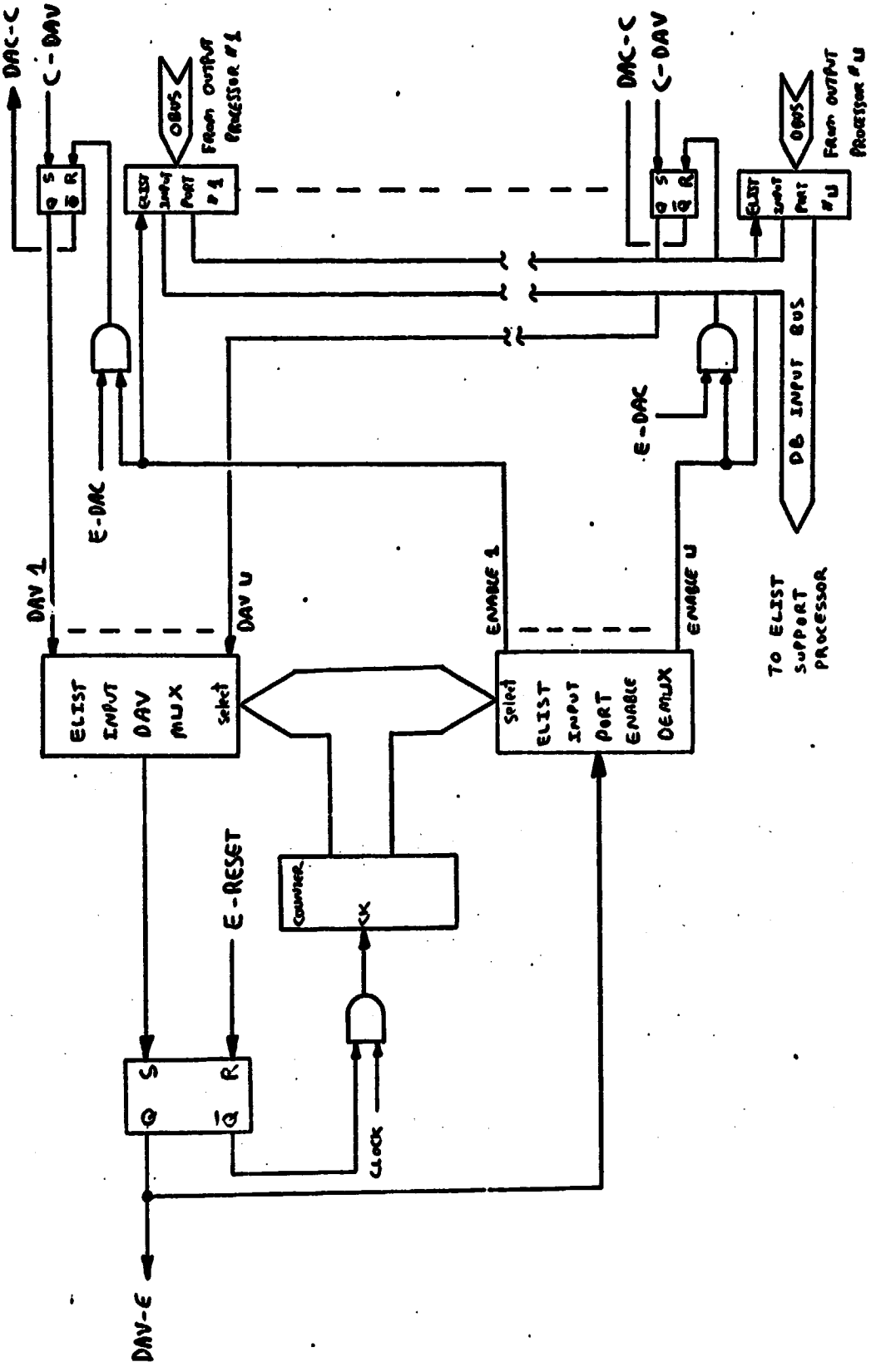


Fig. B2. ELIST INPUT DATA Ports

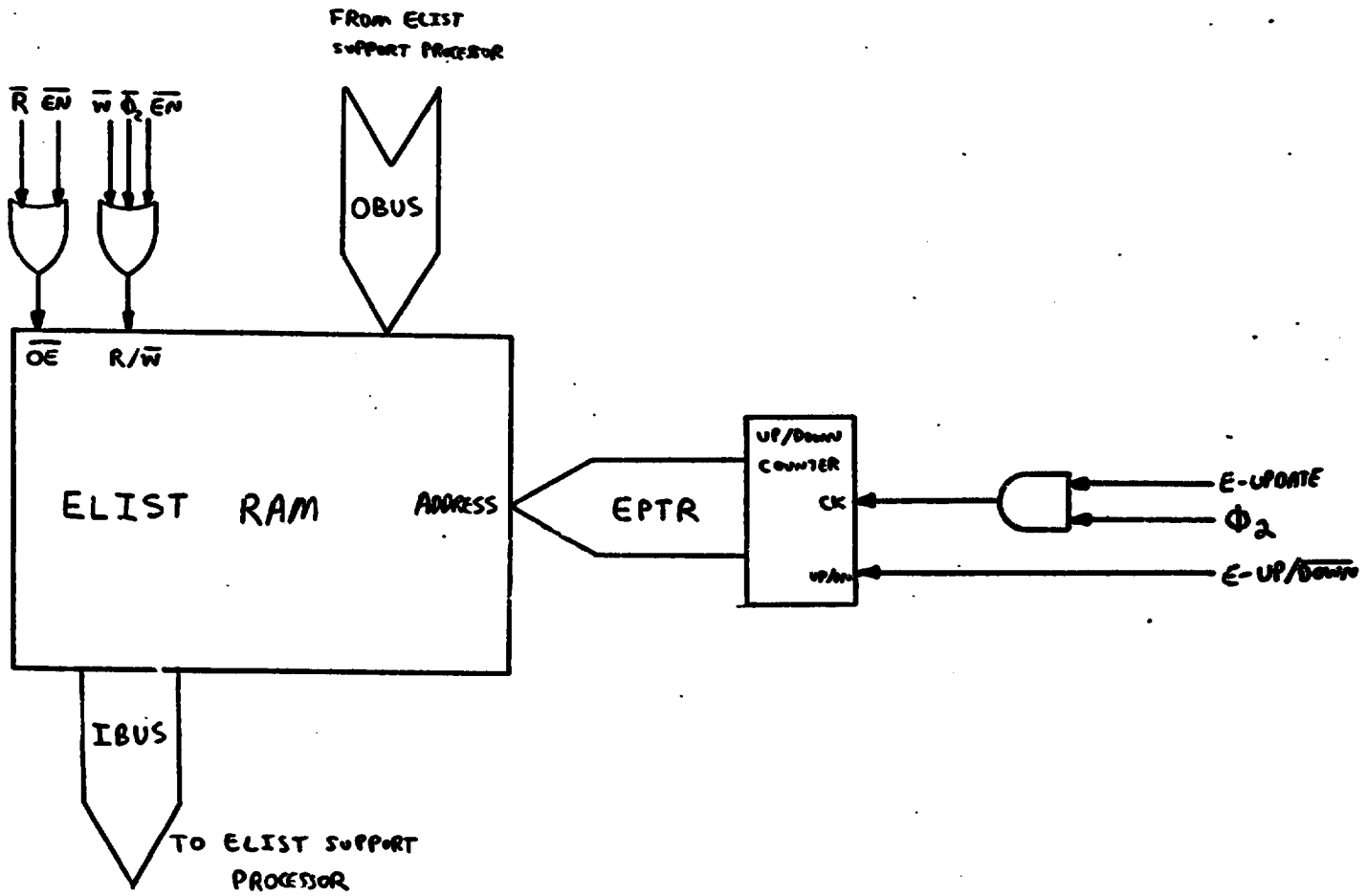


Fig. B3. ELIST RAM Structure

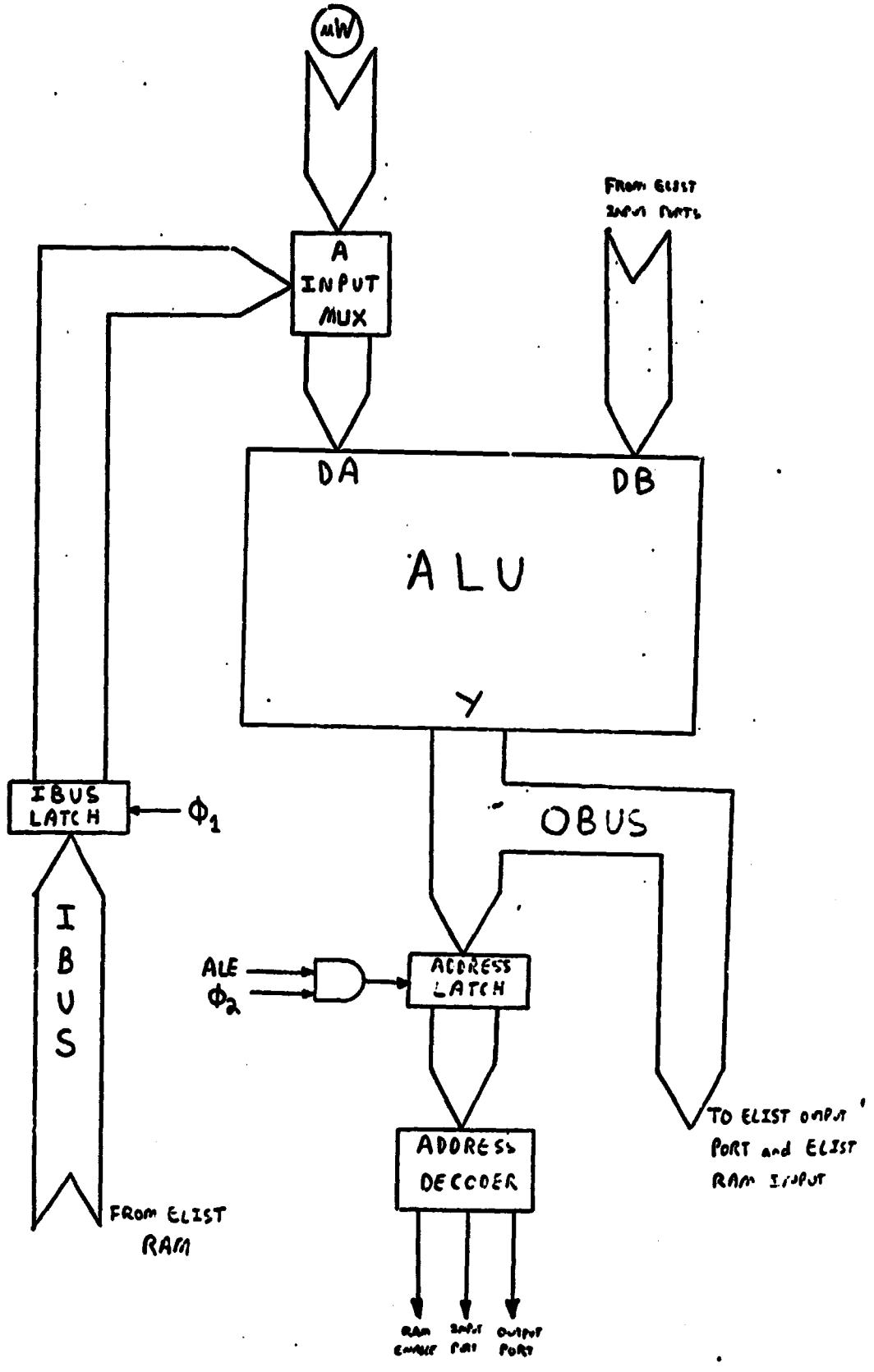


Fig. B4. ELIST Support Processor

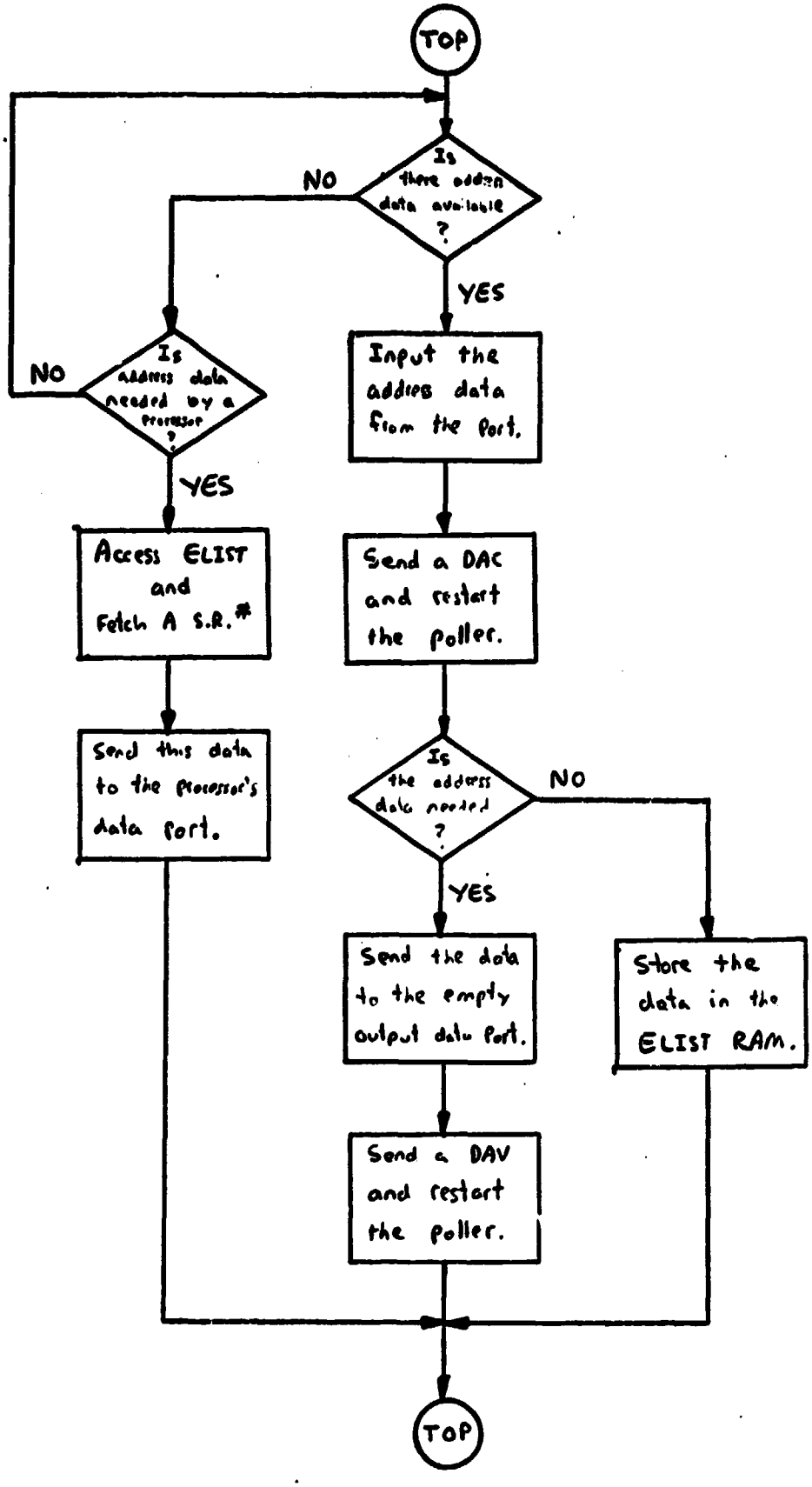


Fig. B5. ELIST Service Routine Flowchart

```
ELIST: If DAV = Ø, JMP to DAC

                                    *Is there a Shift Register num-
                                     ber to input?  NO: Jump to DAC
                                     Routine.

       [Data Port]@Input Polling Circuit → Q

                                    *YES: Input the data from the
                                     selected port.

       If DAC=Ø, JMP to STORE; send a DAC; Release Input Port Poller

                                    *Send a DAC to the input port and
                                     clear the Input Port Poller.
                                     Meanwhile, check to see if any
                                     output port requires new data.
                                     If none do, jump to the STORE
                                     Routine.

       ELIST OUTPUT PORT BASE ADDRESS → ADDRESS LATCH
       Q → Selected Output Port

                                    *If a port requires data, enable
                                     it onto the data bus and send
                                     the data.

       SEND A DAV; RELEASE Output Port Poller; JMP to ELIST

  DAC: If DAC=Ø, JMP to ELIST

                                    *Is there an output port request-
                                     ing service?  NO: JMP to ELIST
                                     Routine.

       ELIST BASE ADDRESS → ADDRESS LATCH
       [ELIST]@EPTR → Q

                                    *YES: Fetch a S.R.# from ELIST

       ELIST OUTPUT PORT BASE ADDRESS→ADDRESS LATCH; Decrement EPTR
       Q → Selected Output Port

                                    *SEND DATA and update EPTR

       SEND A DAV; RELEASE OUTPUT PORT POLLER; JMP to ELIST

STORE: ELIST BASE ADDRESS → ADDRESS LATCH; Increment EPTR
       Q → [ELIST]@EPTR; JMP to ELIST

                                    *STORE the data in ELIST
```

Fig. B6. ELIST SERVICE ROUTINE

ELIST Service Routine

a) S.R.# available as well as requested	6 cycles = 0.72 μSec
b) S.R.# required from RAM	7 cycles = 0.84 μSec
c) S.R.# stored in RAM	5 cycles = 0.60 μSec
d) No data available or required	2 cycles = 0.24 μSec

Fig. B7. Software Execution Times