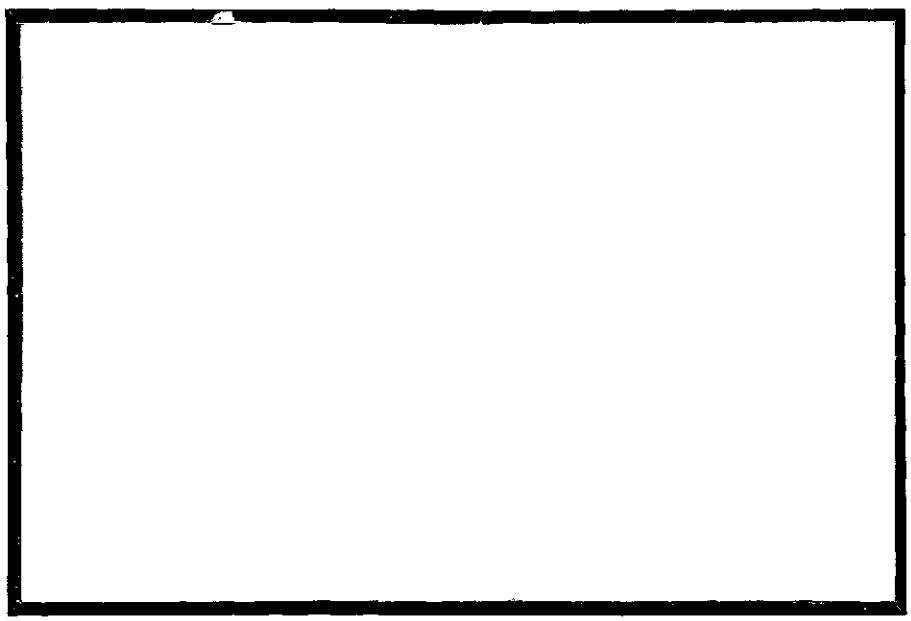
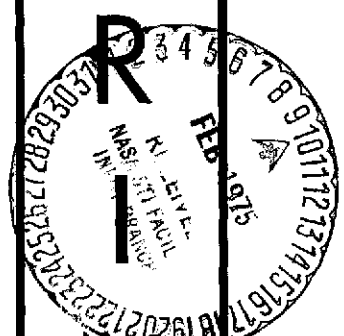


ELECTRICAL ENGINEERING



ENGINEERING EXPERIMENT STATION

AUBURN UNIVERSITY

AUBURN, ALABAMA

N75-16250

A MULTIPROCESSOR COMPUTER SIMULATION MODEL EMPLOYING A FEEDBACK SCHEDULER/ALLOCATOR FOR MEMORY SPACE AND BANDWIDTH MATCHING AND TMR PROCESSING

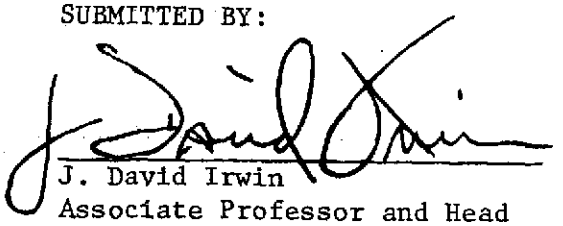
by

David B. Bradley and J. David Irwin

December 1974

Contract NAS8-26930
GEORGE C. MARSHALL SPACE FLIGHT CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HUNTSVILLE, ALABAMA

SUBMITTED BY:



J. David Irwin
Associate Professor and Head
Electrical Engineering

FORWARD

This document is a technical summary of the progress made by the Auburn University Electrical Engineering Department toward fulfillment of contract NAS8-26930. This effort was coordinated with Dr. J. B. White, National Aeronautics and Space Administration, Huntsville, Alabama.

A MULTIPROCESSOR COMPUTER SIMULATION MODEL EMPLOYING A FEEDBACK SCHEDULER/ALLOCATOR FOR MEMORY SPACE AND BANDWIDTH MATCHING AND TMR PROCESSING

David B. Bradley and J. David Irwin

ABSTRACT

A computer simulation model for a multiprocessor computer is developed that is useful for studying the problem of matching a multiprocessor's memory space, memory bandwidth and numbers and speeds of processors with aggregate job set characteristics. The model assumes an input work load of a set of recurrent jobs. A minimal amount of knowledge of individual job requirements for bandwidth is assumed. The model includes a feedback scheduler/allocator which attempts to improve system performance through higher memory bandwidth utilization by matching individual job requirements for space and bandwidth with space availability and estimates of bandwidth availability at the times of memory allocation. This matching factor is then fed back into the job scheduler via internal job priority. A nonfeedback version is made available for comparison purposes and an independent analysis is made to determine maximum improvements. The simulation model includes provisions for specifying precedence relations among the jobs in a job set. Provisions for specifying precedence execution of TMR (Triple Modular Redundant and SIMPLEX (Non Redundant) jobs. Some typical problems are studied by means of the simulator.

The amount of slowdown of a multiprocessor due to memory access conflict is determined by analytical means. A hardware processor to memory interconnection and access conflict resolution scheme is developed for a priority driven multiprocessor.

Documentation of the computer simulation model is included.

ACKNOWLEDGEMENTS

The authors express their appreciation to the following three persons for their excellent programming assistance in the preparation of this document.

Mr. Henry C. Cobb, IV
Lieutenant Marvin M. Edgeworth, Jr., USAF
Airman Richard M. Lundy, USAF

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A multiprocessor computer system is a computer system in which two or more hardware processing units simultaneously share a common memory. A generalized block diagram of the hardware configuration of a multiprocessor is shown in Fig. 1-1. Traditionally the hardware cost of the computer memory has represented the lion's share of the total hardware cost of the system. This high cost of the memory has influenced the architecture of the general purpose computer system more than any other single factor. One exemplification of this heavy influence is the hierarchy of memory found in most members of past and present generations of general purpose computer systems.

The desire to make full utilization of the relatively high cost memory of a computer system foreshadowed the arrival of the multiprocessor system. Thus, it is not surprising that this variant of computer architecture is almost as old as the concept of the stored program computer itself [7], [8], [9], [10]. However, cost of the computer system's memory was and is still not the only impetus for studying the practical implications of such systems. The multiprocessor system arises quite naturally in the realm of high reliability and high availability systems. The multiprocessor system also provides an ideal vehicle for outward expansion and for tailoring for specific applications.

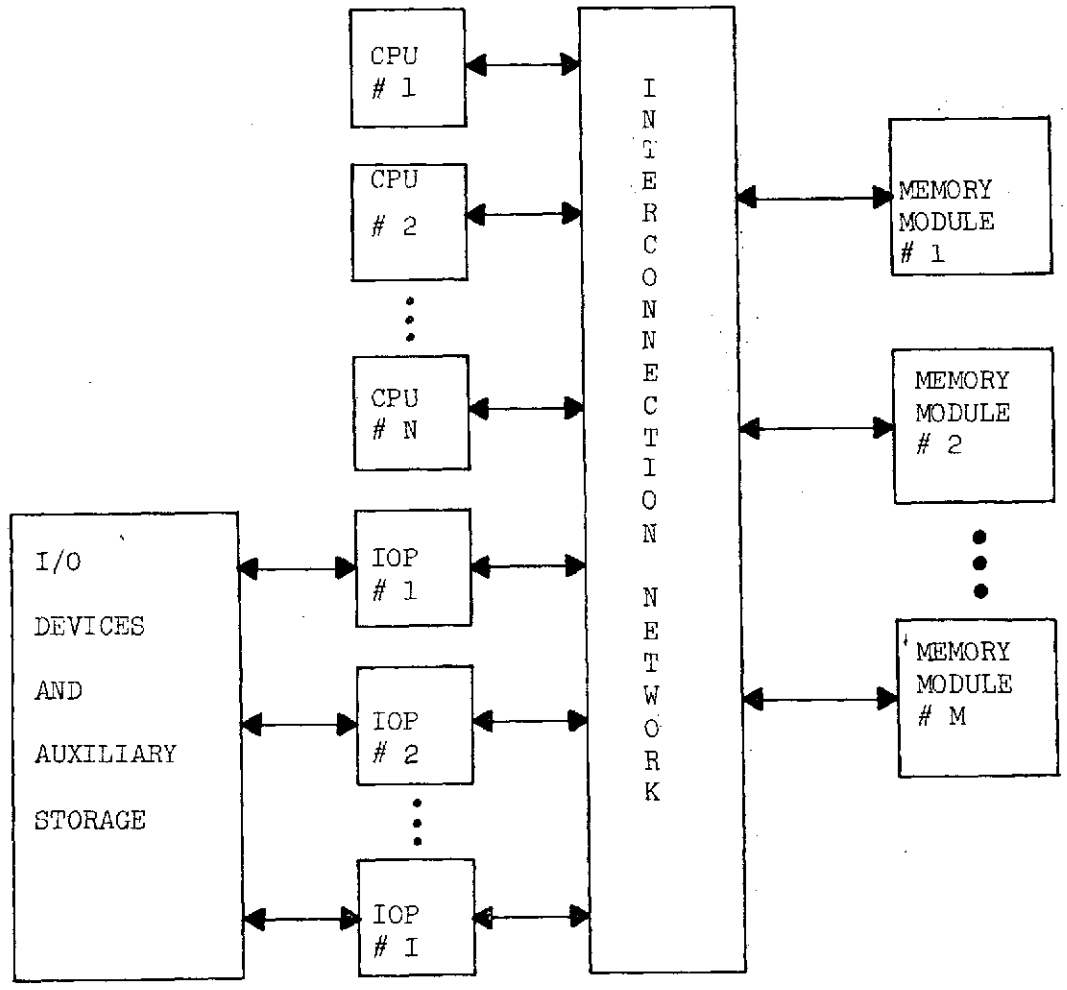


Fig. 1-1 Multiprocessor Hardware Block Diagram

Notwithstanding this considerable motivation for the development of multiprocessor systems, there have in reality been a large number of discussions in the relevant literature about various aspects of multiprocessors but very few physical implementations, at least not in view of the tremendous growth of electronic digital computer systems during the past twenty years. This striking lack of prolification of multiprocessor systems may be attributed in part to the fact that technological improvements in speed, cost, and reliability of the basic hardware components have kept pace with the vast majority of application requirements and it is only from a handful of potential applications such as space exploration, weather forecasting, airport aircraft control and missile defense systems that requirements of extreme reliability, availability and/or processing speed have exceeded that obtainable from conventional computer configurations. The inherent modularity of the system shown in Fig. 1-1 allows great flexibility in expanding a given system to meet increased needs, in tailoring for specific applications and in obtaining highly reliable systems. Another advantageous feature of this configuration is that existing applications software written for conventional uniprocessor computers may be run on a machine of this configuration without alteration. No intrinsic penalty in overall system efficiency or system reliability is incurred by this non-alteration. Thus the huge investment already made in high level applications programming is not wasted when going from a conventional uniprocessor system to a multiprocessor of the form shown in Fig. 1-1.

Difficulties with the Multiprocessor

The realization of the advantages offered by the multiprocessor configuration is contingent upon overcoming, among other things, certain operational difficulties associated with its configuration. Very broadly, these difficulties revolve around recognition and effective exploitation of parallelisms in all phases of the computer operation.

One group of these problems is concerned with scheduling and allocation of the system's resources to jobs in the system's input queue. In accomplishing these tasks compromises must be made between considerations of overall system efficiency and service to individual jobs. In this regard, the system's supervisory programs tend to become quite complex due to the large number of contingencies that may arise.

The interaction between the system's hardware processors arising from their simultaneous use of the main memory's bandwidth is also of central importance in the multiprocessor system. In a multiprocessor system several hardware processors are simultaneously using the common storage area and access contention problems arise. This tends to negate potential speed increases that the multiprocessor configuration offers. Thus the hardware interconnection between the memory and the processors can become a considerable problem due to the required complexity of the controlling mechanism and the speed at which interconnections must be made when several processors are simultaneously using the memory.

The supervisory system programs and the space that they occupy or hardware needed for their implementation represents overhead of the system and tends to nullify some of the advantages promised by the multiprocessor configuration. Thus only the simplest of algorithms should be contemplated for these applications. Indeed, the speed requirements of the processor to memory interconnection problem are so severe that hardware implementation appears to be the only reasonable approach.

Purpose of This Paper

It is the purpose of this paper to investigate some of the problems associated with the multiprocessor such as job scheduling, memory allocation, memory access conflict under both non-bandwidth limiting and bandwidth limiting (i.e. localized bandwidth limiting and mismatch) and the processor to memory module interconnection problem. In view of the importance of the modular multiprocessor to high reliability systems, problems concerned with use of a multiprocessor as a TMR processor will also be investigated. Some of these problems are of a nature that analytical techniques are invalid due to the gross simplifications required to make them manageable. These simplifications cause the results to be of questionable value when applied to practical situations of much broader scope than the analytical model. Thus, for some of these problems, a computer simulation model is the primary instrument through which these investigations are carried out. Actually the development of such a model consumes a

considerable portion of the paper. This development provides insight into specific problem areas as well as an overview of how the interactions of many of the specific problems affect the complete system. The model then becomes more than an intermediary to some specific end. For convenience, and due to the fact that it is believed that most potential application areas for the modular multiprocessor lie in this direction, the model assumes that the input job stream is like that of a real time centralized process control computer or a centralized space borne computer, i.e., the input job stream consists of a fixed set of recurrent jobs of either the periodic or aperiodic type. Thus a minimal amount of knowledge concerning bandwidth requirements of each job may be assumed.

Other Studies Related to the Topic of this Paper

The following topics are a small sample of those that have been and are currently being discussed in the literature. Some of the papers from which the topics were extracted had a considerable influence on the direction taken in this paper. Others tend to point out alternative approaches or place different emphasis on those areas of general agreement and thereby help to put the present work in proper perspective.

The topics are subdivided into the following categories: Problem areas common to parallel processing and parallel processors, high and low level scheduling, task graphs, criterion for parallel processing, parallelism recognition, and dynamic memory allocation.

Problem Area Survey

Lehman [22] has summarized a number of broad problem areas that are common to parallel processing and parallel processors in general. Both hardware and software problems were surveyed. The software problems include requirements of high level languages in the recognition, and/or provision for expression, of the parallel structure of computer programs at this level as well as the additional burdens placed on the executive portion of the system in the areas of communication and interaction between the parallel segments of the program ensuing from effective exploitation of the recognized or expressed parallelisms. Hardware problems surveyed include the processor to memory interconnection problem. The paper focuses on bringing to light the vast potential that parallel processing systems have in influencing the future development of the field of computing that is a result primarily of their amenability to improvements in availability, reliability, efficiency, expandability and performance to cost ratio. The results of simulation in a multiprocessor environment of certain applied mathematical problems were included. The paper has an extensive bibliography that traces the inception and solution development of many of the problem areas up to the point in time of the paper.

High and Low Level Scheduling

Lorin [15] has classified the scheduling function into two major parts: High level scheduling and low level scheduling. Operating systems that are high level dominated have traditionally

been called batch mode or service oriented. Those that are predominantly low level are real time, time shared, or resource oriented.

The primary function of the high level scheduler is to act as an interface between the operating system and its operational environment. It does this by selecting from the list of jobs that have entered the system and are candidates for activation the next job(s) to be passed to the active list. As each job enters the system a profile of its major attributes such as priority, urgency, initiation time, deadline and precedence relations are stored in a table. It is primarily on the basis of these parameters that the selection is made. Other inputs to the high level scheduler include initiation signals which are triggered by a time clock or events such as new job arrivals, job completions, I/O completions, release of key resources, and system error. The output of the high level scheduler consists of a partially ordered list in which are reflected the demands placed on the system in absolute and relative time by the input job stream.

The function of the low level scheduler is to dynamically allocate time resources to jobs in the active list. The low level scheduler insures that jobs in the active list receive service at rates consistent with assigned external priorities and that the available resources are used in a semi-optimum fashion. In the performance of its function the low level scheduler has a central problem; task switching. In general task switching occurs when the current task in control of the CPU is unable to continue due to a need for I/O service or service from some other asynchronous task. Another but somewhat more subtle difficulty

with which the low level scheduler must contend is the fact that there are incompatabilities between resource balance and external priority. A basic concept for resource allocation is to grant high priority to I/O bound jobs so that channel and peripheral device utilization is high and assign lower priority to CPU bound jobs so that the CPU bound jobs tend to run during the interrupted times of the I/O bound jobs. This concept is the thesis of multiprogramming. Even when priority is used in a resource balancing scheme it is usually only responsive to an average of the I/O or CPU boundedness of the job and is not responsive to run to run variations or to the fluctuations occurring during the course of a particular run.

Task Graphs

Schedules for jobs based on task or computation graphs for execution by multiprocessors have been studied by Hu [16] and by Ramamourthy, Chandy and Gonzalez [17], [18]. A task graph or computation graph is a partially specified directed graph whose nodes denote some entity of processing to be done and whose arrows are used to represent precedence relations among the entities. The processing time and memory space requirement for each node may also be shown on the graph. Schedules based on these graphs determine the sequence in which to process the various tasks of the graph in order to complete the graph in the minimum amount of time as well as the minimum number of processors required, or given that n processors are available a schedule is determined so that the graph is executed in the minimum

amount of time. Generally such schedules do not allow preempting of tasks nor do they consider memory contention problems. Muntz and Coffman [19] consider the two processor case in which preempting is taken into consideration. Graham [20] and Manacher [21] have studied an aspect of schedules based on task graphs known as Richards' anomalies which were first noticed by P. Richards in 1960. Briefly, Richards noticed that a slight decrease in the execution time of certain tasks within a task graph could cause considerable increases in the overall execution time of the task graph.

Criterion for Parallel Processing

Bernstein [23] has studied a method for determining whether two segments of a program are executable in parallel by observing during the execution of each segment how memory locations used by each segment are treated. In this regard four cases were identified. The four cases are as follows:

(1) The location is only fetched during execution of segment s_i.

(2) The location is only stored during execution of segment s_i.

(3) The first operation on this location is a fetch and subsequent operations of segment s_i stores into this location.

(4) The first operation involving this location is a store and subsequent operations of s_i fetches this location.

The parallel processability of the various segments of the program is determined from logical unions and intersections of the above four classes of memory locations for each program segment s_i.

Parallelism Recognition

Reigel [27] has studied parallelism in computing systems and has identified six levels at which parallelisms capable of being exploited occurs. The six levels identified are as follows:

(1) Between independent jobs

(2) Between groups of statements (tasks) within a job

(3) Between statements

(4) Within expressions

(5) At machine language instruction interpretation level

(6) At machine bit level

Parallelisms between independent jobs and between tasks within a job are exploitable by multiprocessors having only a small number of processing units as well as parallel processing systems having a large number of processing units. A task graph may be used to provide the parallel processing system with sufficient information for efficient exploitation at the task level but some preprocessing is required to obtain the task graph. Parallelisms between statements are more readily exploitable by array processors. An example of this level of parallelism is contained in the two statements x = A + C * D and y = B + C * D. Another example is each statement in the range of a DO loop. Parallelisms within expressions are readily exploited by pipe line processors. For example the expression A + B + C * D would require 2 add periods and 1 multiply period on a serial processor but it may be accomplished in the time required for one add period plus the larger of an

add or a multiply period if an adder and a multiplier are simultaneously available. There is the possibility of exploiting parallelisms at the machine instruction interpretation level on any type processor since all the phases of instruction fetch, decoding and execution are not strictly sequential, especially with microprogrammed processors. Parallelism at the machine bit level is also exploitable with any type processor and was probably the first to be utilized. These parallelisms include widths of data paths and registers in the machine. All these levels of parallelisms are explored in considerable detail by Lorin [15].

Dynamic Memory Allocation

Randell and Kuehner [24] have associated four aspects of dynamic memory allocation that, in some manner, are descriptive of the capabilities and fundamentals of present hardware oriented dynamic allocators with the concepts of name space, predictive information, artificial contiguity, and uniformity of units of storage allocation.

Name space refers to a set of names that a program can use to refer to items of information. The simplest name space consists of the set of integers 0, 1, 2, ..., n used as absolute addresses in the main memory. The name space can also be divided into a number of segments with each segment composed of a linear name space. There is no implied ordering of the segment names although this may be done in which case the segment name space is said to be linearly segmented. If there is no ordering among the segment names the segment name space

is said to be symbolically segmented. The sizes of the various segments need not be the same nor does the size of a particular segment necessarily remain constant during the execution period of some program having space in the segment. Thus the segmented name space differs from the generalized notion of a two-dimensional matrix, although it is sometimes referred to as a two-dimensional name space. The major disadvantage of the segmented name space is the complexity of actually addressing the physical storage linearly. Its major advantages are its added utility and convenience in dynamically allocating storage, movement of information between levels of storage, protection and sharing among programs, error detection in addressing, and in relieving the programmer of the responsibility of name assignment.

Predictive information refers to methods for predicting the probable use of storage over a relatively short period of time in the near future. These predictions may be explicitly incorporated by the programmer, discovered by a compiler, or made by observations by the system of previous utilizations. This information may be used to determine when to bring new blocks of storage into main memory and which blocks already in the main memory have the lowest probability of being referred to in the near future and thus may be replaced.

Artificial contiguity refers to the concept of mapping a contiguous name space into a non-contiguous address space. This is accomplished by providing an automatic mapping mechanism between points in

programs at which names are referred to and the system's hardware for accessing a corresponding absolute address. The mapping scheme is invisible to the program that refers to the name. The most frequent use for such schemes is to disguise or artificially extend the apparent size of the system's physical main memory space. Systems employing this technique are often referred to as virtual storage systems. The major disadvantages of the use of a mapping scheme for provision of artificial contiguity are its cost and reduction of addressing speed.

A dynamic memory allocation scheme may allocate space in fixed size blocks (a single size or several sizes) or an attempt may be made to make the size of allocation just equal to the size needed by the information. In the latter case the memory space tends to become fragmented into many small pieces with an attendant high overhead for linking them together or a lot of shuffling of blocks of information so as to consolidate the small blocks back into larger more useful blocks. If this linking or relocating of blocks of information is not done then the net result may be that a portion of main memory may be unusable for a certain percentage of time. The extent of this unusability depends, to a considerable degree, on the ratio of the average block size to that of the total memory size. The technique of dynamically allocating memory only in fixed size blocks is exemplified by paging systems in which the program usable portion of main memory is broken up into page frames all of the same size. Hardware then auto-

matically performs placement of pages as they are referenced by the program during execution. This form of dynamic memory allocation has been discussed by Denning [25]. Various replacement algorithms have been studied by several people including Belady [26]. In such systems the actual placement of blocks is made trivial once the replacement algorithm has decided which page to push out if this is necessary. The actual fragmentation of memory does not occur with paging systems, but the effect of fragmentation (i.e. time-space product waste) does occur since a given program may be occupying many pages in memory while waiting on the transfer of a page following a page fault. Fragmentation can also occur within pages since many page frames may be only partially used.

II. A DESCRIPTION OF THE MODEL

Cursory Discussion

The computer program of the model generates, schedules, allocates memory to, assigns processors to, and executes sets of jobs whose parameters are controlled by data and commands from the input stream. It also prints out statistics concerning service to individual jobs and utilization of all system resources. The execution proceeds in a discrete manner with some assumed number of basic clock periods per simulation step. This number is a variable of the system.

Two types of memory allocators are used. One is a so called 'optimum-fit' algorithm [11] in which memory allocations are made on the basis of the number of available blocks and the requirements of the job. The second allocator is an adaptation of the 'best-fit' algorithm [12] but also utilizes information concerning the bandwidth of the available blocks of memory. The goodness of fit in bandwidth and space is fed back into the job scheduler for determination of priority with this second method.

Under both types of memory allocators an allocation is assumed to be in one contiguous block. Allocations for TMR jobs are made in triplicate from three sets of disjoint memory modules. Although the memory consists of M modules all of the same size and speed, it is viewed as one contiguous block in so far as the allocation routines are concerned.

The system utilizes a form of community multiprogramming in which both TMR as well as SIMPLEX jobs may be in progress simultaneously. The entire system is priority driven. A schedule in this system does not consist of an explicit assignment of resources for predetermined time intervals to particular jobs, but rather, consists of the more flexible priority ordering of jobs based on terms concerned with external priority, amount of time spent in the job queue, nearness to some target time, and memory preference. Interrupts for I/O operations are included in the model. Job execution proceeds in essentially two phases as follows:

Phase 1--A number of memory access requests is generated for each active job from a normal distribution with mean and standard deviation determined by the speed of the processor and some assumed average number of processor cycle periods per memory access which is based on the particular characteristics of the job.

Phase 2--Some portion of the number of memory accesses requested by each job is granted to each job. This portion is based on the number available from the job's memory space, the priority of the job relative to that of any other active jobs having space in the same modules and the number of memory accesses actually requested by the job.

A job is considered to be active at a particular simulation step if it has been scheduled and has all the resources that it needs for processing

at that particular step. Scheduling is done in small batches and is triggered by comparing short term system utilization averages with long term utilization averages as well as considerations of instantaneous system and job queue status. The input job stream is assumed to be like that of a centralized process control computer or a centralized space borne computer, i.e. the jobs consist of a fixed set (fixed for the duration of a particular run) of which some are executed on a periodic basis while others are executed aperiodically. The system has provisions for handling precedence relations among the jobs in the input job set. An estimate of the bandwidth requirements for each job is assumed to be available for use with the feedback scheduler/allocator.

Block Diagram Description

A simplified block diagram of the model is shown in Fig. 2-1. The main program accepts commands and data from the input stream. At the start of a sequence of simulation runs it invokes a set of routines labeled Job Generator which generates a set of jobs with characteristics specified by the data from the input stream and places these jobs into a bulk storage table. Upon command from the input stream a specified selection of these jobs is loaded into the job status and control table. The main program then calls the High Level Scheduler which performs a limited amount of pre-processing on the set of jobs in the job status and control table. The High Level Scheduler then calls either the Future Events Chain Manager or the Current Events Chain Manager for

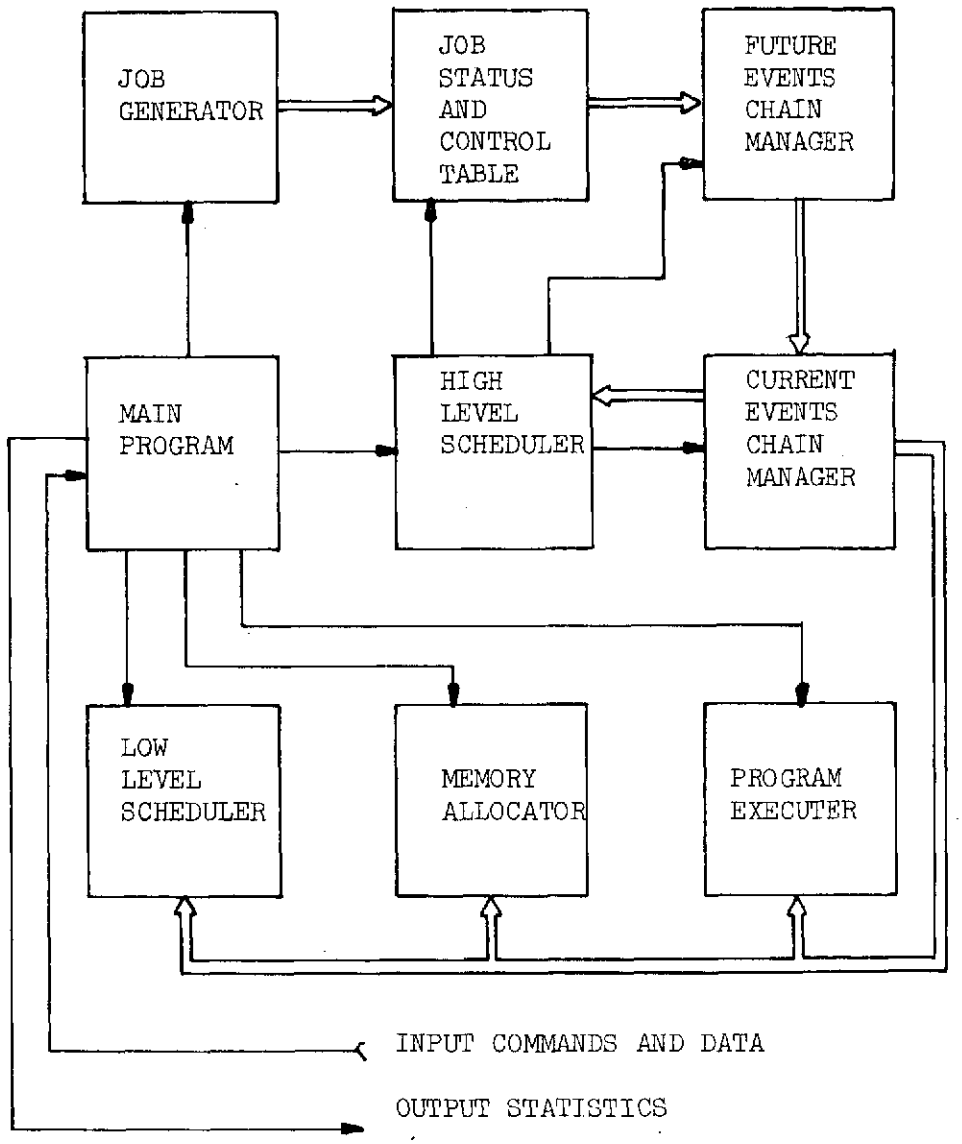


Fig. 2-1 Simplified Block Diagram of Simulation Model

placement of those jobs for which a target time could be set up during the initial pre-processing into either the Future Events Chain or the Current Events Array depending on the nearness of this target time to the present time. Control is then returned to the Main Program. If the non-feedback scheduler/allocator is specified in the input data then the Main Program calls the Current Events Chain Manager for chaining the jobs in the Current Events Array together according to their internal priority. Control is then returned to the Main Program where upon the Memory Allocator is called for allocation of main memory space to as many of the jobs in the Current Events Chain as is possible. Control is then returned to the Main Program. If the feedback scheduler/allocator is specified then the main program first calls the Memory Allocator for reserving space for as many of the jobs in the Current Events Array as possible. It then calls the Current Events Chain Manager for chaining together all the jobs in the Current Events Array according to their internal priority. In this manner the goodness of fit in both space and bandwidth influences the internal priorities of the jobs in the Current Events Array which are allocated memory. Control is then returned to the Main Program. In either of the above cases after memory has been allocated to as many of the jobs in the Current Events Array as possible and the jobs have been chained together according to priority the main program calls the High Level Scheduler for an initial assignment of Processors. After this initial assignment of processors the Program Executer is called for one step of execution processing for all active jobs.

The Program Executer performs an execution processing step by generating a number of memory access requests for each active job and then granting some portion of this number of requests to each of the active jobs. After granting memory accesses the Program Executer updates the completion counters for each of the active jobs and checks for I/O interrupts and processing completions. If an I/O interrupt point has been reached or a job has completed the Program Executer calls the Low Level Scheduler for reassignment of processors. Before returning to the Main Program the Program Executer updates data arrays containing information concerning the utilization of system resources for the present step of execution processing and service to individual active jobs. After returning to the Main Program following a step of execution processing the Main Program checks to see whether a new schedule should be initiated or whether or not the present run has been completed. If neither of these events is to occur or has happened then the simulation clock is advanced by one and the Future Events Chain Manager is called to remove any jobs from the Future Events Chain whose next event time has been reached. If a new schedule is to be initiated then the Memory Allocator and Current Events Chain Manager are called to perform the necessary processing. Processing continues in this manner with control alternating between the Main Program and Program Executer until the run has been completed or until no active jobs are in the Current Events Chain. In case the run is not completed but no active jobs are in the Current Events Chain the simulation clock

is advanced to the time of the leading event in the Future Events Chain provided such an event exists.

In case there is no such event in the Future Events Chain the simulation run is abnormally terminated and no further runs can be initiated. In case a simulation run completes normally the statistics for the run are printed and another run may then be initiated by a command from the input stream.

Let us now consider some of the topics that were mentioned or implied above in more detail.

Granularity of Simulation Time

Each step of simulation corresponds to a large number of CPU cycle periods or alternately a smaller number of memory module cycle periods. This allows a simulation run to proceed in a reasonable amount of real time. However, there are bounds on the amount of granularity that can be tolerated if meaningful results are to be obtained. Specifically, peak demands on memory access requests tend to become filtered by averaging during each simulation step. Thus very little slow down of the overall system due to memory access conflict that would arise in a real situation due to two or more processors simultaneously executing programs from a common memory shows up in the simulation. Also, bandwidth utilization tends to be higher with the simulator than it would be with a real system due to this same cause. If I/O interrupts are frequent in each job or if the average number of simulation steps per job is small then there tends to be proportionally

large amounts of wasted simulation time because the step in which the interrupt or completion occurs is on the average only half used by the job for which the event occurs. The memory bandwidth that the interrupted or completing job would have claim to can be redistributed to any other active jobs having space in the same modules but processors are not reassigned, at least not for further processing, until the end of the simulation step. Thus processors assigned to interrupting or completing jobs are actually only partially used during the step in which the interrupt or completion occurs.

From the above discussion we may conclude that there are necessary constraints on the relationships between the granularity of simulation time, mean processing speed of jobs, frequency of interrupts and average total number of memory accesses required per job for completion. On the one hand the number of memory module cycle periods per simulation step should be small in order to more closely approximate a real system but on the other hand it should be large in order to keep the amount of real processing time within reasonable bounds. Thus the number of basic clock periods per simulation step is made a parameter of the system.

The Internal Priority Equation

The scheduling and allocation of system resources to individual jobs is priority oriented. Priority provides a convenient, compact, and versatile vehicle for driving the system. It may be used to express both the absolute and relative demands placed on the system by the

input job stream. The priority equation used by the model has the following form:

$$INP = MIN(MAXPR, W_1*EXP + W_2*NDLN + W_3*IWAIT + W_4*MAPF)$$

where INP represents internal priority, MAXPR is the maximum priority allowed, EXP is an externally declared priority, NDLN is a measure of nearness to some target time or deadline, IWAIT represents the amount of time that the job has spent in the ready queue and MAPF is a measure of how close the requirements of the job for space and bandwidth match that available from the block from which its main memory space was obtained. The variables W_1, W_2, W_3 and W_4 are weighting factors for their associated terms. The exact meanings of the terms NDLN, IWAIT and MAPF are as follows:

$$NDLN = MAX\left(0, V_1 - \frac{|\text{present time} - \text{target time}|}{\text{estimated minimum processing time}}\right)$$

$$IWAIT = \begin{cases} MAX\left(0, \dfrac{1}{V_2 * (1 + X)}\right) & \text{if } 0 < V_2 \leq 1 \\ V_2 * X \quad \text{if } V_2 > 1 \text{ and } X \leq 1 & \\ V_2 * (1 + .5X) \quad \text{if } V_2 > 1 \text{ and } x > 1 & \end{cases}$$

$$\text{where } X = \frac{\text{time in ready queue}}{V_1 * \text{min processing time}}$$

$$
\text{MAPF} = \left[\begin{array}{l} 0 \quad \text{if IFBK} = 0 \\ \\ \text{Max over available blocks at time of schedule} \left(\begin{array}{ll} \text{DSM} * \text{MMSPF} & \\ + & \\ \text{MMAPF} * \text{DUM} & \text{if DUM} \leq 1 \\ \text{or MMAPF/DUM} & \text{if DUM} > 1 \end{array} \right) \Big/\!\Big/ \left(\begin{array}{l} \text{previously selected blocks at present schedule} \end{array} \right) \\ \\ \qquad\qquad \text{if IFBK} = 1 \end{array} \right]
$$

where $\text{DSM} = \dfrac{\text{MMS}}{\text{BLK SIZE}}$

$$
\text{DUM} = \frac{\text{estimated available bandwidth of space in block occupied by job}}{\text{estimated mean bandwidth required by job}}
$$

MMS = Main memory space required,
MMSPF = Memory space preference factor
MMAPF = Memory access preference factor
V_1, V_2 and IFBK are all variables of the system

From the priority equation we may observe that the scheduler utilizing this equation as its activating mechanism may be made to take on many forms. For example, if W_1, W_2, and W_4 are small and W_3 is large with $V_2 > 1$ the scheduler tends to operate according to the first in first out discipline. If $0 < V_2 \leq 1$ then it works like a last-in first-out scheduler. If W_1, W_3, and W_4 are small but W_2 is large then the scheduler is similar to a deadline scheduler, etc.

I/O Interrupt Mechanism

The I/O interrupt mechanism is facilitated by assigning one of a limited number of normalized processing curves to each job at the time of job generation. There are no specific requirements for these

curves but they should loosely approximately the manner in which computer programs tend to operate in regard to the points in the program at which the I/O processing occurs and the duration of these interrupts. Many computer programs initially read a number of variables and arrays into the main memory, perform some operations on these variables and arrays, and finally output some data. Although no figures are available, it is believed that the initial read operations usually involve a larger number of operations. Thus most of the processing curves used by the model assumes on the order of two to four times as much I/O processing near the beginning of the curve as near the end. The program which calculates the expected minimum processing time for each job during the initial preprocessing phase immediately before a simulation run assumes that the number of main-memory accesses between interrupts is from a normal distribution for both CPU and IOP processing. Therefore, closer estimates of the minimum processing time can be expected if these numbers of accesses are actually from a normal distribution.

The manner in which these curves are actually used to obtain the I/O interrupts is as follows: In the job status table for each job is a completion count word and a next interrupt point word. The contents of the next interrupt point word are updated with the number of memory accesses required before the next interrupt at the beginning of program execution and immediately after each interrupt. This is done by multiplying the total number of CPU memory accesses or IOP memory

accesses, as the case may be, for the job under consideration by the abscissa points of the normalized processing curve at which interrupts occur. The smallest such product that is greater than the old next interrupt point is the new next interrupt point. The Program Executer then compares the completion count with the current next interrupt point word after each step of program execution in order to determine when the interrupt point has been reached.

Job Switching

Multiprogramming is a technique for improving system efficiency in which two or more jobs are concurrently executed, i.e., second job, third job, etc., is started before the first job completes. Thus, when a job interrupts for I/O processing the CPU may be reassigned to some other job, thereby avoiding waste of CPU time. The major purpose of the Low Level Scheduler is to manage the system's processor time resources. It does this by job switching in a multiprogramming environment. In conjunction with job switching is a preempting scheme in which jobs wanting a CPU may preempt lower priority jobs that have a CPU. Due to the assumed nature of the bulk storage medium that the IOP's operate with no preempting of IPOs is allowed. The option is also available whereby jobs interrupting for I/O processing may voluntarily return their CPU(s) to an available pool or retain the CPU(s) during the I/O processing depending on their priority and the number of jobs in the job queue. The allowed preempting is as follows:

1. A TMR job may preempt the lowest priority TMR job among the three lowest priority jobs of lower priority.

2. A TMR job may preempt the three lowest priority SIMPLEX jobs of lower priority provided no lower priority TMR job is among the three lowest priority jobs.
3. A SIMPLEX job may preempt the lowest priority SIMPLEX job of lower priority.
4. A SIMPLEX job may preempt the lowest priority TMR job of lower priority provided there is no lower priority SIMPLEX job.

Upon voluntarily giving up CPU(s) at I/O interrupt initiation the Low Level Scheduler looks for other jobs for reassignment of CPU(s) to. The following scheme is in effect.

Case 1. The interrupting job is TMR.

All three of the interrupting job's CPUs are assigned to the highest priority TMR job among the three highest priority jobs that want CPUs provided such a TMR job exists, otherwise as many of the CPUs as possible are assigned to the highest priority SIMPLEX jobs that want a CPU. Any remaining unassigned CPUs are placed in an idle state.

Case 2. The interrupting job is SIMPLEX.

1. The highest priority job that wants a CPU is SIMPLEX.

   The interrupting job is assigned to the highest priority job that wants a CPU.

2. The highest priority job that wants a CPU is TMR.

   The interrupting job's CPU is assigned to this TMR job provided there are two other idle CPUs that may also be

assigned to this TMR job or there is one idle CPU and at least one simplex job of lower priority that may be preempted of its CPU. If there is no idle CPU then two lower priority SIMPLEX jobs that may be preempted of their CPUs will suffice, otherwise, the interrupting job's CPU is assigned to the highest priority SIMPLEX job that wants a CPU if one exists. If all the above tests fail then the CPU is placed in an idle state.

A similar scheme is used upon completion of I/O processing in which IOPs are to be reassigned except that, as already mentioned, no preempting of IOPs is allowed.

Methods for Triggering New Schedules

The frequency of and points in time at which new schedules are initiated is of great importance in regard to system efficiency as well as service to individual jobs. If new schedules are too frequent then the amount of overhead attributed to schedule processing can become a considerable portion of total system processing. Conversely, if new schedules are too sparse or do not occur at the correct points in time then system thruput decreases. The model attempts to detect when a new schedule should be initiated by comparing short term running average CPU, IOP, and memory bandwidth utilizations with long term averages in conjunction with instantaneous conditions of job queue status and memory space availability. In order to facilitate a better understanding

of this let us make a few definitions.

Definition: Processor step utilization (PSU)

$$PSU = \frac{1}{NPROS} \sum_{i=1}^{NACT} \frac{NMAG(i)}{NMAR(i)}$$

where NPROS is the total number of processors, NACT is the number of active processors, NMAG(i) is the number of memory accesses granted to the i^{th} active processor and NMAR(i) is the number of memory accesses requested by the i^{th} active processor.

Definition: Bandwidth Step Utilization (BWSU)

$$BWSU = \frac{NMAG}{NMAA}$$

where NMAG = number of memory accesses granted

and NMAA = number of memory accesses available.

The short term running average utilizations are obtained as follows:

$$RPU_I = RPU_{I-1} * \frac{NSTEP-1}{NSTEP} + PSU/NSTEP$$

$$RBU_I = RBU_{I-1} * \frac{NSTEP-1}{NSTEP} + BWSU/NSTEP$$

where RPU_I and RBU_I are running processor and bandwidth utilizations at step I and NSTEP is a variable of the system which essentially adjusts the number of previous steps over which the average is taken. This is not the same type of running average that is used in statis-

tical work but it serves the purpose here of smoothing step fluctuations and requires storage of only the current estimate. Thus the new schedule equation may be of the form

$$\text{NEW SCHED} = \text{TSLS} \cdot [\text{AVMS} \cdot \text{NJMT} + (\text{STCU} + \overline{\text{STCU}} \cdot \text{NICU}) \cdot (\text{STIU} + \overline{\text{STIU}} \cdot \text{NICU}) \cdot (\text{SCWM} + \overline{\text{SCWM}} \cdot (\text{WMNS} + \overline{\text{WMNS}} \cdot \text{NWMNS}))] \qquad 2.2$$

where all the variables are Boolean variables and have the following meanings.

TSLS =
- 1 if the time since last schedule > N
- 0 otherwise

AVMS =
- 1 if the amount of available space is $\geq$ one-half the total memory space
- 0 otherwise

NJMT =
- 1 if the number of jobs waiting memory is $\geq$ 1/2 the number of scheduled jobs
- 0 otherwise

STCU =
- 1 if short term CPU utilization is < TH1 * long term CPU utilization
- 0 otherwise

NICU =
- 1 if the number of idle CPUs is > 1/2 the number of CPUs
- 0 otherwise

STIU =	1	if short term IOP utilization is < TH1 * long term IOP utilization
	0	otherwise
NIOU =	1	if the number of idle IOPs is > 1/2 the number of IOPs
	0	otherwise
SCWM =	1	if the number of jobs that are scheduled and waiting memory is > 0 and the amount of free space is $\geq$ 1/4 the total space
	0	otherwise
WMNS =	1	if the number of jobs waiting memory but not scheduled is > 0 and at least 1/4 of the total memory space is available
	0	otherwise
NWMNS =	1	if there are any jobs in the job queue that are not waiting memory and have not been scheduled
	0	otherwise

and N and TH1 are system parameters.

Memory Allocation Schemes

'Best fit' and 'first fit' memory allocation methods are described by Knuth [12]. Briefly each algorithm assumes that information concerning the available blocks is maintained in a list that is linked according to starting addresses in main memory. In the first-fit scheme when an allocation of size X is to be made a sequential search

of the list is made beginning at the top of the list until the first block of size $\geq$ X is found. An allocation of size X is then made from this block and any remaining portion of the block is left in the available list, otherwise, the list is relinked around this block. The best fit method assumes that the entire list is searched and the location in the list of the best fitting block of size greater than or equal to X is retained. If such a block exists then an allocation is made from this block. An 'Optimum Fit' strategy is described by Campbell [11]. This method assumes that in addition to the available list the number N of available blocks is also maintained. Before an allocation is made an integer F(N) is calculated where F(N) $\leq$ N. The linked list is then searched through the first F(N) - 1 entries and the location of the best fitting block is retained. The search is then continued and the first block that is a better fit than any of the preceeding blocks is selected.

There are several measures of the effectiveness of the memory allocation scheme. One such measure is its ability to make an allocation in one contiguous block at any time. Under such a measure the scheme which tends to keep most of the available space in a single contiguous block tends to have as good (if not better) chance for making a large allocation at any time as any other scheme. Simulation seems to indicate that under this measure there are only marginal differences between the first-fit, best-fit and optimum-fit algorithms.

Other measures take into consideration the amount of time that is required to implement the algorithm. The algorithms would definitely be ranked first-fit, optimum-fit and best-fit if the only consideration were that of minimum time for implementation. However, even here, whether or not this difference is significant in a practical sense depends on the actual average number of available blocks of storage. In this regard, it is noted in passing that there appears to be some negative correlation between the ratio of average request size to total memory size and the average number of available contiguous blocks of storage.

The simulation model uses two types of memory allocation schemes. The first method utilizes the optimum-fit algorithm in a non-feedback scheduler/allocator. The second method uses an adaptation of the best-fit algorithm in a feedback scheduler/allocator. In this second allocation method some loose optimization in bandwidth matching is attempted by associating some estimated bandwidth requirement for the jobs to which memory is to be allocated with an estimate of bandwidth availability from the available blocks of memory at allocation time. As already noted during the discussion of the internal priority equation this goodness of fit in bandwidth as well as space also influences the schedulers via the internal priority of each job. For this reason this method of scheduling and memory allocation is called a feedback scheduler/allocator.

Mechanics of the Optimum-Fit Algorithm

Let N = number of available memory blocks.

Let MAVL be the list of available memory that are linked according to starting address in main memory.

Upon memory request, calculate an integer, F(N), defined by

$$\frac{1}{F(N)} + \frac{1}{F(N)+1} + \ldots + \frac{1}{N-1} \leq 1 \leq \frac{1}{F(N)-1} + \frac{1}{F(N)} + \ldots + \frac{1}{N-1}$$

or for $N \geq 9$, $\frac{N}{\epsilon} < F(N) < \frac{N}{\epsilon} + 2 - \frac{1}{\epsilon}$

where $\epsilon \simeq 2.718$.

Then search the linked list, MAVL, and select the first block after the $F(N) - 1^{th}$ block that is a better fit than any of the previous blocks. If no better fitting block is found after block F(N) - 1 then select the best fitting block among the first F(N) - 1 blocks, provided one was found in this group.

The rationale behind the optimum-fit strategy is explained in [11]. It should suffice here to say that the method assumes that the block sizes are uniformly distributed along the linked list and that there are on the average a number of choices from which an allocation could be made. If these statistical properties do in fact hold, the method stands a good chance of actually selecting a block that is close to the best fit as well as reducing the search time below that required by the best fit algorithm, especially if there is a large number of blocks in the list. It is shown in [11] that the probability of actually choosing the best fitting block approaches 1/2.718 with large N.

Mechanics of the Best Fit Algorithm with Feedback

Let MAVL be the list of available blocks linked according to starting addresses. Let NAML be a list of jobs waiting memory.

1) Search over all blocks in MAVL and all jobs in NAML for best combined fit in space and bandwidth.
2) Make an allocation to the job from the block for which the space and bandwidth available from the block most closely approximates that required by the job.
3) Repeat steps 1) and 2) until either all memory space has been assigned, all job requirements have been satisfied, or no more assignments are possible.

In each of the above schemes used by the model it is assumed that a small batch of jobs are to be allocated memory at the same time. Any unused portion of a block from which an allocation is made remains in the list as an available block. Also, when making second and third allocations to a TMR job these additional allocations must be made from disjoint sets of memory modules. Actually no allocation is made to a TMR job until it has been determined that three allocations can be made from three disjoint sets of modules.

Estimation of Bandwidth of Available Space

In order for the feedback scheduler/allocator to work an estimate of the bandwidth available from each memory module is required. This is obtained by means of counters which count the number of accesses made to each memory module over some period of time. In order for the simu-

lator to obtain the required estimates three one dimensional arrays are required as follows:

MAS(*)--module space registers

IBWCTR(*)--elapsed time counters

NAA(*)--available access accumulators

The number of elements required in each array is equal to the number of memory modules. In a real system these arrays would be hardware registers. The current estimate of the available bandwidth from each module is updated by the Program Executer during each step of execution processing immediately after the total number of access requests to each memory module has been determined for the processing step. The IBWCTR array is also updated at this time. The updating of the NAA array is as follows.

$$NAA(I) \leftarrow NAA(I) * (1 - 1/MAXCNT) \quad \text{if } IBWCTR(I) \geq MAXCNT \text{ and } NTR(I) \geq ICON1$$

$$NAA(I) \leftarrow NAA(I) * (1 - 1/MAXCNT) + (ICON1 - NTR(I))/MAXCNT \quad \text{if } IBWTR(I) \geq MAXCNT \text{ and } NTR(I) < ICON1$$

$$NAA(I) \leftarrow NAA(I) * (IBWCTR(I))/MAXCNT + ((MAXCNT - IBWCTR(I)) * ICON1 * MAS(I)/(MAXCNT * MODSIZ) \quad \text{if } IBWCTR(I) < MAXCNT \text{ and } NTR(I) \geq ICON1$$

$$NAA(I) \leftarrow NAA(I) * (IBWCTR(I)/MAXCNT$$
$$+((MAXCNT - IBWCTR(I) * ICON1 * MAS(I))/MAXCNT * MODSIZ)$$
$$+ (ICON1 - NTR(I))/MAXCNT$$
$$\text{if } IBWCTR(I) < MAXCNT$$
$$\text{and } NTR(I) \geq ICON1$$

where MAXCNT is a parameter of the system that effectively sets the number of processing periods over which the bandwidth estimates are obtained. IBWCTR(I) contains the minimum of the number of processing steps since an allocation or release of space from memory module I has been made and MAXCNT, NTR(I) contains the total number of accesses requested from module I at the processing step under consideration, and ICON1 is the number of memory module cycle periods per simulation step.

Estimation of Job Bandwidth Requirements

Estimates of job bandwidth requirements are based on some assumed known mix of the types of instructions to be executed by the job as well as the speed of the system's hardware processors. Actually only two types of instructions are assumed for these calculations. The estimated bandwidth requirements are computed as follows:

$$MNAR = ICON/(ITA + IPCT * (NPCS * MIX + NPCL(1 - MIX)))$$

where ICON is the number of processor cycle periods per simulation step, ITA is memory access time, IPCT is the processor cycle period time, NPCS is the number of processor cycles per short instruction, NPCL is

the number of processor cycles per long instruction, and MIX is the proportion of short instructions that the job performs.

Processing Scenario

In order to clear up any ambiguities that may have inadvertently arisen from the discussions of the preceeding parts of this chapter let us consider the sequence of events that take place during a simulation run following job generation and initial preprocessing. In this discussion assume that the non-feedback scheduler/allocator is employed and only SIMPLEX jobs were generated.

(1) A schedule is obtained whereby the jobs that are ready for execution are ordered according to the priority scheme.

(2) Memory is reserved for as many of the jobs that were scheduled in step (1) as is available from the memory space.

(3) As many IOPs as is consistent with system constraints is assigned to the jobs that are in the present schedule and have memory reserved in the order of the schedule.

(4) The IOPs that were assigned in step (3) begin loading the jobs to which they are assigned and continue this loading operation until each of their respective jobs has been loaded into memory, whereupon they are reassigned to other jobs or else placed in an idle condition until some job requires their service.

(5) As soon as a job becomes loaded its status is set to wait CPU.

(6) A search is then made for a CPU for assignment to this job. The first idle CPU is assigned provided one is found, otherwise a search is made to see if a job of lower priority has a CPU. If the latter condition exists then the lowest priority job having a CPU is preempted and its CPU is reassigned to the higher priority job.

(7) The job obtaining the CPU then begins program execution and continues until it either completes execution, reaches an I/O interrupt point or is preempted by a higher priority job. In case of reaching an I/O interrupt point in its program the CPU may either be retained by the job if its priority is sufficiently high and no other jobs are currently waiting for a CPU or a search may be made for some other job needing the service of a CPU. If in the latter case no job is found waiting for a CPU then the CPU may be placed in an idle condition and remain so until some job has a need for its service. Having in some manner disposed of the CPU that was assigned to the job under consideration, a search is then made for an idle IOP for assignment to this job. In case such an IOP is found it is assigned to and remains the exclusive property of this job until this job's current I/O processing assignment is completed, thereafter a search is made for some other job that is waiting for an IOP. If the new job

that is waiting for an IOP is not in a load condition or if the system constraint concerning the number of IOPs that may be assigned to load operations at any given time is not presently reached then the IOP is assigned to this new job, otherwise the search is continued either until a suitable new job is found or no further candidates remain. If the latter condition is the case then the IOP is set to an idle state.

(8) If the new job under consideration already has a CPU then processing continues; otherwise continuation for this job is made in step (6).

(9) The job processing described in steps (1) through (8) continues until a new schedule is triggered by system resource utilization and system status and job queue status monitors. The entire process beginning with step (1) is repeated with inclusion of any jobs that were not completed during processing of the previous schedule.

Efficiency Considerations

In order for the efficiency of a multiprocessor system as hypothesized so far in this paper to approach the efficiency that is obtainable from a uniprocessor, i.e., a special case of the multiprocessor having only one CPU, certain relationships must exist between the distribution of memory space and bandwidth requirements of the input job set and that available from the memory as well as speeds of the CPUs and IOPs relative to that of the memory.

From the above scenario it appears that if no jobs are permanently resident in the main memory then the IOPs that are assigned to load operations should be able to load jobs as fast as the CPUs and IOPs that are assigned to processing operations can complete jobs, otherwise the CPUs and a portion of the main memory stand a higher chance of being idle while waiting for jobs to be loaded into the memory. On the other hand, if these IOPs can load jobs faster than the CPUs and IOPs that are assigned to processing operations can complete jobs then these IOPs and that portion of main memory into which they may have just completed loading will be idle until some jobs are completed and their memory space is relinquished, thereby allowing more jobs to be loaded. Furthermore, the IOPs that are assigned to processing operations should be able to perform the I/O processing for the jobs that are initiated as fast as the CPUs can perform the CPU processing on this same set of jobs, otherwise a portion of the total space bandwidth product of the memory and either CPU or IOP time is wasted while one is waiting on the other. Thus it seems reasonable that to approach maximum utilization, after a steady state condition has been reached, approximately one-half the memory should be utilized for loading in new jobs and that the other half should be utilized for processing of those jobs that have completed loading. Furthermore, half of that half of memory space in which jobs are being processed should on the average be utilized for CPU processing with the other half of that half being utilized for I/O processing. In this manner utilization of memory, CPUs and IOPs should approach the maximum value that is obtainable with a particular set of jobs.

III. THE MEMORY CONFLICT PROBLEM

Memory conflict arises in the multiprocessor system when two or more processors attempt to obtain an access from the same memory module simultaneously. Obviously if all the processors involved in a conflict desired to read from the same location or write the same information into the same location then there would be no logical reason why they could not, however, the expected frequencies of these occurrencies are so small that they do not warrant the cost involved in detection and provision of special hardware for implementation. Thus, we may assume that all but one of the processors involved in a memory access conflict will be delayed by one memory module cycle period for each such conflict in obtaining the accesses that they desire. If it is assumed that no look-ahead or look-behind is used in obtaining memory accesses then all but one of the programs whose processors are involved in a particular memory access conflict will be delayed by a memory module cycle period. In this regard let us define slowdown of a particular job and slowdown of the system. In order to do this we need to assume a period of time over which the amount of slowdown is to be measured. It is most convenient to measure time by counting the number of memory module cycle periods contained in the interval of time that it is desired to measure. Then the amount of slowdown for a particular job over some interval of time containing T memory

module cycle periods is the ratio of the number of memory module cycle periods at which the job made a memory access request but did not obtain one to the total number of memory module cycle periods at which the job made an access request during the interval T. Similarly the slow down of the system over this interval T is the ratio of the sum over all memory module cycle periods in the interval T of the sum of the total number of access requests made at each period but not granted to the sum over all memory module cycle periods in the interval T of the sum of the total number of accesses requested.

These definitions may be made clearer through the following equations that are useful for calculating the amount of slowdown.

$$\text{Let } r(i,j,k) = \begin{cases} 1 & \text{if job j requests a memory access from module k at time i.} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Let } rng(i,j,k) = \begin{cases} 1 & \text{if } r(i,j,k)=1 \text{ and module k does not grant an access to job j at time i.} \\ 0 & \text{otherwise} \end{cases}$$

Let NMOD be the number of memory modules in the system. Then slowdown for job j (SLDN(j)) over the interval [a,b] is given by

$$SLDN(j) = \frac{\sum_{i=a}^{b} \sum_{k=1}^{NMOD} rng(i,j,k)}{MAX\left(1, \sum_{i=a}^{b} \sum_{k=1}^{NMOD} r(i,j,k)\right)}$$

Let NJOB(i) be the number of jobs in the system at time i. Then slowdown of the system over the interval [a,b] is given by

$$SLDNTOTAL = \frac{\sum_{i=a}^{b} \sum_{k=1}^{NMOD} \sum_{j=1}^{NJOB(i)} rng(i,j,k)}{MAX\left(1, \sum_{i=a}^{b} \sum_{k=1}^{NMOD} \sum_{j=1}^{NJOB(i)} r(i,j,k)\right)}$$

It would be useful to obtain a feeling of how severe this problem really is under certain conditions.

Let us assume that the sum of the average access rates to a particular memory module of all processors executing jobs having space in this module to be less than or equal to the bandwidth of the module.

First let us consider the simplest case in which two processors are simultaneously executing programs from the same memory module and the mean access rate of each processor to the module is one-half the bandwidth of the module and uniformly distributed in time. Then at a given module access time t the probability of a conflict is 1/4 provided no conflict occurred during the previous access period t-1, and 1/2 provided a conflict occurred at time t-1. This is true since it may be assumed that if a processor does not obtain the access that it wants at time T then it will keep trying to obtain this same access at time T + 1 with probability 1. On the other hand, if it obtains an

access at time T then its probability of attempting to obtain an access at time T + 1 is 1/2. Thus we have the relations

$$P_t = 1/4(1 - P_{t-1}) + 1/2\ P_{t-1} = 1/4 + 1/4\ P_{t-1}$$

$$P_{t-1} = 1/4 + 1/4\ P_{t-2}$$

Actually there are only a finite number of these relations that we need to consider in order to find the probability of conflict at time t. However, it is easier to solve for the case of an infinite number of these relations. In the infinite case

$$P = \sum_{i=1}^{\infty} \left(\frac{1}{4}\right)^i = \frac{1/4}{1 - 1/4} = \frac{1}{3}$$

Thus, in this simple case we see that if each processor has equal probability of obtaining an access in case of conflict then each program is expected to proceed at 5/6 = .833 of normal speed. This may be justified on the basis of the assumption that the number of accesses over a given time interval is from a normal distribution for each job. The combined distribution is then also a normal distribution with mean equal to the sum of the means of the individual distributions and variance equal to the sum of the individual variances. Thus the combined standard deviation is less than or equal to the sum of the individual standard deviations.

Under the constraint that the sum of the mean bandwidth requirements of the programs being executed from a memory module does not exceed the module bandwidth then the case where each program has equal

bandwidth requirements is the worse case for conflict. This may be intuitively verified in the simple case of two processors simultaneously executing programs from the same memory module. Consider the case where the mean bandwidth requirements are 3/4 and 1/4 that of the memory module and uniformly distributed in time. Also assume that the probability of each processor obtaining the access in case of conflict is 1/2. Then the probability of conflict at time t is $1/4 \cdot 3/4$ if no conflict occurred at time t-1. It is $1 \cdot 1/4$ if conflict occurred at time t-1 and the processor with bandwidth requirement of 3/4 did not receive an access at time t-1. It is $1 \cdot 3/4$ if conflict occurred at time t-1 and the processor with bandwidth requirement of 1/4 did not receive an access at time t-1. Thus

$$P_t = 3/16\ (1 - P_{t-1}) + 1/4 \cdot P_{t-1} \cdot 1/2 + 3/4\ P_{t-1} \cdot 1/2$$

$$= 3/16 + 5/16 \cdot P_{t-1}$$

so $$P_t = 3/16 + 15/256 + 75/4092 + \ldots .$$

Thus we have another geometric sequence with initial value 3/16 and common ratio of 5/16. In the infinite case the probability of conflict is P = 3/11. The general result is easily proven by elementary methods of calculus.

In the general case let xB_m be the mean bandwidth requirement of the first job and let $y \cdot B_m$ be the mean bandwidth requirement of the second job and assume that the access rate for each job is uniformly distributed in time. Furthermore let $c = x + y \leq 1$ where c is a constant. Then

$$P_t = x \cdot y\,(1 - P_{t-1}) + x \cdot P_{t-1} \cdot 1/2 + yP_{t-1} \cdot 1/2$$

$$= xy - xyP_{t-1} + 1/2\,(x + y)P_{t-1}$$

$$= xy + 1/2\,(x + y - 2xy)P_{t-1}$$

$$P_{t-1} = xy + 1/2\,(x + y - 2xy)\,P_{t-2}$$

so

$$P_t = xy + 1/2\,(x + y - 2xy)(xy + 1/2\,(x + y - 2xy))P_{t-2}$$

$$= xy + 1/2\,(x + y - 2xy)\,(xy) + [1/2\,(x + y - 2xy)]^2\,P_{t-2}$$

Finally

$$P_t = xy \cdot \sum_{i=o}^{\infty} [1/2\,(x + y - 2xy)]^i$$

$$= \frac{xy}{1 - \frac{1}{2}\,(x + y - 2xy)}$$

$$= \frac{x(c - x)}{1 - \frac{1}{2}\left(c - 2x(c - x)\right)}$$

Differentiating with respect to x and setting result to 0 we obtain

$$\frac{dP_t}{dx} = \frac{(c - 2x)\left(1 - \frac{1}{2}\,(c - 2x(c - x)\right) - \left(xc - x^2\right)\left(c - 2x\right)}{\left(1 - \frac{1}{2}\,(c - 2x(c - x))\right)^2} = 0.$$

Thus

$$(c - 2x)\left(1 - \tfrac{1}{2}c + cx - x^2\right) - \left(xc - x^2\right)\left(c - 2x\right) = 0$$

$$x = \frac{1}{2}\,c.$$

Now consider 3 processors executing programs from a single memory module. Assume also that the access rate for each program is uniformly distributed over its processing period and that all processors have equal probability of obtaining an access in case of conflict. We must now worry not only with the probability of conflict but also with the degree of conflict. Let us denote these degrees at time t by $P_{1,t}$ and $P_{2,t}$ where $P_{1,t}$ is the probability of conflict involving 2 processors and $P_{2,t}$ is the probability of conflict at time t involving 3 processors.

The probability of conflict of degree 1 at time t is the probability that exactly two processors contend for an access at time t. Thus the probability of contention of degree 1 at time t is $\frac{2}{3}\left(\frac{1}{3} \cdot \frac{1}{3} \cdot \binom{3}{2}\right)$ provided no contention occurred at time t-1. The probability of contention of degree 1 at time t is $1 \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot \binom{2}{1}$ provided contention of degree 1 occurred at time t-1. It is equal to 2/3 provided contention of degree 2 occurred at time t-1. Thus

$$P_{1,t} = \frac{2}{9}\left(1 - P_{1,t-1} - P_{2,t-1}\right) + \frac{4}{9} \cdot P_{1,t-1} + \frac{2}{3} P_{2,t-1}$$

$$P_{1,t} = \frac{2}{9} + \frac{2}{9} P_{1,t-1} + \frac{4}{9} P_{2,t-1}$$

Similarly the probability of conflict of degree 2 is

$$P_{2,t} = \left(\frac{1}{3}\right)^3 \cdot \left(1 - P_{1,t-1} - P_{2,t-1}\right) + \left(\frac{1}{3}\right)^2 \cdot P_{1,t-1} + \frac{1}{3} P_{2,t-1}$$

Thus by backing up step by step in the time and solving for $P_{1,t}$ recursively we find that

$$P_{1,t} = \sum_{i=1}^{\infty}\left(\frac{2}{9}\right)^{i} + \frac{4}{9}\cdot\sum_{i=1}^{\infty}\left(\frac{2}{9}\right)^{i-1} P_{2,t-i}$$

and similarly,

$$P_{2,t} = \frac{1}{27}\cdot\sum_{i=1}^{\infty}\left(\frac{8}{27}\right)^{i-1} + \frac{2}{27}\cdot\sum_{i=1}^{\infty}\left(\frac{8}{27}\right)^{i-1} P_{1,t-i}$$

In order to make the problem tractable let us assume that the probabilities of conflict of each degree are all stationary time series [13], [14]. Thus

$$P_{j,t-i} = P_{j,t} \quad \text{for } i = 1, 2, \ldots$$

Then we may obtain

$$P_{1,t} = \frac{2}{7} + \frac{4}{7} P_{2,t}$$

$$P_{2,t} = \frac{1}{19} + \frac{2}{19} P_{1,t}$$

Therefore

$$P_{1,t} = \frac{42}{125} = .336$$

and

$$P_{2,t} = \frac{209}{2375} = .088$$

The slowdown due to memory conflict is

$$\frac{1}{3} \times .336 + \frac{2}{3} \times .088 = .112 + .059 = .171$$

Under the assumption of equal chance for obtaining an access in case of conflict each program is expected to proceed at .829 of normal speed.

For the four processor case under the assumptions as above the pertinent equations are:

$$P_{1,t} = \frac{27}{101} + \frac{45}{101} P_{2,t} - \frac{27}{101} P_{3,t}$$

$$P_{2,t} = \frac{3}{43} + \frac{6}{43} P_{1,t} + \frac{45}{43} P_{3,t}$$

and

$$P_{3,t} = \frac{1}{193} + \frac{3}{193} P_{1,t} + \frac{15}{193} P_{2,t} \quad .$$

The solution to this set of simultaneous equations is:

$$P_{1,t} = .326, \; P_{2,t} = .137 \quad \text{and} \quad P_{3,t} = .0209$$

Thus the slowdown is

$$\frac{3}{4} P_{3,t} + \frac{1}{2} \times P_{2,t} + \frac{1}{4} \times P_{1,t}$$

$$= .0157 + .0685 + .0815 = .166$$

so each program proceeds at .834 normal speed.

Thus we note at this point that the slowdown due to memory contention is not very sensitive to the number of processors executing programs from a single module so long as the sum of the means of the

bandwidth requirements are kept less than or equal to that of the memory module.

Now let us consider the general case of N processors executing N programs from a single memory module in which each processor has a mean access rate equal to 1/N of the bandwidth of the memory module. Also assume the following:

(1) The access rate for each program to the memory module is uniformly distributed over its processing interval.

(2) All processors have equal chance for obtaining the access in case of an access conflict.

(3) The probabilities of conflict of each degree form stationary time series.

(4) All contending processors not gaining an access when conflict occurs continue to contend for an access at subsequent access periods until they obtain an access.

Under these conditions the general formula for the probability of conflict of degree k at time t is given by

$$P_{k,t} = \sum_{i=0}^{MIN(N-1,\ k+1)} \binom{N-i}{k+1-i} \left(\frac{N-1}{N}\right)^{N-k-1} \left(\frac{1}{N}\right)^{K+1-i} P_{i,t-1} \qquad (3.1)$$

for $k = 1, 2, \ldots N-1$

and $P_{o,t-1} = 1 - \sum_{j=1}^{N-1} P_{j,t-1}$.

This formula leads to a set of N-1 simultaneous linear equations in N-1 unknowns when developed in the manner as shown for 2,3 and 4 processors.

The amount of slowdown due to memory contention under the assumptions of the preceeding discussion is plotted for 2 through 16 processors in Fig. 3-1. Also plotted are three other curves representing mean bandwidth requirements of $\frac{1}{4N}$, $\frac{1}{2N}$, and $\frac{3}{4N}$ times the memory module bandwidth for each processor. From this figure it should be observed that the amount of slowdown due to memory contention peaks at approximately 17% for the three processors case and decreases slowly to approximately 11% for 16 processors when the mean bandwidth requirement per processor is equal to $1/N \cdot B_m$ where B_m denotes the memory module bandwidth. Thus we may conclude, aside from the physical problem of contention resolution, that while the amount of slowdown due to memory contention (i.e., processors getting in one another's way) is not insignificant it is not intolerable. In this context it should be pointed out that a distinction has been made between slow down due to conflict when sufficient memory module bandwidth is available to meet the mean demands of all the jobs being simultaneously executed from that module and slowdown due to contention when insufficient bandwidth is available to meet the mean demands. In the first case the slowdown may be attributed completely to the relationships between the time of occurrence of the individual job demands and the manner in which the actual execution of each program proceeds (i.e., on

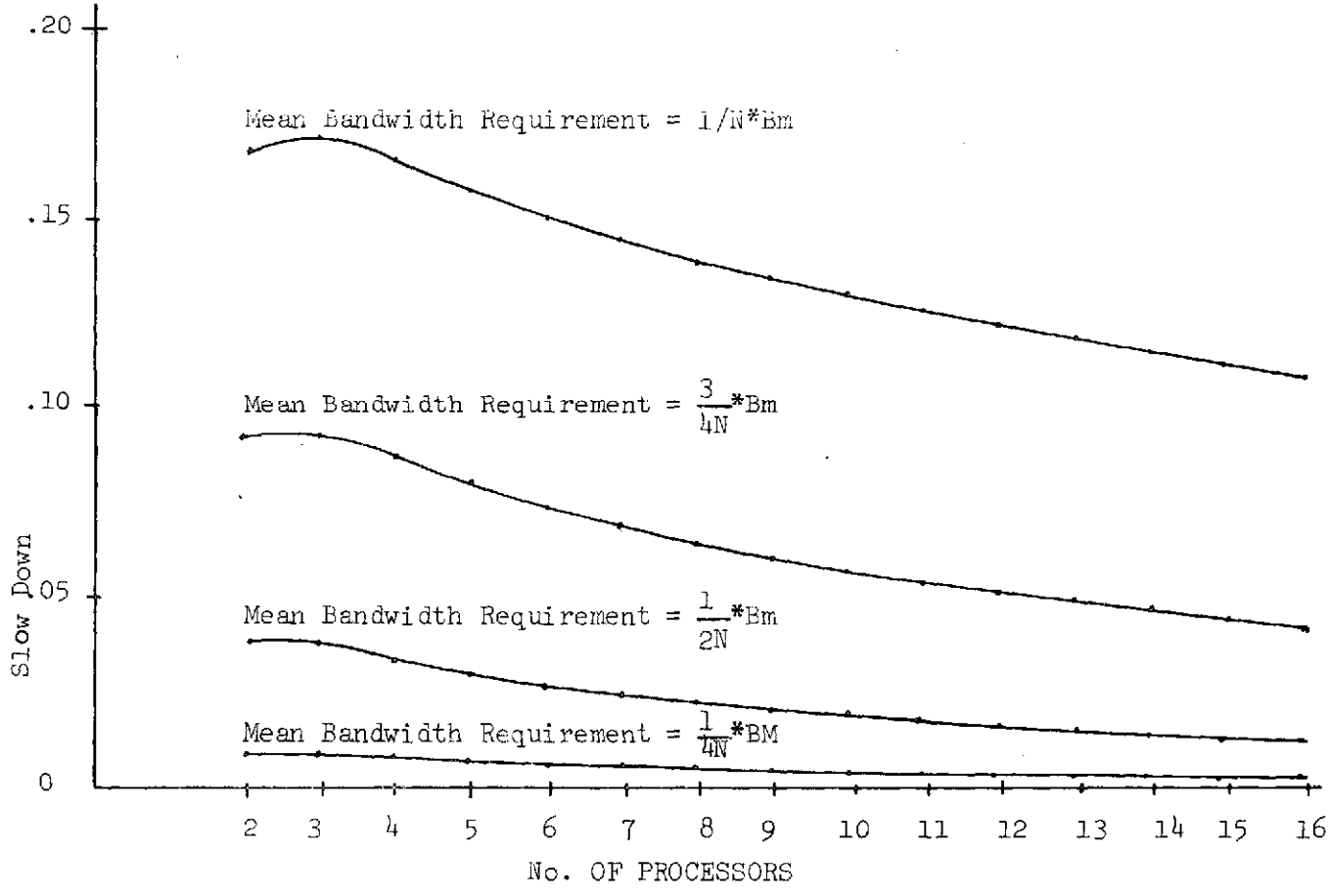


Fig. 3-1. Slowdown due to memory contention without bandwidth limiting

a fixed sequence of instruction executions in which no remaining instruction can proceed until its immediate predecessor has been executed). Attempts to lessen the effects of slowdown due to contention in this first case include the use of look-ahead and look-behind techniques. Look-ahead attempts to capitalize on the basically sequential nature of computer programs by reading the contents of the next n consecutive instruction locations from the main memory (usually an interleaved memory) at each memory access period into a dedicated local storage area for the processor obtaining the access at each period. This technique is described in considerable detail by Burnett [1] and by Burnett and Coffman [2]. The look-ahead technique is also embodied to a certain extent in high speed buffer storage memory systems as described by Conti [3] [4] and by Liptay [5] [6]. Look-behind attempts to exploit short length backward branches in computer programs such as short DO loops by retaining the preceeding n instructions in a local dedicated storage area for the processor.

Slowdown due to localized bandwidth limitation can occur with a uniprocessor as well as with a multiprocessor but the probability of occurrence is much greater with the multiprocessor. This probability is high when several processors are simultaneously active and when module size is much greater than mean job space size or when job space requirements are dispersed over a large range and several high bandwidth jobs are simultaneously executed from a single memory module. The complement problem to localized bandwidth limitation is inefficient

bandwidth utilization caused either by jobs with large space to bandwidth ratio requirements or by the use of scheduling and memory allocation routines that do not take bandwidth requirements into consideration.

The effects of these second causes of slowdown and bandwidth nonutilization will be explored in the next section.

The amount of nonutilization of memory bandwidth can also be easily obtained by means of the preceeding analysis through the following equation:

$$P_{nu} = \left(\frac{N-A}{N}\right)^N \cdot P_{o,t-1}$$

where

P_{nu} = probability of nonutilization,

N = number of processors accessing the common memory module,

$\frac{A}{N}$ = mean bandwidth requirement of each job,

and

$P_{o,t-1}$ = probability of no conflict at time t-1.

Fig. 3-2 shows a plot of P_{nu} for four values of A equal to .25, .50, .75, and 1.0.

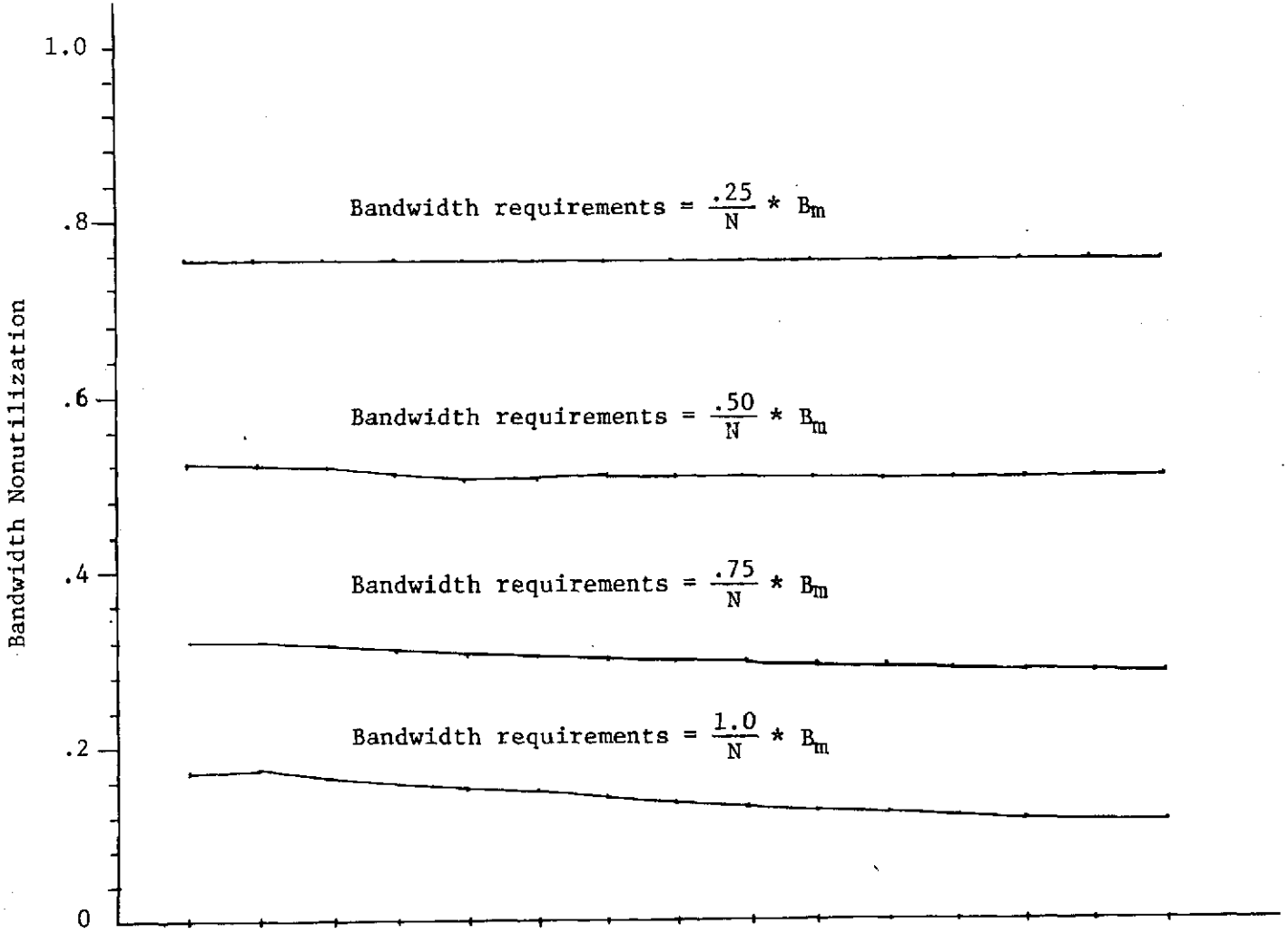


Fig. 3-2. Bandwidth Nonutilization.

IV. LOCALIZED BANDWIDTH LIMITATION AND MISMATCH

In equation 3.1 we may replace 1/N by z where z represents a time average probability that a job will require an access from its memory space at any given access period without regard to remembrance of previous response to access requests. Doing this we obtain

$$P_{k,t} = \sum_{i=0}^{MIN(N-1,k+1)} \binom{N-i}{k+1-i} (1-z)^{N-k-1} (z)^{k+1-i} P_{i,t-1} \qquad (4.1)$$

for $k = 1, 2, \ldots N-1$, $0 \leq z \leq 1$

and $P_{o,t-1} = 1 - \sum_{j=1}^{N-1} P_{j,t-1}$.

Thus, z is an average probabilistic measure of the bandwidth requirement from some memory module of a particular job having space in that module. Now as the bandwidth requirement of the job increases beyond the bandwidth available from the memory module the probability of requesting an access at any given access period approaches 1 without regard to any recall that the job is assumed to have about a prior access that may have not yet been granted. Thus when the combined mean bandwidth requirements of two or more jobs that are being simultaneously executed from a common memory module is increased beyond the bandwidth available from the common memory module the effects of memory access conflict due to the processors merely occasionally getting into

each other's way is quickly swamped by the effects of bandwidth limitation. The essence of the above statement is contained in Fig. 4-1 in which the slowdown obtained from solutions to the simultaneous linear equations ensuing from equation 4-1 have been plotted for 3 through 16 processors with values of z ranging from $.25/N \cdot B_m$ through $3/N \cdot B_m$ and for two processors from $.25/N\ B_m$ through $2/N \cdot B_m$ all with steps of $.25/N \cdot B_m$. Now consider the following tabulation obtained with the aid of the six processor case of Fig. 4-1.

Bandwidth Requirement $\div B_m$	Slowdown Due To Bandwidth Limitation	Total Slowdown From Curves	Slowdown Due To Access Contention
1.00	.0000	.150	.150
1.25	.2000	.245	.045
1.50	.3333	.350	.0167
1.75	.4285	.430	.0015
2.00	.500	.500	.0000
2.25	.5555	.5555	.0000

This brief analysis indicates that the contribution of access conflict to slowdown becomes insignificant when compared to that of bandwidth limitation for combined mean bandwidth requirements above 1.25 times the bandwidth available from the memory module.

It is apparent that if the bandwidth demands made to all the memory modules of the multiprocessor system is very much greater than

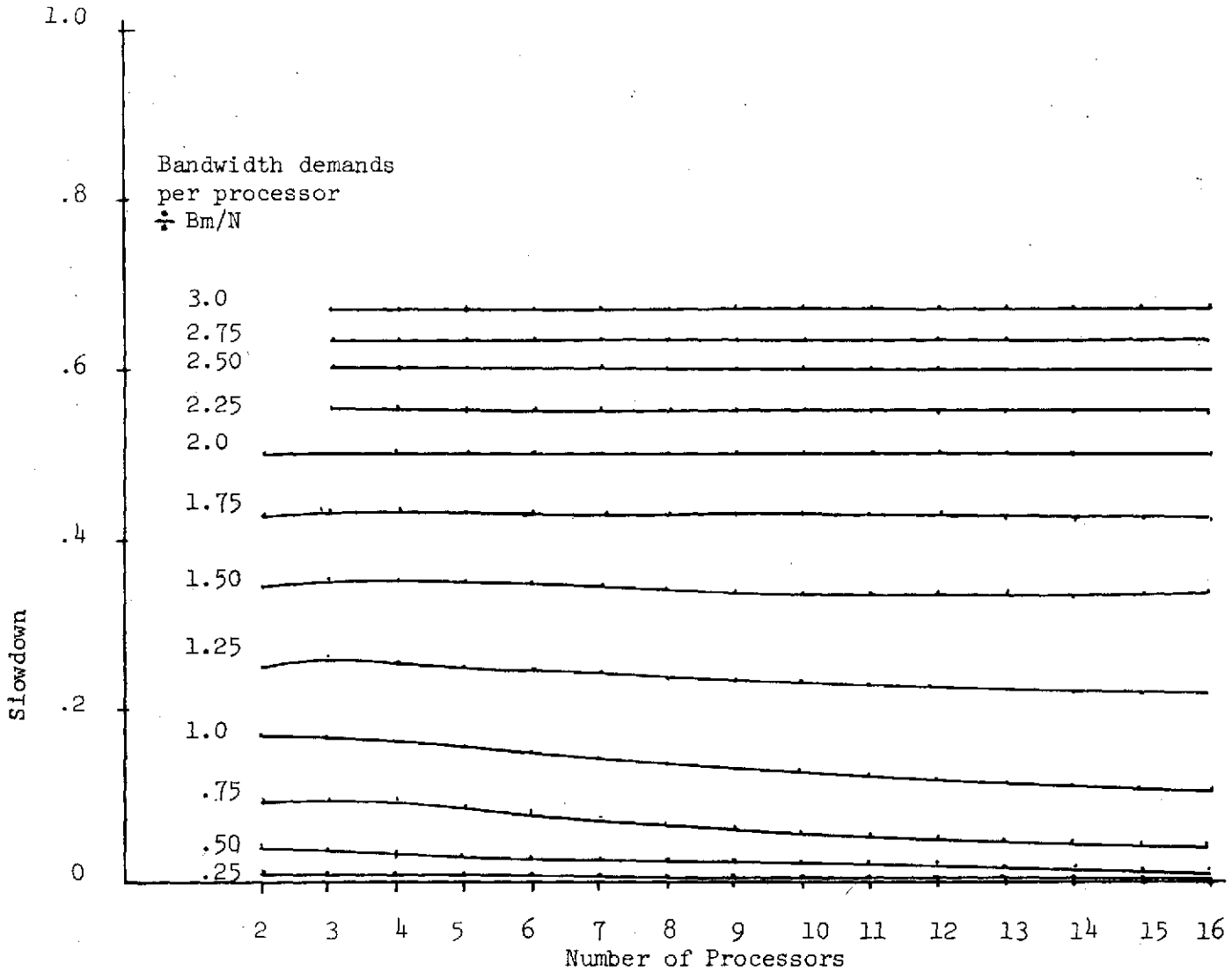


Fig. 4-1. Slowdown due to memory contention and bandwidth limitation.

that available from the modules then very little can be done through memory allocation techniques to alleviate the problem. On the other hand if the problem is one of distribution wherein the overall bandwidth demands are unevenly distributed among the memory modules then scheduling and allocation techniques which take into account bandwidth considerations may be effective in relieving the problem. First let us try to get a feeling for how severe the problem actually is.

Suppose that as input to the multiprocessor system we have an unlimited supply of incoming jobs (in our case a small set of recurring jobs) of two mean bandwidth requirements and one space requirement. The two mean bandwidth requirements are b_1 and b_2 where $b_1 + b_2 = B_m$ and $b_1 >> b_2$. The one space requirement is S_1 which is equal to one-half the space of a memory module (S_m). Furthermore let us assume that each type of job requires approximately the same amount of processing time and there are enough processors so that all jobs having memory space may be in progress simultaneously. Now depending upon how the jobs occur in time and how the job scheduler and memory allocator act in scheduling the jobs and placing them into the memory there may or may not be localized bandwidth limitation or mismatch. In this regard let us assume that there are enough of either type of job in the job queue at any schedule time to fill the available memory. Thus, a memory module may hold one of the three combinations b_1b_1, b_1b_2, or b_2b_2. For the combination b_1b_1 the jobs are slowed to approximately one-half of their normal speed due to insufficient bandwidth. However, the

bandwidth utilization is expected to be close to 1. In the second case (b_1b_2) the jobs are expected to proceed at approximately normal speed except for memory access conflict. The bandwidth utilization in this case is expected to be close to 1. In the case with b_2b_2 the jobs are expected to proceed at normal speed except for memory access conflict but the bandwidth utilization is expected to be closer to 0 than to 1. As a specific example let $b_1 = .9B_m$ and $b_2 = .1B_m$. Then with the combination b_1b_1 the slowdown due to insufficient bandwidth is expected to be .44. For the combination b_1b_2 no slowdown due to bandwidth limitation occurs and bandwidth utilization is close to 1. The b_2b_2 combination should not be slowed down due to bandwidth limitation but bandwidth utilization is expected to be only .2. With a random placement the probabilities of the combinations b_1b_1, b_1b_2, b_2b_2 are 1/4, 1/2, and 1/4 respectively. Thus slowdown of the overall system due to localized bandwidth limitation under this random placement is expected to be 1/4 x .44 + 1/2 x 0 = .11 and expected bandwidth utilization is 1/4 x 1 + 1/2 x 1 + 1/4 x .2 = .25 + .50 + .05 = .8.

The above introductory example suggests a method for investigating the amount of slowdown and nonutilization due to localized bandwidth limitation and mismatch under a random placement scheme. To aid in this investigation let us assume the following:

(1) The input job stream consists of an infinite supply of jobs of one space requirement $S_1 = S_m/n$ and n bandwidth requirements.

(2) Jobs with each bandwidth requirement occur with equal frequency.

(3) The scheduler and memory allocator shows no bias in regard to bandwidth requirements.

(4) Processing time is the same for all jobs regardless of bandwidth requirement.

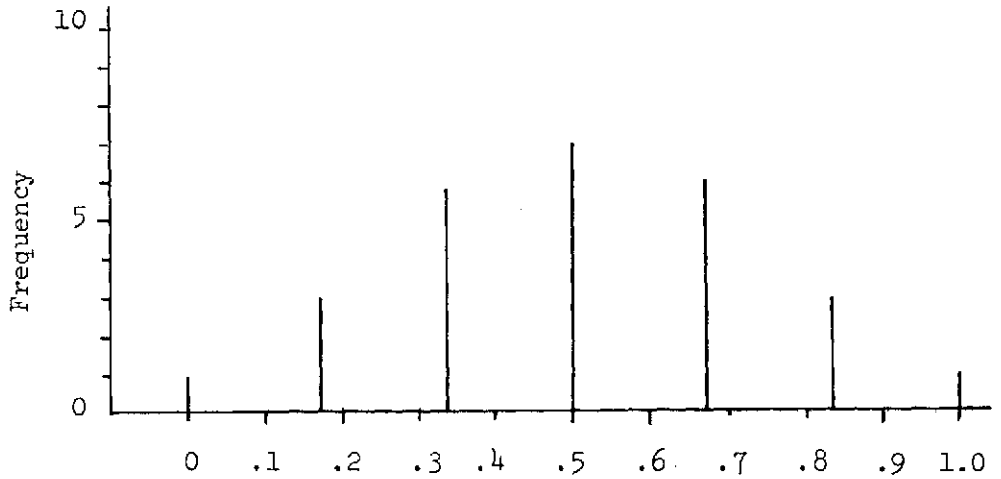
(5) The sum of the bandwidth requirements is A B_m and the actual requirement array BW is BW = (0, 1/n W, 2/n · W, . . . $\frac{n-1}{n}$. W), where $W = \frac{2A}{n(n-1)}$.

Under (1), (2), (4) and (5) there is at least one job scheduling and allocation scheme in which (3) holds and no slowdown due to bandwidth limitation or non-utilization of bandwidth due to mismatch results. This scheme is of course the one in which one job of each bandwidth requirement is loaded into each memory module. A random placement scheme would result in total bandwidth requirements that are in accordance with the multinomial distribution. For example, for the three processor case there are three different patterns of bandwidth selections. These three patterns are (x)(y)(z), (x)(y,y), and (x,x,x). The x's, y's and z's represent a particular bandwidth requirement. Clearly, in so far as the total bandwidth requirement is concerned permutations of the groups in a pattern or permutations of the elements in a group of a pattern or elements among groups of the same size in a member of a pattern makes no difference. For A = 1, the first pattern has the

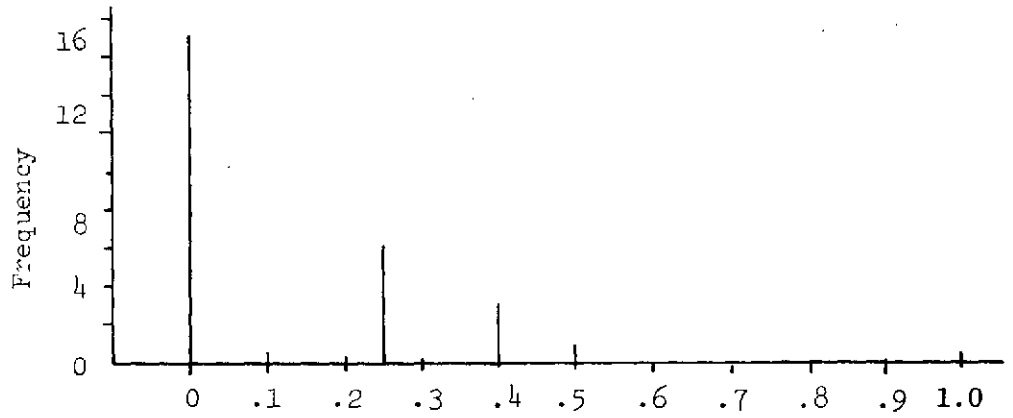
single member (0)(1/3)(2/3). It occurs with frequency $3!/(1! \times 1! \times 1!) = 6$ when all its permutations are totaled. For the second pattern the members are (0)(1/3, 1/3), (0)(2/3, 2/3), (1/3)(0,0), (1/3)(2/3, 2/3), (2/3)(0,0), and (2/3)(1/3, 1/3). Each of these members occurs with frequency $3!/(1! \times 2!) = 3$. Finally, the third pattern has members (0, 0, 0), (1/3, 1/3, 1/3) and (2/3, 2/3, 2/3). Each of these members occurs with frequency $3!/3! = 1$.

The distribution of total bandwidth requirement, slowdown due to bandwidth limitation and bandwidth utilization obtained by means of the above multinomial technique is shown in Fig. 4-2 for the three processor case.

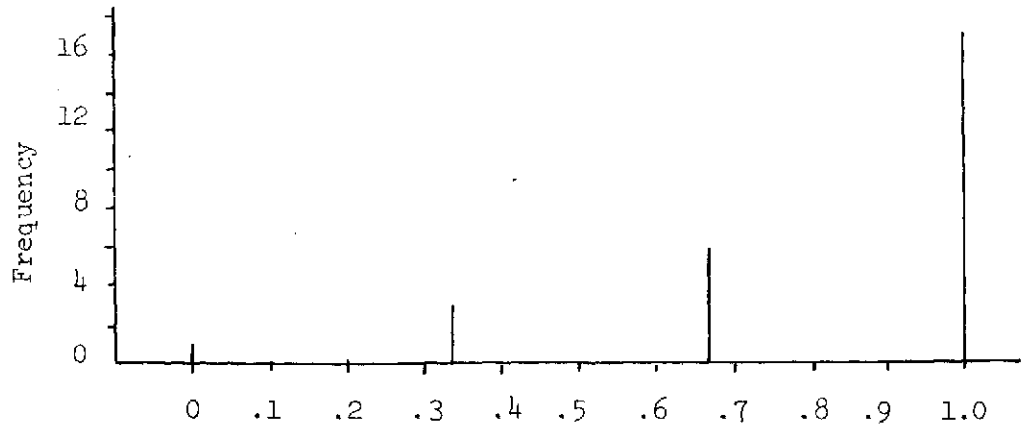
Unfortunately, the analysis by means of the multinomial technique is impractical for more than about eight or nine processors due to the very large number of terms that must be examined with more than this number of processors. For example, the number of terms for the three processor case is 10, for four processors it is 35 and for five processors it is 123. Thus in the low range of the numbers of processors the number of terms to be examined appears to be growing at the rate of approximately $(3.5)^i$. If this rate of growth is constant the 16 processor case will require examination of $10 \times (3.5)^{13} \simeq 1.183 \times 10^8$ terms. When all the time for other numbers of processors up to 16 are added to this and runs are made for several different values of A the amount of computer time could be several hundred hours. Therefore, a different technique is required for the solution of the problem. Let us use a random number trials procedure in which at each



A. Bandwidth Requirement
Normalization Factor = NPROS * BW(NPROS)



B. Slowdown



C. Bandwidth Utilization

Fig. 4-2 Distributions of bandwidth requirement, slowdown and bandwidth utilization for the three processor case from multinomial distribution
BW(I) = (I-1)A/(.5NPROS*(NPROS-1)) , A=1.0.

trial n indices are generated which represents the indexes of the bandwidth requirements array BW. Graphs may then be plotted for the frequencies of each total bandwidth requirement range, bandwidth utilization range, and slowdown range. The results of this procedure for 3, 10, and 16 processors with values of A equal to 0.75, 1.0, and 1.25 are shown in Fig. 4-3 through Fig. 4-14. The means and standard deviations for slowdown and bandwidth utilization with A equal to 0.75, 1.0, and 1.25 are plotted for 3 through 16 processors in Fig. 4-15 through Fig. 4-17. When interpreting these curves it should be kept in mind that the distributions may be highly unsymmetrical about their mean values as is evident from Fig. 4-6 through 4-14.

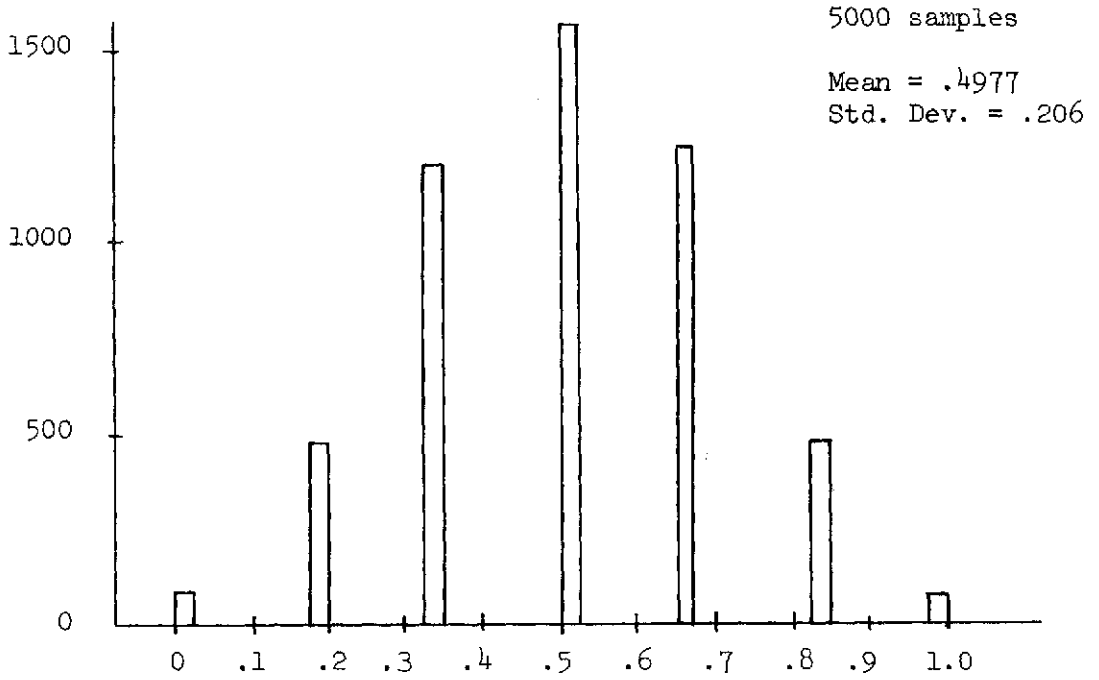


Fig. 4-3 Frequency-Gram for bandwidth requirement for NPROS = 3, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).

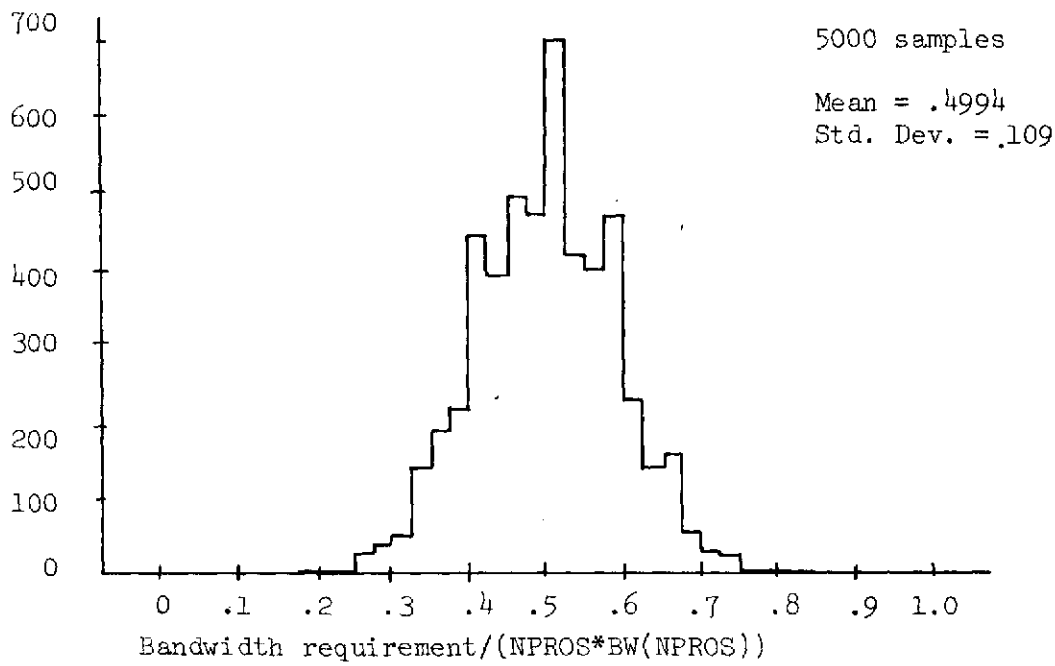


Fig. 4-4 Frequency-Gram for bandwidth requirement for NPROS = 10, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).

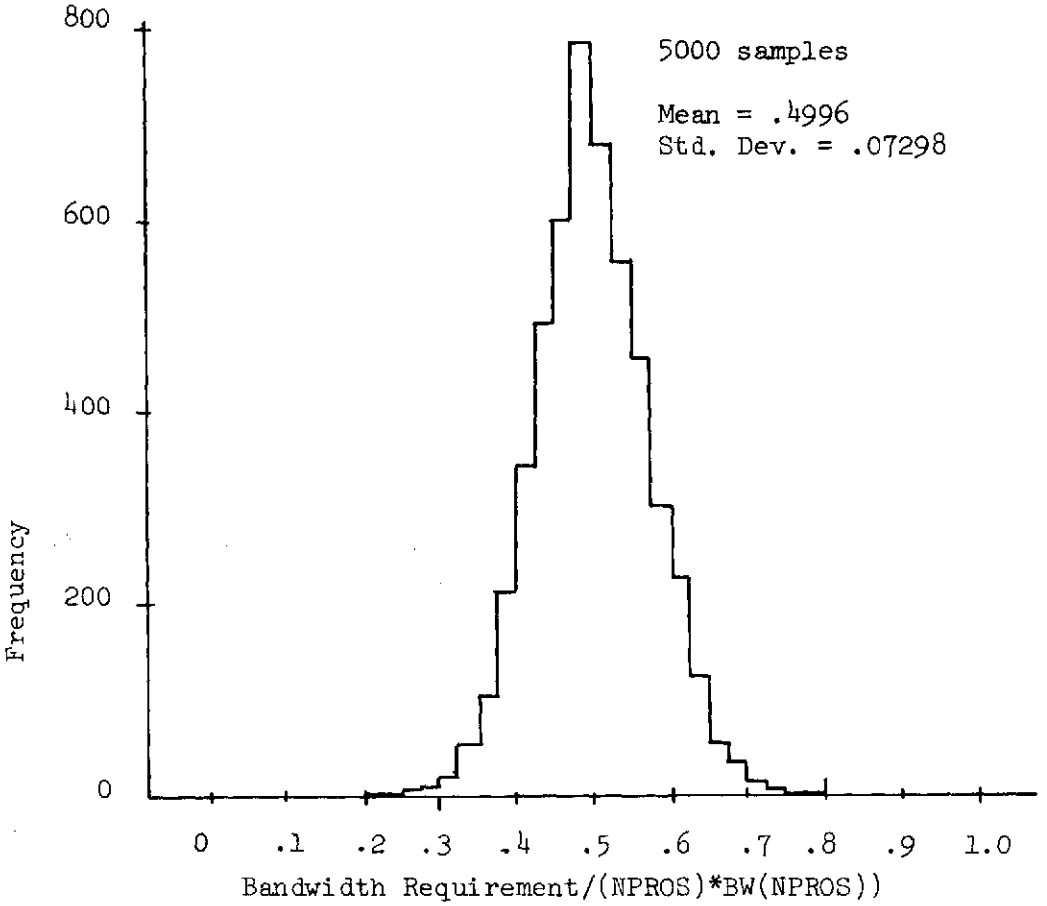
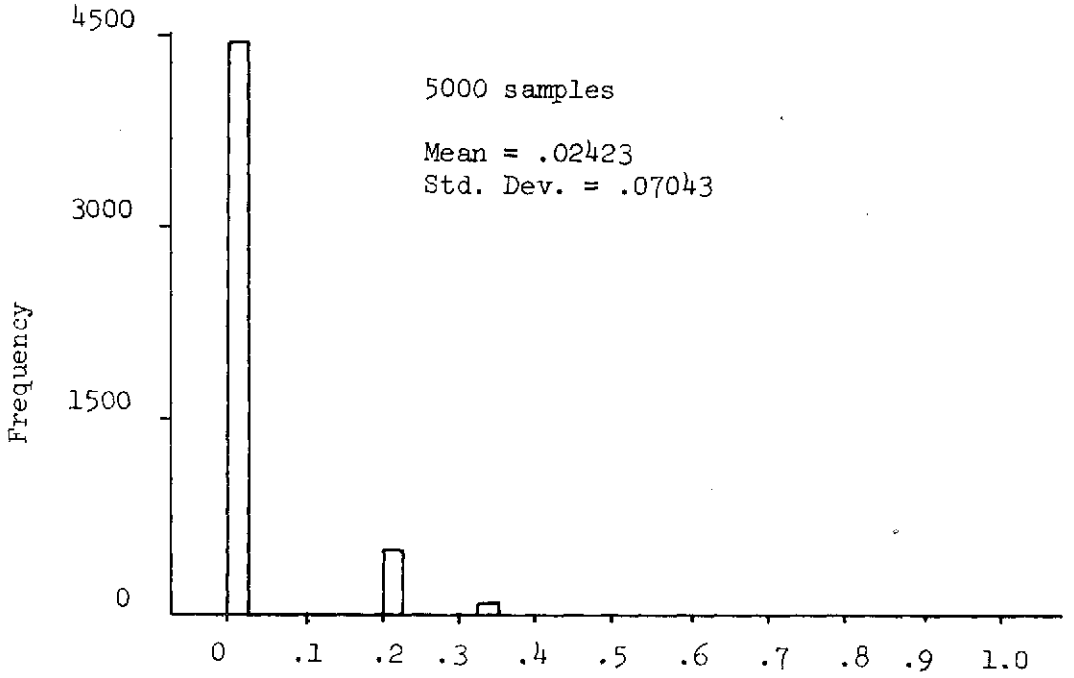
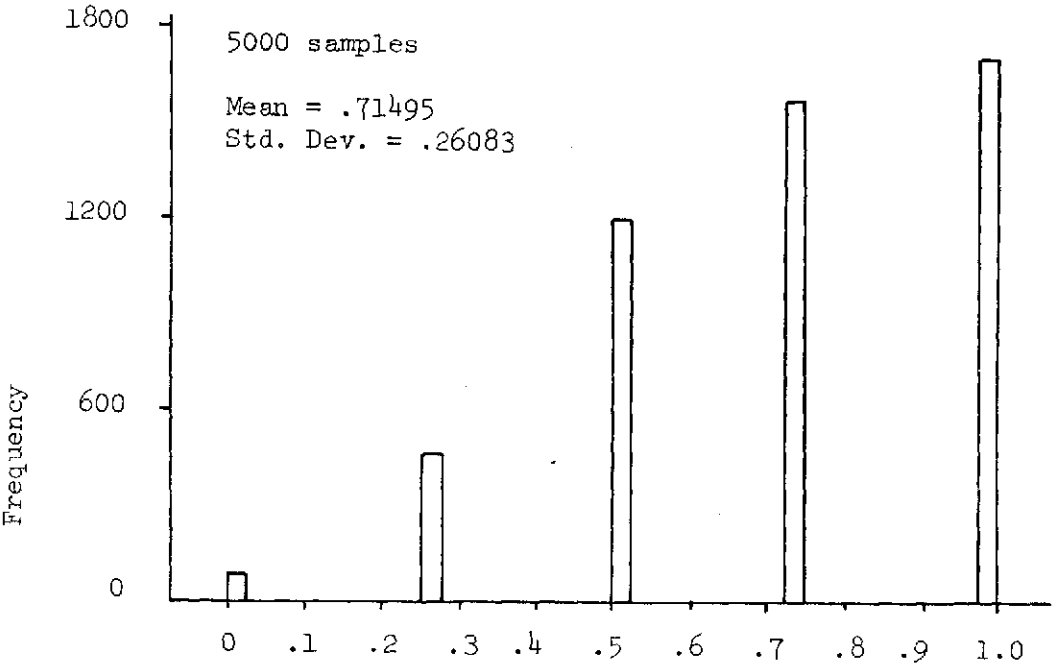


Fig. 4-5 Frequency-Gram for Bandwidth Requirement for NPROS = 16, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).

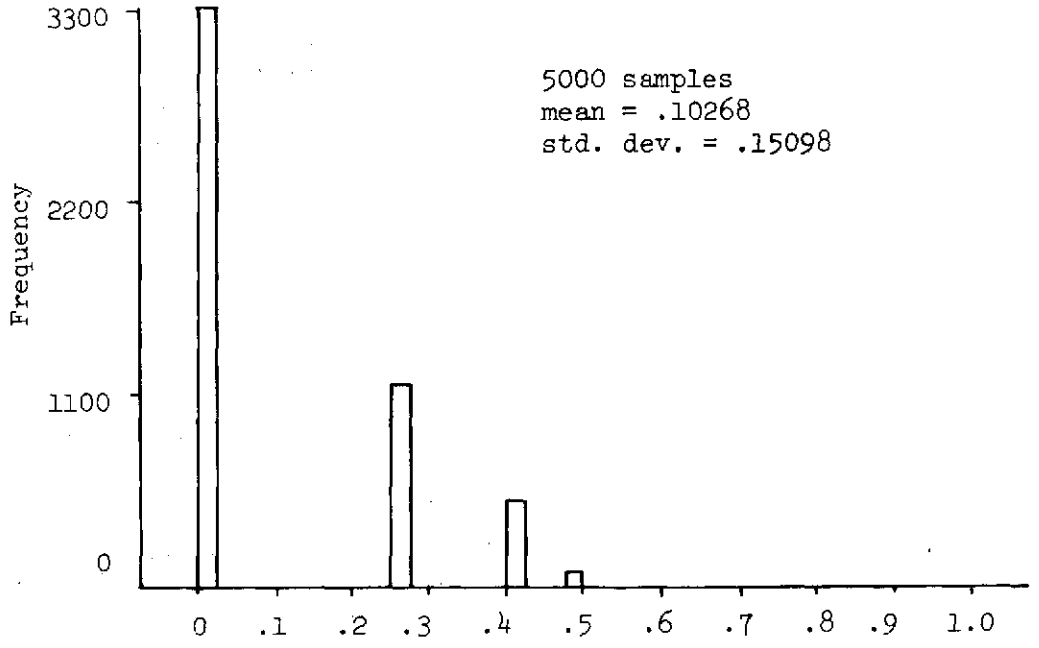


A. Slowdown

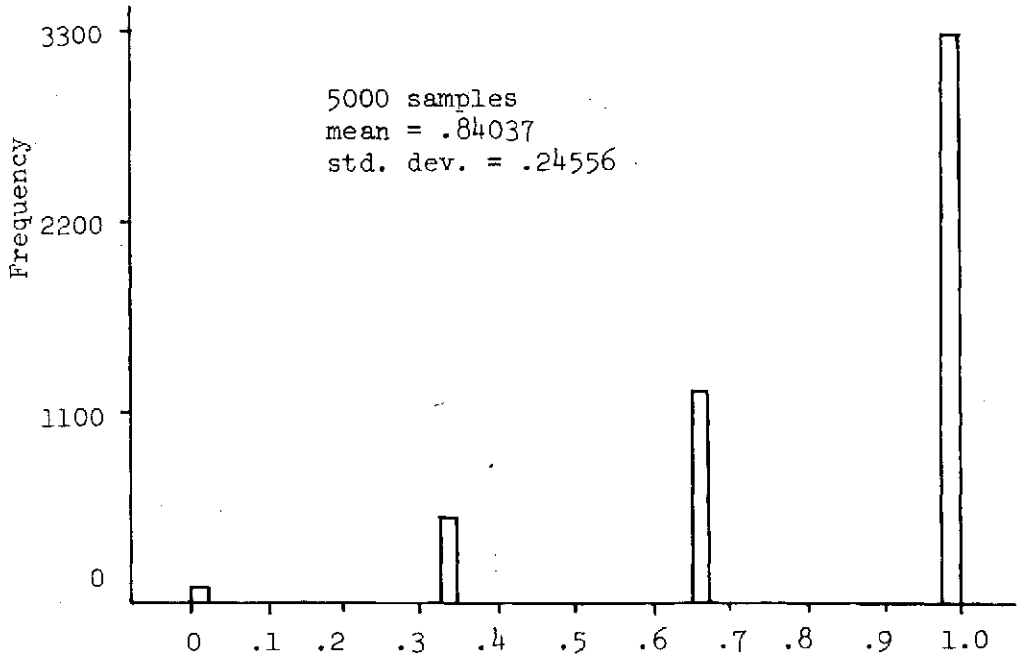


B. Bandwidth utilization

Fig. 4-6 Frequency Grams for slowdown and bandwidth utilization NPROS = 3, A = 0.75 BW(I)=(I-1)A/(.5NPROS*(NPROS-1)).

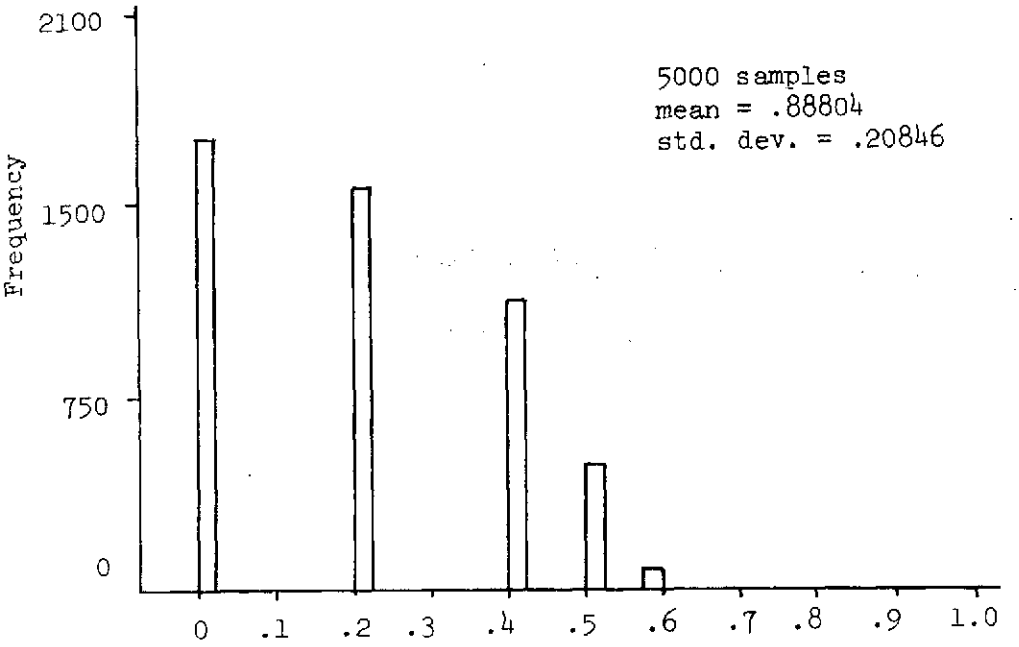


A. Slowdown

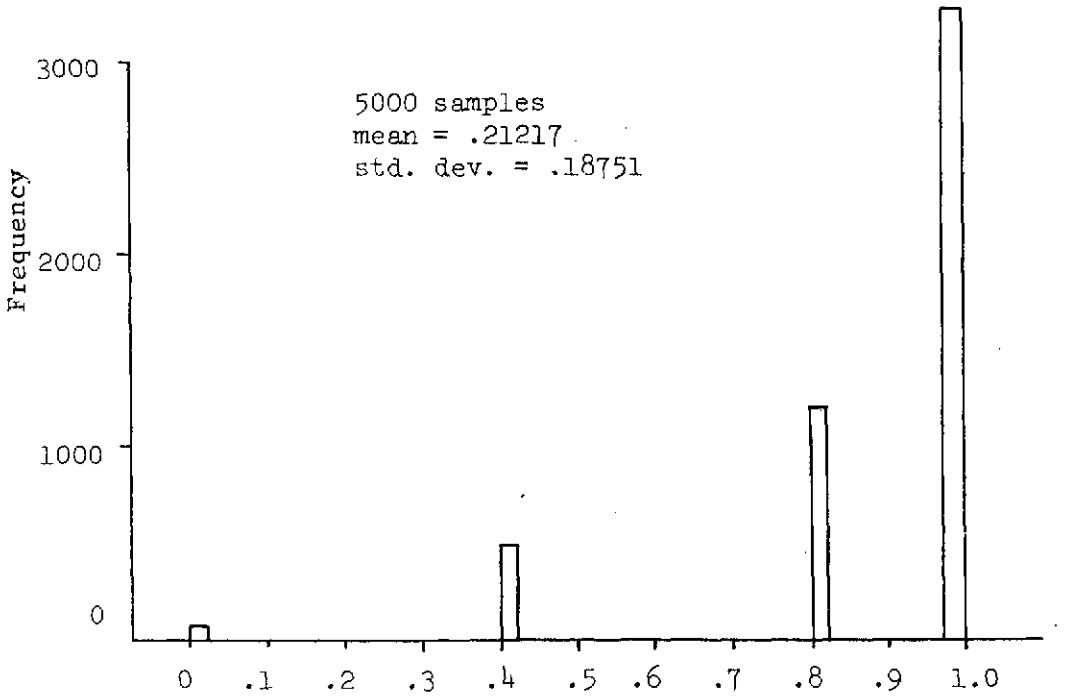


B. Bandwidth Utilization

Fig. 4-7. Frequency-grams for slowdown and bandwidth Utilization, NPROS + 3, A = 1.0, BW(I) = (I-1)A/(.5NPROS*(NPROS - 1)).

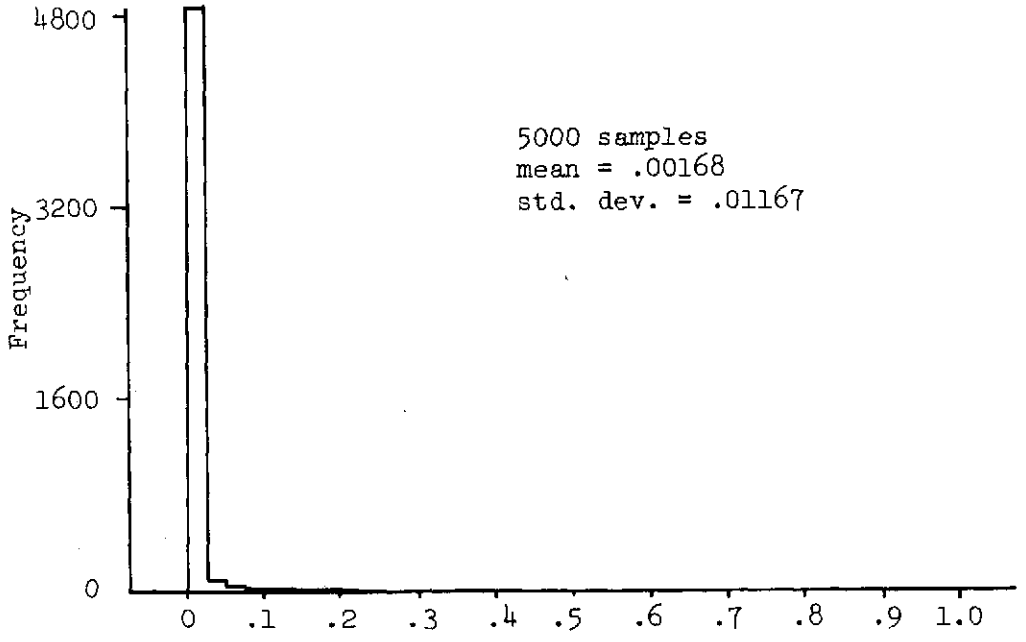


A. Slowdown

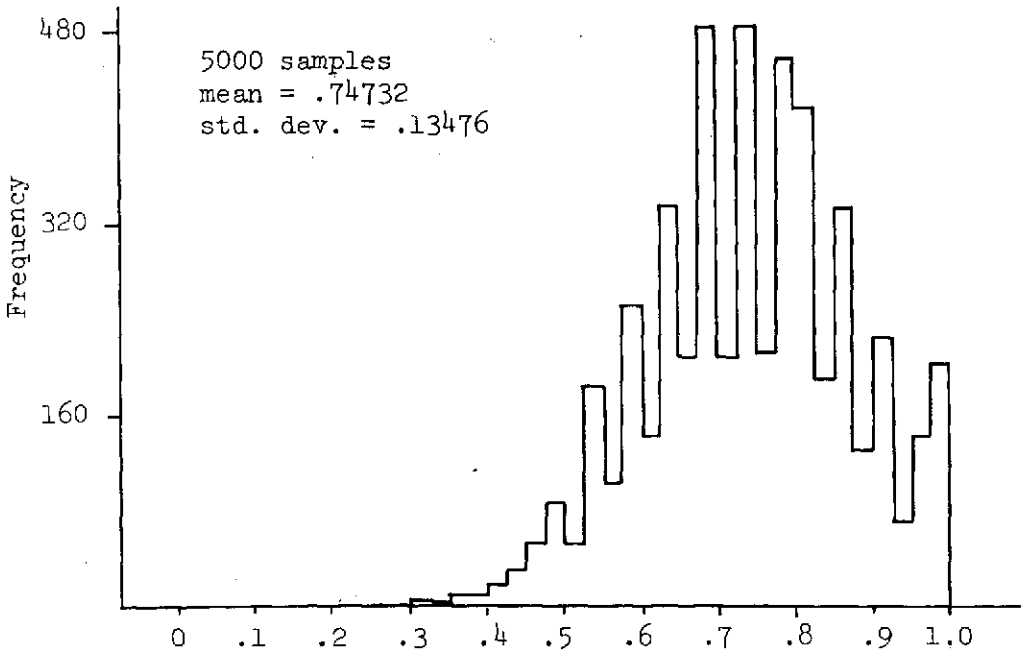


B. Bandwidth Utilization

Fig. 4-8. Frequency-grams for slowdown and bandwidth Utilization, NPROS = 3, A = 1.25, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).

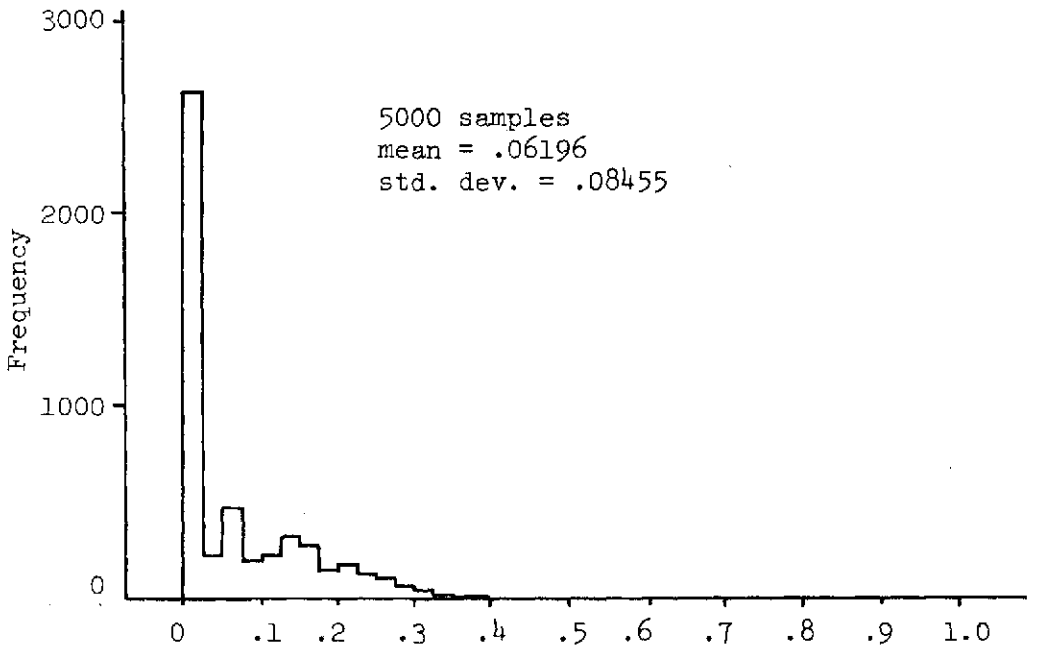


A. Slowdown

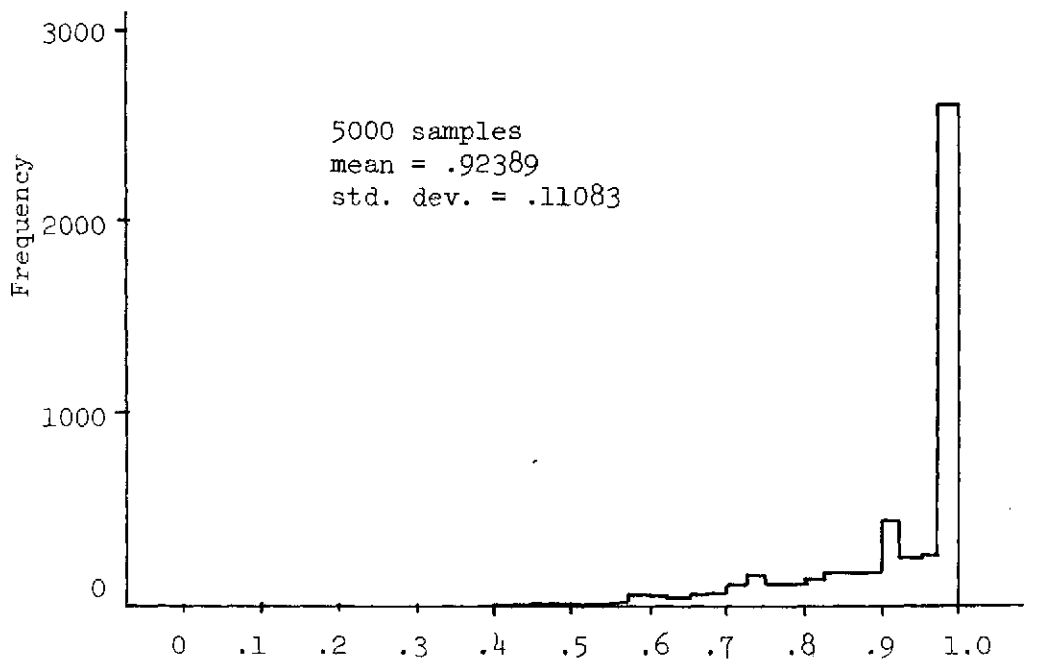


B. Bandwidth Utilization

Fig. 4-9. Frequency-grams for slowdown and bandwidth Utilization, NPROS=10, A = 0.75, BW(I) = (I-1)A/(.5 NPROS*(NPROS - 1)).

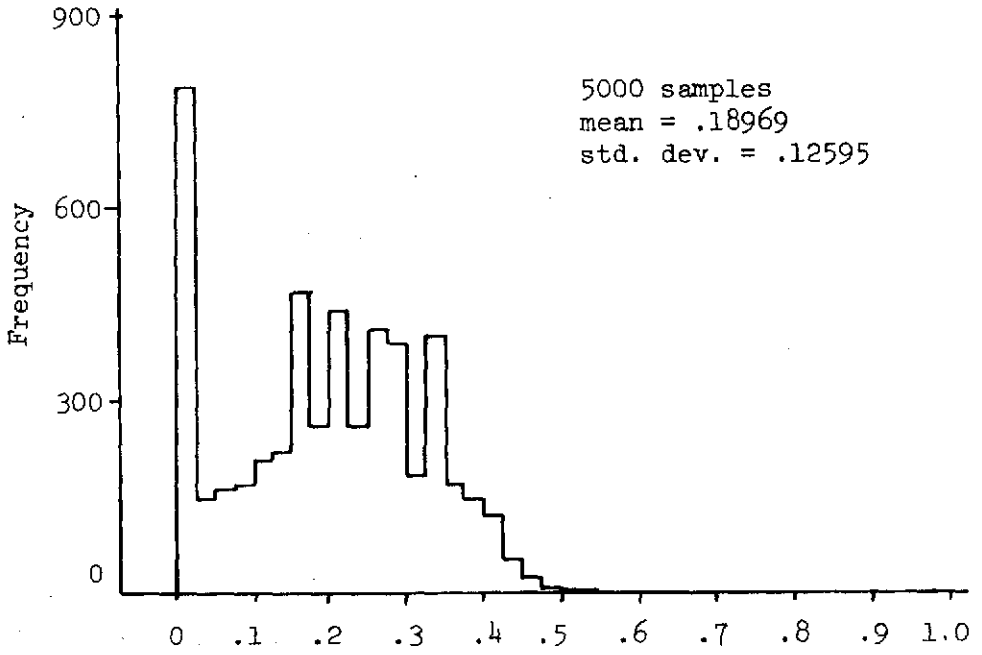


A. Slowdown

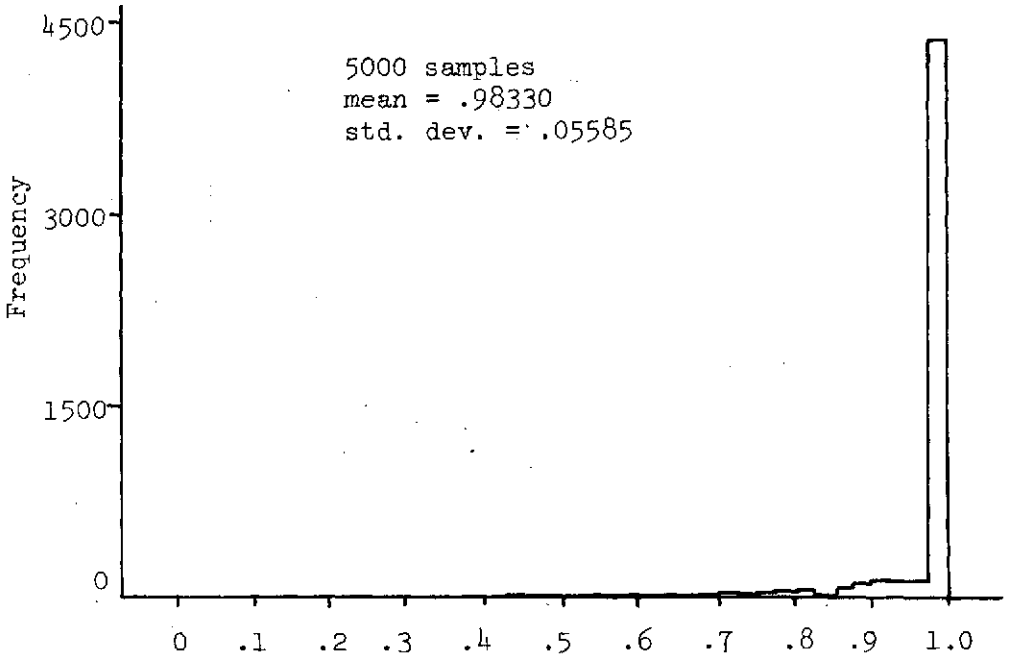


B. Bandwidth Utilization

Fig. 4-10. Frequency-grams for slowdown and bandwidth Utilization, NPROS=10, A = 1.0, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).

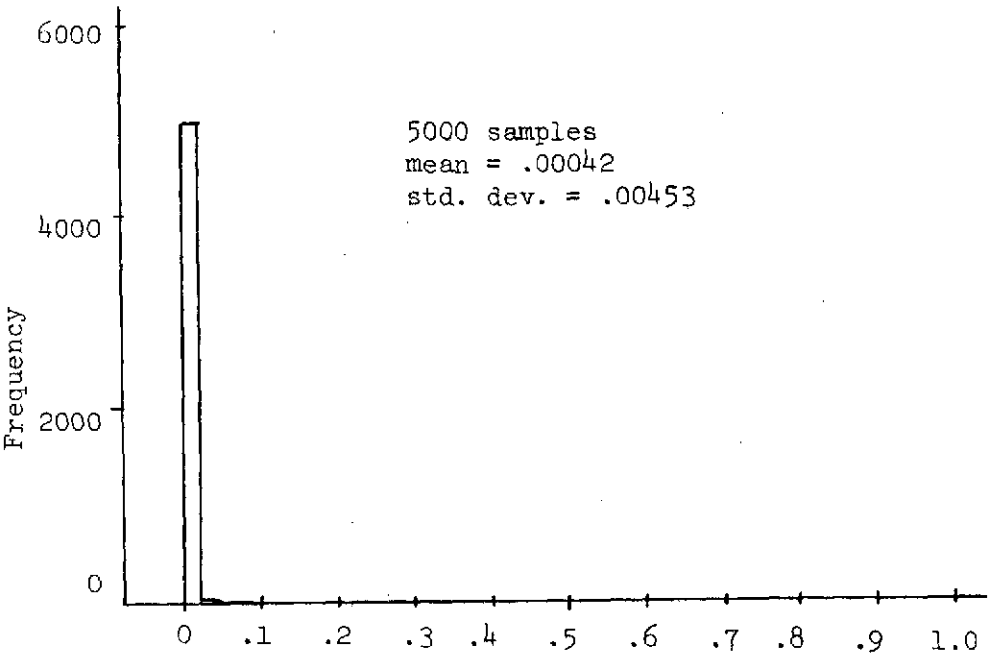


A. Slowdown

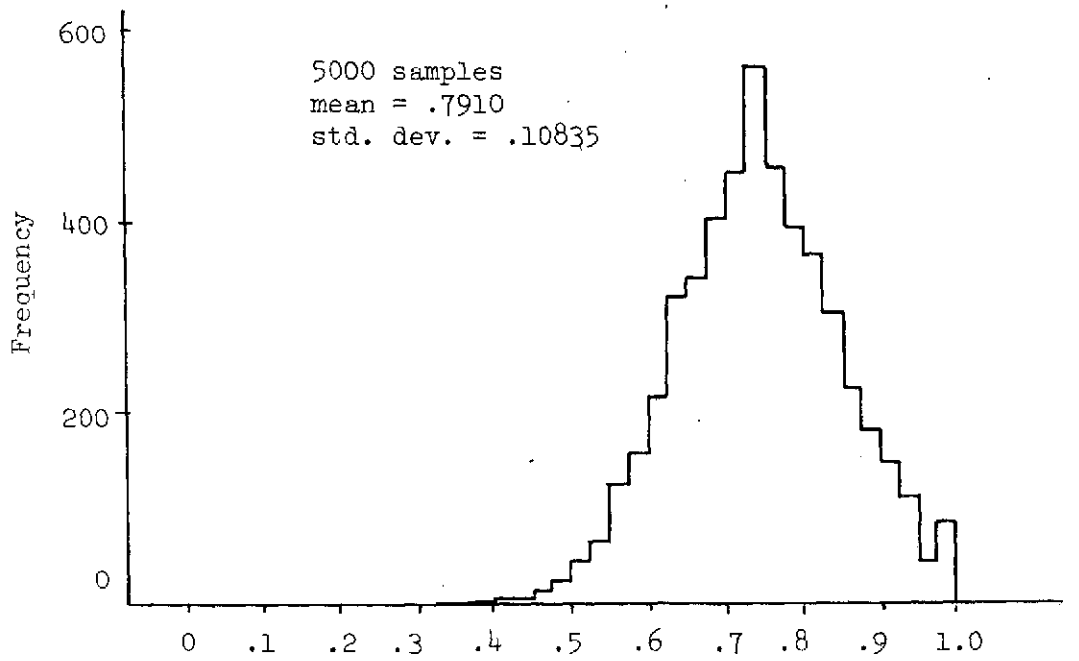


B. Bandwidth Utilization

Fig. 4-11. Frequency-grams for slowdown and bandwidth Utilization, NPROS=10, A = 1.25, BW(I) = (I-1)A/(.5NPROS*(NPROS - 1)).

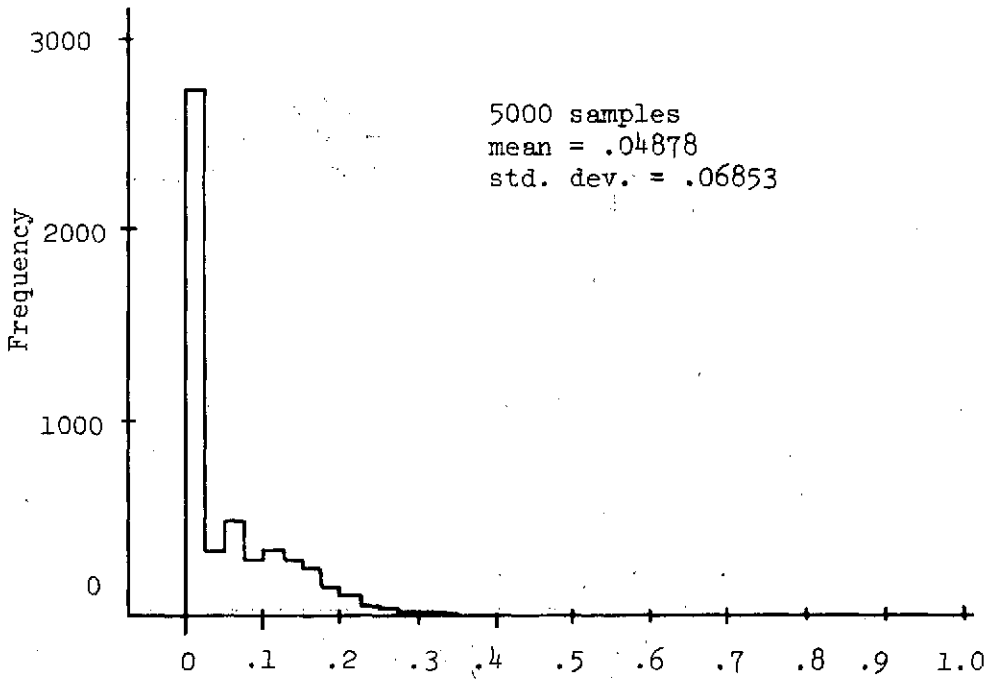


A. Slowdown

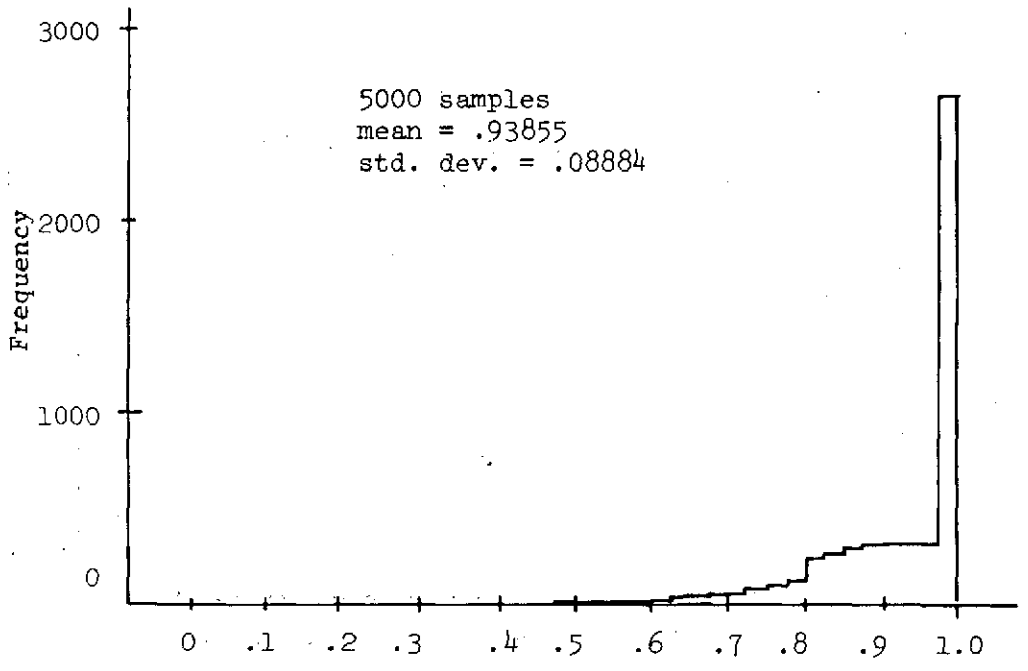


B. Bandwidth Utilization

Fig. 4-12. Frequency-grams for slowdown and bandwidth Utilization, NPROS=16, A = 0.75, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).

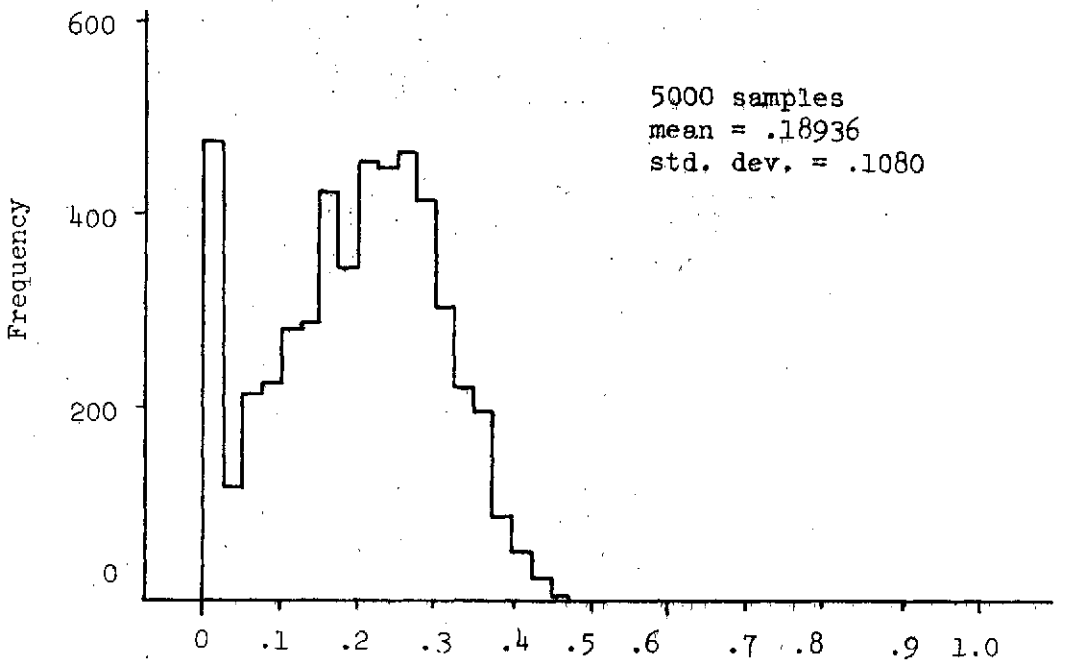


A. Slowdown

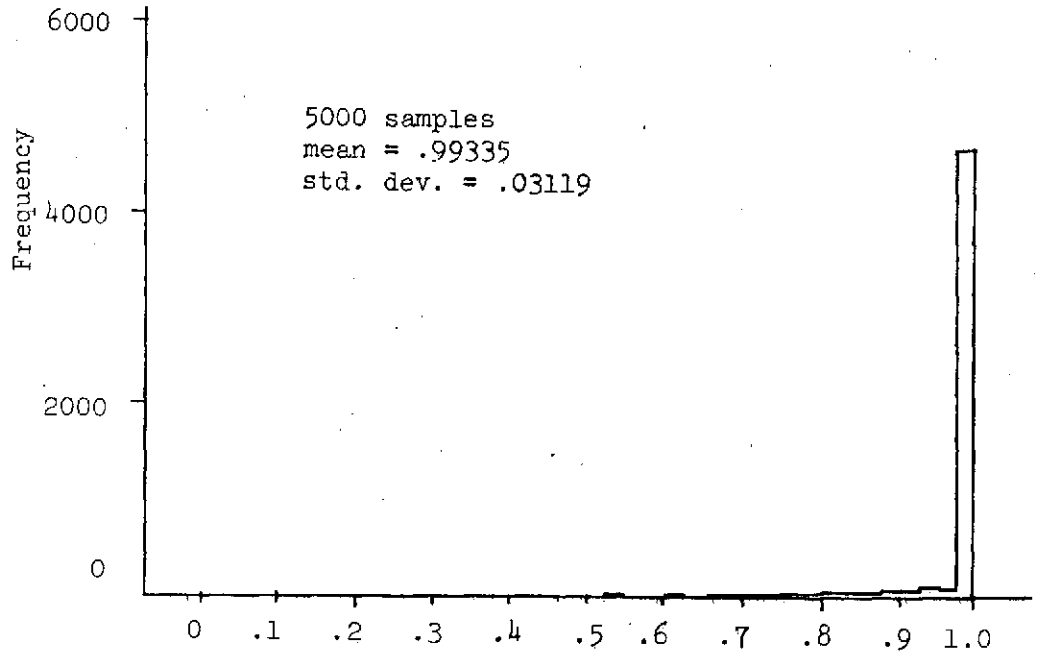


B. Bandwidth Utilization

Fig. 4-13. Frequency-grams for slowdown and bandwidth Utilization, NPROS = 16, A = 1.0, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).



A. Slowdown



B. Bandwidth Utilization

Fig. 4-14. Frequency-grams for slowdown and bandwidth Utilization, NPROS = 16, A = 1.25, BW(I) = (I-1)A/(.5NPROS*(NPROS-1)).

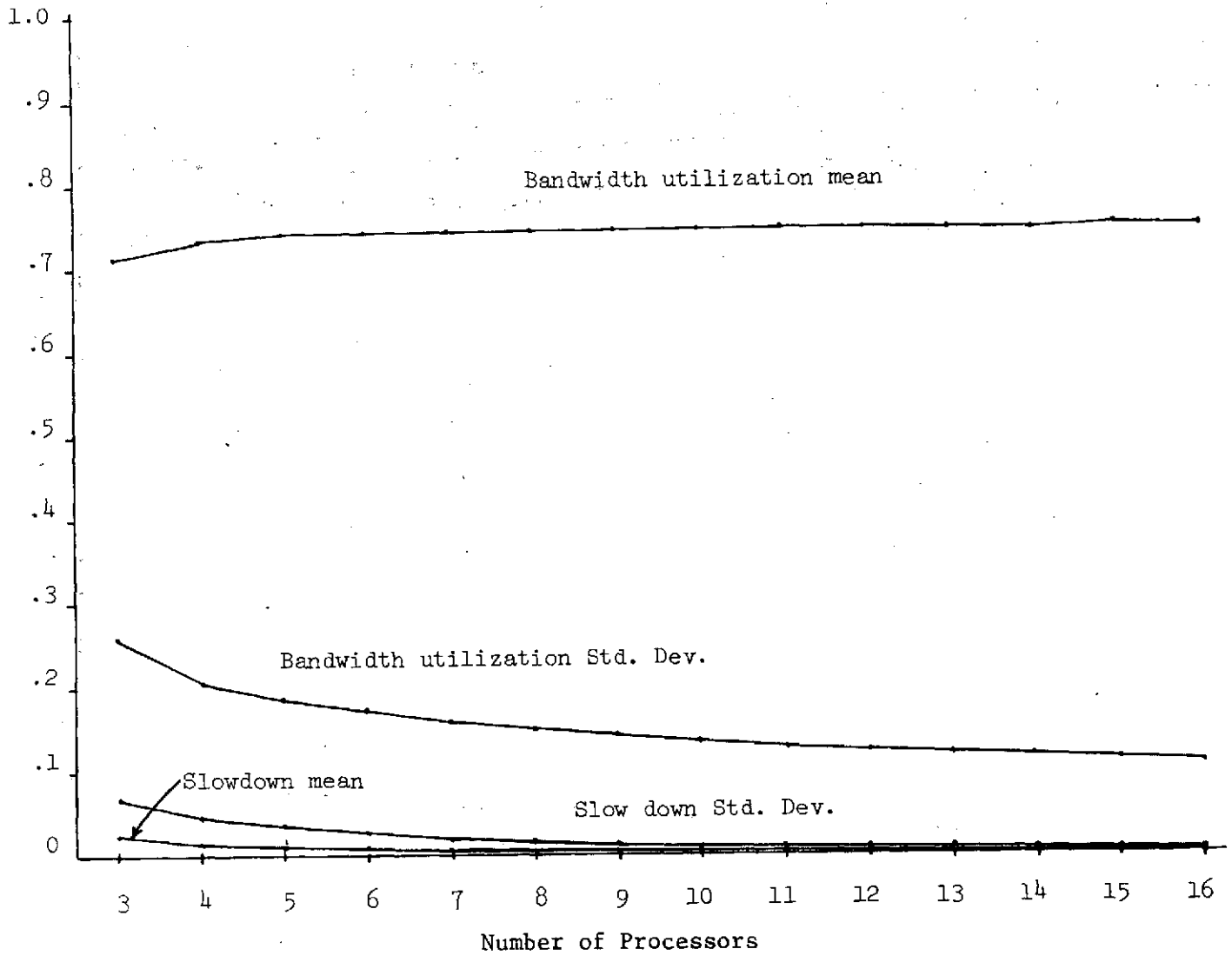


Fig. 4-15. Means and Standard Deviations For Slowdown and Bandwidth Utilization, A = 0.75.

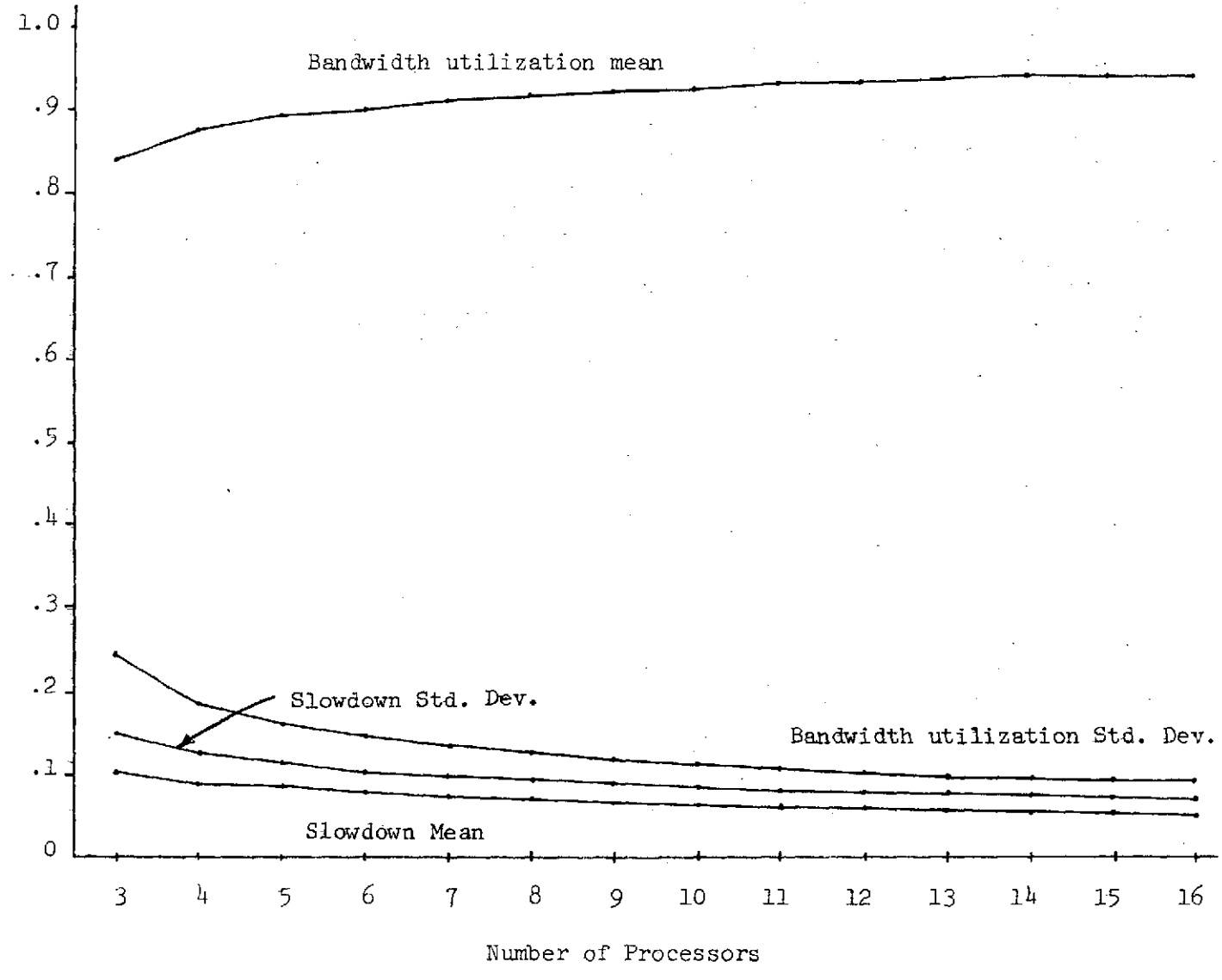


Fig. 4-16. Means and Standard Deviations for Slowdown and Bandwidth Utilization, A = 1.0.

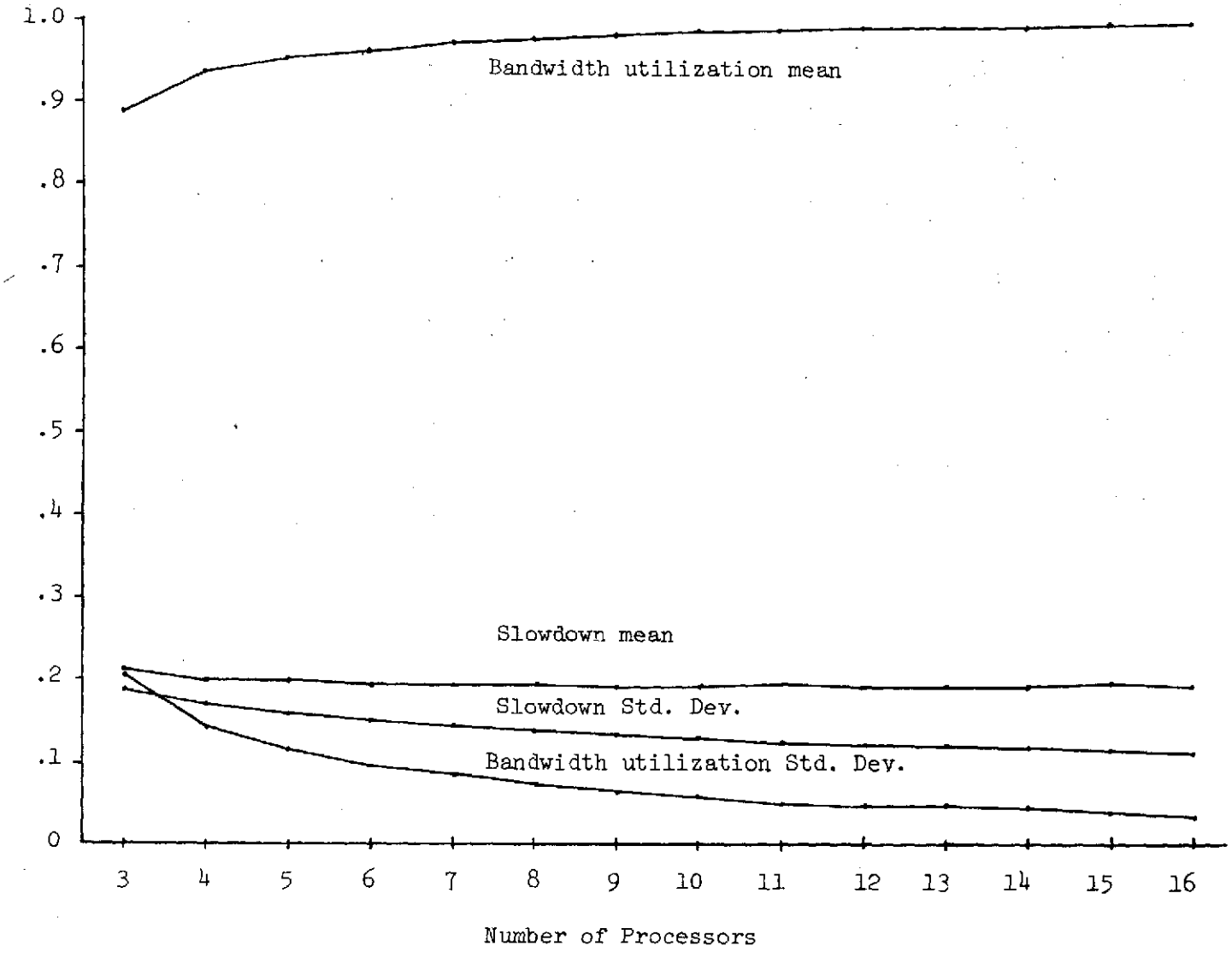


Fig. 4-17 Means and standard deviations for slow down and bandwidth utilization, A=1.25

V. THE PROCESSOR TO MEMORY INTERCONNECT PROBLEM

In Fig. 1-1 we have implied that each processor may gain access to any memory module through the interconnection network. A major requirement of this network is that it not cause any slowdown in the operation of the system; i.e., it should allow transfer of data in a random access fashion to or from the memory as fast as the maximum speed of the memory. Furthermore, access to each memory module should be made on a memory module cycle basis so as to avoid potential waste of memory bandwidth that is inherent in schemes which make a processor to module connection with minimum interconnection time extending over several module cycle periods.

The two hardware arrangements which appear to be able to meet these requirements, at least from a theoretical standpoint, are the multiplexed bus and the full crossbar as depicted respectively in Fig. 5-1 and 5-2. The multiplexing of the multiplexed bus could conceivably be done through some frequency diversity or angle modulation scheme, however, these techniques are not compatible with the present state of main memory logic elements nor is the transmission medium noisy enough or signalling power so severely limited as to warrant the additional complications imposed by such schemes.

Thus, we are left with time multiplexing. The time multiplexed bus requires less hardware to implement than the full crossbar, but

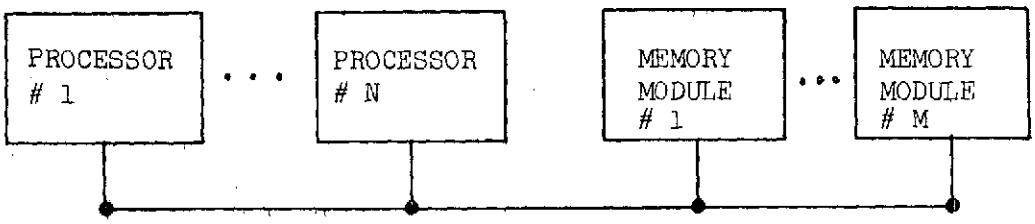


Fig. 5-1 Logic structure of time multiplexed bus

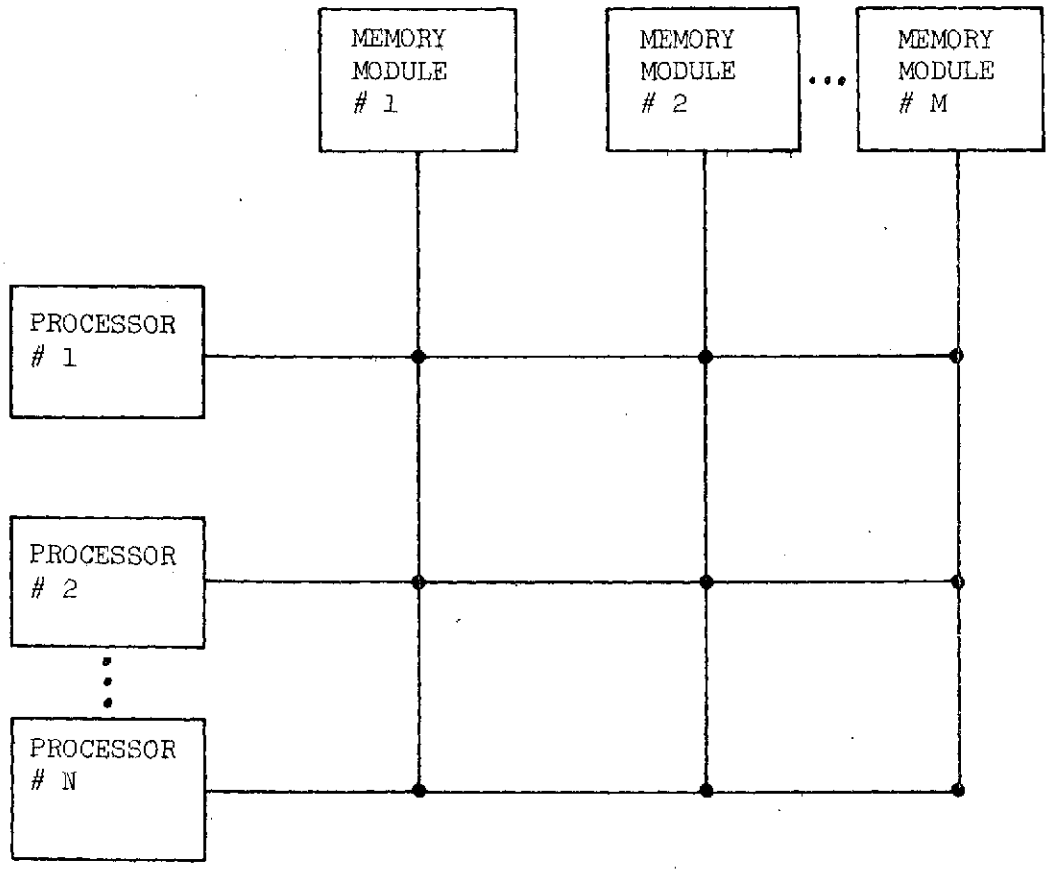


Fig. 5-2 Logic structure of full crossbar

more stringent speed requirements are implied by the time multiplexed bus since only one processor can be connected to one of the memory modules at a time. Therefore, in making the interconnections, it is sequential by module or by processor depending upon the end of the bus from which the arrangement is viewed. The full crossbar requires more hardware to implement than the time-multiplexed bus but its speed requirements are much less severe when several processors and memory modules must be interconnected. Moreover, the additional hardware is repetitive with a regular structure. This regularity in logic networks is amenable to large scale integration techniques. Hence the cost of the full crossbar does not necessarily increase directly with the number of processors and memory modules which must be interconnected. For these reasons the full crossbar scheme is considered to be the best candidate for the multiprocessor system even though it may not be quite as flexible as a bus system in regards to ease of expansion.

The crossbar interconnection arrangement is equivalent to a set of switches with one switch per memory module as shown in Fig. 5-3. This version will be used in developing an interconnection scheme for a priority driven multiprocessor.

In a priority driven system each job that is executed by the system obtains use of the system's facilities on a demand basis according to its priority relative to that of any other jobs being simultaneously executed by the system. Thus in the context of use of the memory bandwidth each job that is being processed by the multiprocessor system

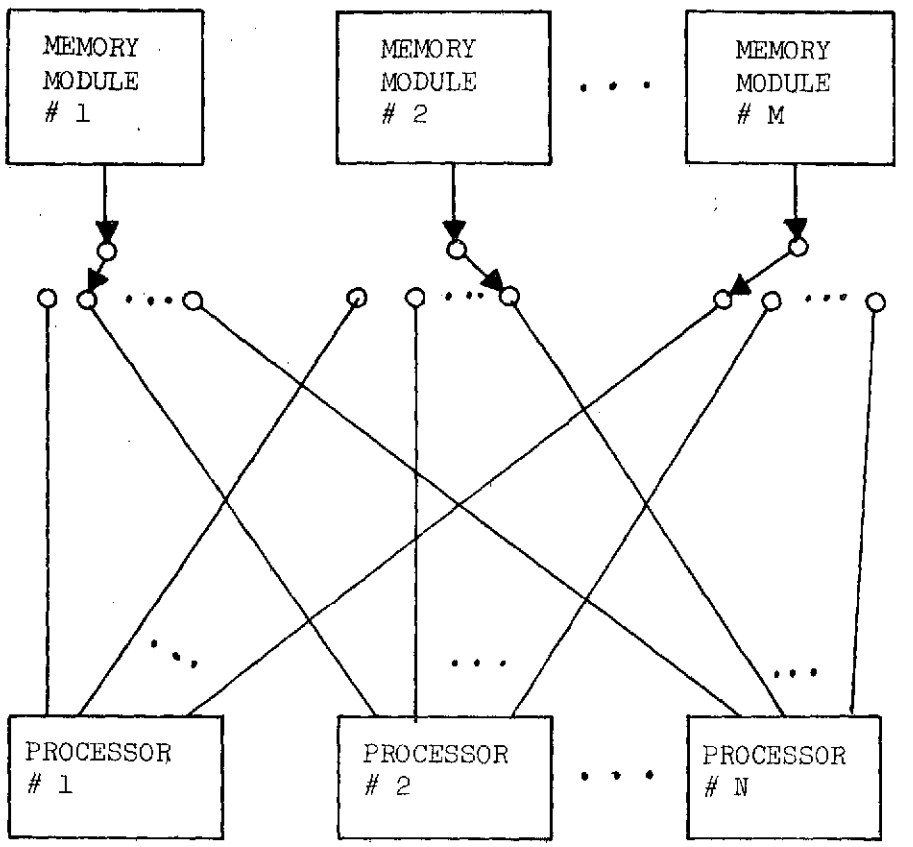


Fig. 5-3 An equivalent circuit of the full crossbar.

should be able to obtain accesses from the memory modules in which it has space approximately in accordance with its relative priority and as it so demands in time. From an overall efficiency standpoint the determination as to which processor has the right of access to a given memory module at a given time should be made on a module cycle basis since if some processors do not demand an access at a certain module cycle time due to the variability of the program that they are executing then they need not be considered in the resolution of possible access conflict. Consequently lower priority jobs stand a better chance of obtaining accesses during slack periods of memory access activity by the higher priority jobs. This means, however, that the determination of which processor is to obtain the next access from each memory module must be made in less than a memory module cycle period. Thus only simple schemes implemented in hardware appear to be feasible. The scheme must also ensure that the lower priority jobs should not be locked out from accessing memory modules in which they have space for periods longer than their relative priority dictates. Thus, our scheme for determining the right of access to each memory module should be responsive not only to the relative priorities of all contenders for each access but also to some simple function of the amount of time that each job has been in contention for an access but has not yet obtained it.

In view of the above discussion consider the access right function of priority and time in contention given by

$$ACR_j(P_j, t_c) = t_c P_j$$

where t_c is the number of module access periods that job j has been in contention for an access but has not yet received it, and P_j is the priority of job j. This simple function seems to possess most of the characteristics that we require. It may be obtained simply by adding the priority of the job to the present value of the function at each access period at which job j does not obtain the access that it is contending for. Otherwise, the value of the function is set back to the priority of job j if job j receives the access that it is contending for; thereby, in either case, readying for the next access period. The contention resolution process then consists of selecting from among all jobs that are contending for an access from a particular memory module the job possessing the largest access-right word. In case of a tie some mechanism is needed to determine a winner. Let us temporarily assume that the processors are ranked according to their numbering with lowest number representing highest ranking.

Now let us see how this scheme will work through a short example. Suppose processors 1 through 4 are executing jobs A, B, C, and D with priorities 50, 100, 200, and 25 respectively from a single memory module. Furthermore, let us assume that each processor is in contention for every module access, regardless of whether or not it received the preceeding access. This represents an extreme case of bandwidth limiting but it is entirely possible in a multiprocessor. Underlines indicate the job obtaining the access in the tabulation shown in Fig. 5-4.

RANK	PRIORITY	TIME																
		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}
1	50	100	150	200	50	100	150	200	50	100	150	200	250	300	350	400	50	100	150
2	100	200	100	200	300	400	100	200	300	400	100	200	300	400	100	200	300	400	100
3	200	200	400	200	400	200	400	200	400	200	400	200	400	200	400	200	400	200	400
4	25	50	75	100	125	150	175	200	225	250	275	300	25	50	75	100	125	150	175

Period of length 12

Fig. 5-4. Tabulation of Access Right Word as a Function of Time for Priorities of 50, 100, 200, and 25.

From this tabulation it should be observed that the sequence of accesses granted becomes periodic after time T_3 and has period of length 12. During this period jobs A, B, C, and D received accesses in the ratios of 1/12, 1/6, 1/2 and 1/12 respectively. Their priorities relative to their total priority are 2/15, 4/15, 8/15 and 1/15 respectively. The next natural question is concerned with whether or not the actual ratios of accesses granted can be made to more closely approach the priority ratios by some permutation of the ranking of the processors. There is a total of $4! = 24$ different rankings of the four processors. The results for all 24 rankings are shown in Fig. 5-5. From this figure we may observe that the exact priority ratios in accesses granted may be obtained by ranking the processors according to the priorities of the jobs that they are executing.

Unfortunately, the exact priority ratios in accesses granted can seldom be obtained by ranking the processors according to the priorities of the jobs that they are executing. However, this technique does appear to result in a more equitable distribution of accesses when some of the priorities are multiples of some of the others. It has no effect when no priority is a multiple of any of the others. Thus its adaptation does no harm in the latter case. Therefore, from this point on, let us assume that the processors are first ranked according to the priorities of the jobs that they are executing and secondarily according to their numbering. In this manner we preclude the possibility of a tie.

PRIORITY	RANKING																							
50	1	1	1	1	1	1	2	2	3	3	4	4	2	2	3	3	4	4	2	2	3	3	4	4
100	2	2	3	3	4	4	1	1	1	1	1	1	3	4	2	4	2	3	3	4	2	4	2	3
200	3	4	2	4	2	3	3	4	2	4	2	3	1	1	1	1	1	1	4	3	4	2	3	2
25	4	3	4	2	3	2	4	3	4	2	3	2	4	3	4	2	3	2	1	1	1	1	1	1
Length Before Becoming Periodic	3	3	8	3	8	8	9	9	9	9	9	9	8	8	8	8	8	8	3	8	9	8	9	8
Periodic Length	12	12	12	12	12	12	14	14	14	14	14	14	15	15	15	15	15	15	8	8	8	8	8	8

ACCESS FRACTION

Priority	Fraction	Periodic Length 8	12	14	15
50	2/15	1/8	2/12	2/14	2/15
100	4/15	2/8	3/12	4/14	4/15
200	8/15	4/8	6/12	7/14	8/15
25	1/15	1/8	1/12	1/14	1/15

Fig. 5-5. Tabulation of Access Mechanism for All Processor Ranking with Four Jobs of Priorities 50, 100, 200, and 25.

In the preceeding example all the priorities were multiples or divisors of each other. However, we would like to know what happens when the priorities are not multiples, or divisors of each other. In this regard let us observe the following facts about this scheme for the resolution of access conflicts.

(1) Under the assumption that N processors simultaneously begin contending for an access at time 1 with their access right words all equal to the priority of the jobs that they are executing, and that all contend for every access, the length (i.e., no. of memory module cycle periods) before the lowest priority job obtains its first access, designated by LBLP, satisfies the relation

$$\mathrm{MAX}\left(N, \left\lceil \frac{P_H}{P_L} \right\rceil + 1\right) \leq \mathrm{LBLP} \leq \left\lceil \frac{(N-1)P_H}{P_L} \right\rceil + 1$$

where P_H is the largest priority, P_L is the smallest priority, and $\lceil x \rceil$ indicates the smallest integer $\geq x$.

This is easily verified since the lowest priority job cannot obtain an access before any of the higher priority jobs nor before its access right word is greater than the highest priority. Thus the left-hand portion of the relation must hold. Concerning the right-hand portion of the relation, it is observed that, considering the N-1 highest priority jobs alone, the largest value that any of their access right words may reach before the lowest priority job obtains its first access is $(N-1)P_H$. Thus the largest value that the lowest priority job's

access right word must exceed before obtaining its first access is $(N-1)P_H$.

(2) It is difficult to make improvements on the bounds in (1) without recourse to the exact ratios of all the priorities involved. An analytical formulation in terms of these priority ratios and number of contending processors appears to be a formidable problem in itself and would actually add very little to the task at hand. The results of a statistical analysis would be more useful in gaining an overview of the manner in which the access granting mechanism is expected to work.

(3) The proposed scheme performs poorest when two processors are continuously contending and $P_H = 2P_L-1$. In this case the two jobs obtain alternate accesses even though their ratio of priorities is much closer to 2 to 1 than to 1 to 1. However, this is not as bad as it may at first appear since in a real situation neither of the processors would be expected to be contending at every access period. From the discussion of Chapter III concerning two contending processors and memory module bandwidth equal to the sum of the mean bandwidth requirements of the two jobs being executed by the two processors and each of equal bandwidth requirements the probability of contention at time t+i, provided contention occurred at t, t+1, . . . t+i-1, is of the form $(1/2)^i$. Thus, the probability of continuous contention of length 1 is twice that of length 2, four times that of length 3, etc.

On the other hand the problem in the special case of two processors could be alleviated somewhat by adding only a fraction of the priority to the access right word of the processor not obtaining the access when the priorities are of certain ratios. However, this represents further complication to the overall mechanism with attendant potential for slowdown of the system. Therefore it will not be considered any further in this paper.

The results of a statistical analysis of the length before the lowest priority job obtains its first access and the length between the first and second access is shown in Fig. 5-6 through Fig. 5-17 for 2, 3, 5, 10, 13 and 16 processors with priorities ranging from 1 through 255. The frequency-grams of these figures were plotted for 1000 samples each from a uniformly distributed random variable. The values were normalized about the integer ratio of the sum of the priorities to the lowest priority. From these frequency-grams it may be observed that the expected mean length before the lowest priority job obtains its first access is related to the number of contending processors as well as the normalizing factor $NF = \left[\sum_i P_i/P_L\right]$. The length between the first and second access by the lowest priority job is also related to the number of contending processors and the normalizing factor, but the frequency grams indicate that this length is slightly less than that before the first access by the lowest priority job. These two parameters give an indication of the type of service that may be expected by a random selection of jobs being simultaneously

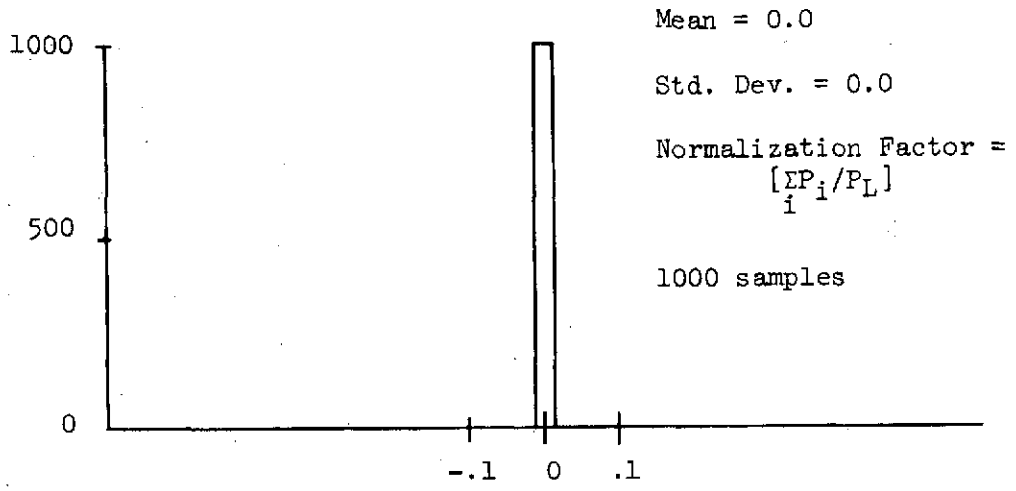


Fig. 5-6 Frequency-Gram for length before lowest priority job obtains first access for 2 processors.

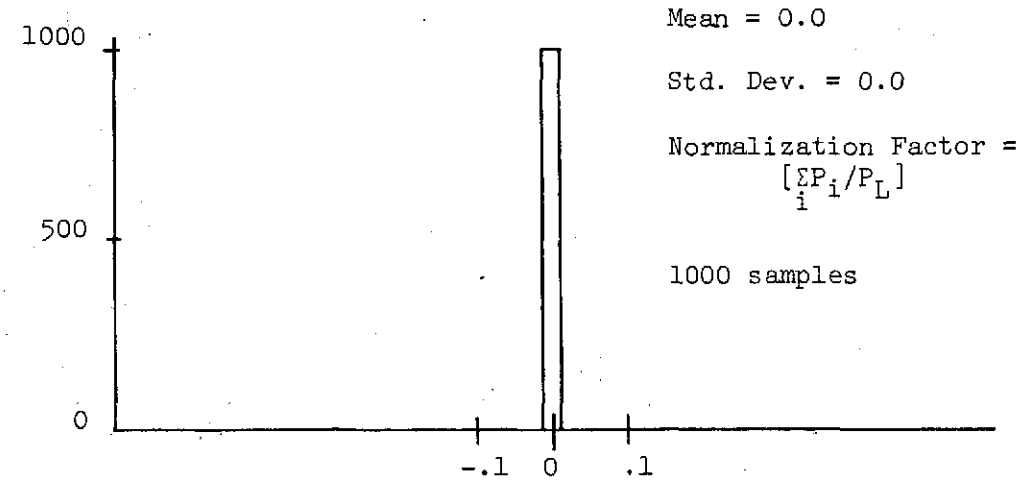


Fig. 5-7 Frequency-Gram for length between 1st and 2nd access by lowest priority job, 2 processors.

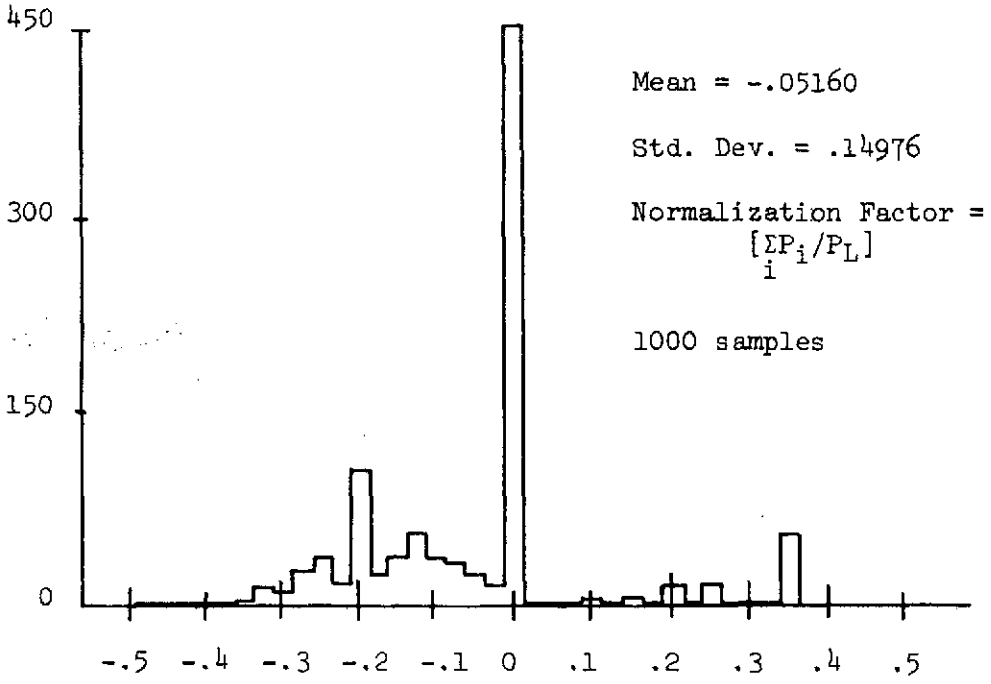


Fig. 5-8 Frequency-Gram for length before lowest priority job obtains first access for 3 processors

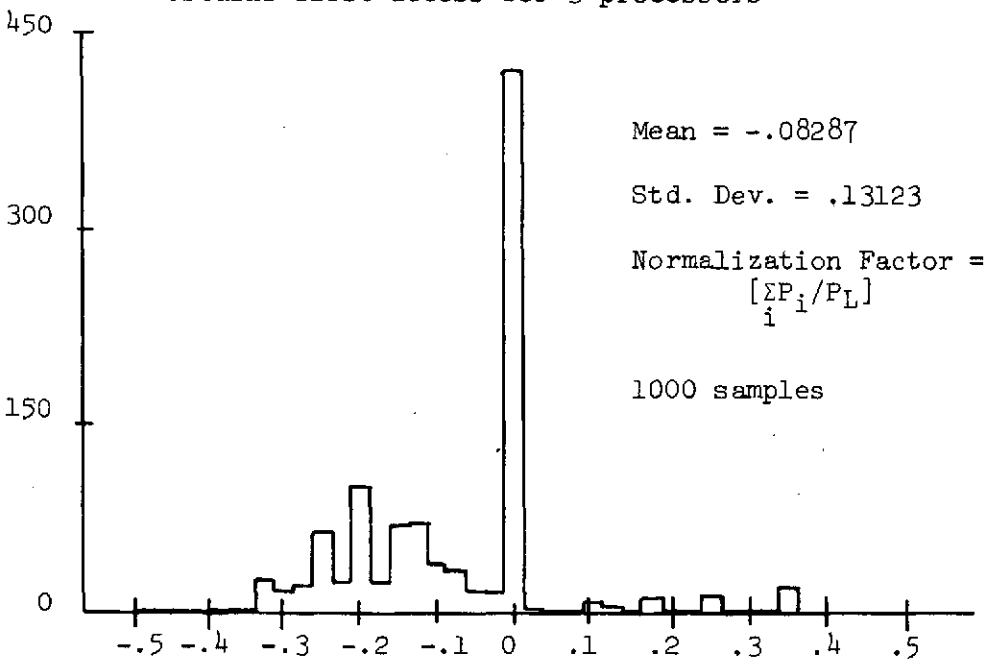


Fig. 5-9 Frequency-Gram for length between 1st and 2nd access by lowest priority job, 3 processors.

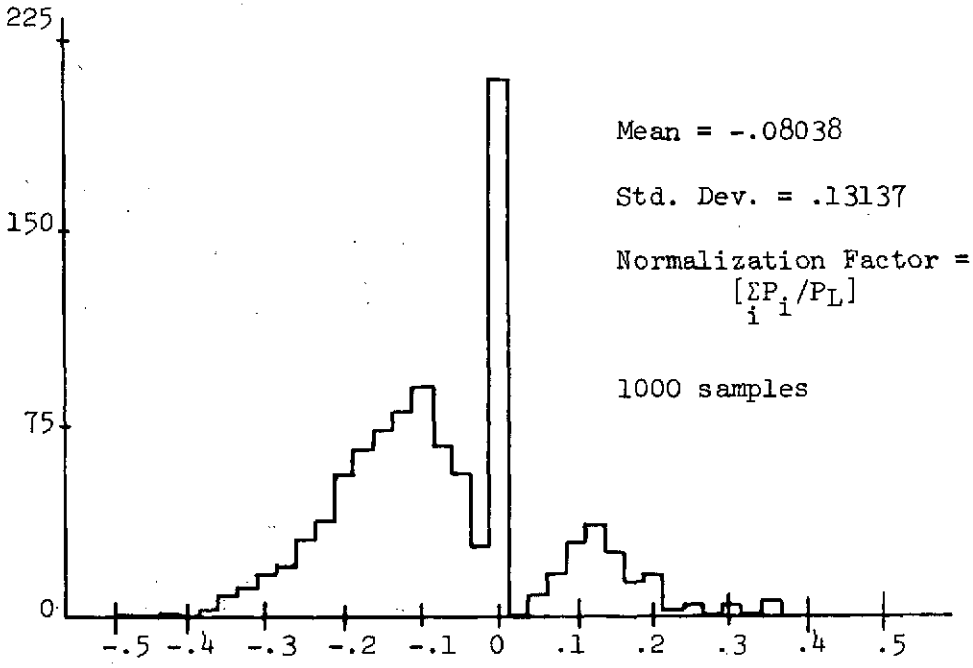


Fig. 5-10 Frequency-Gram for length before lowest priority job obtains first access for 5 processors

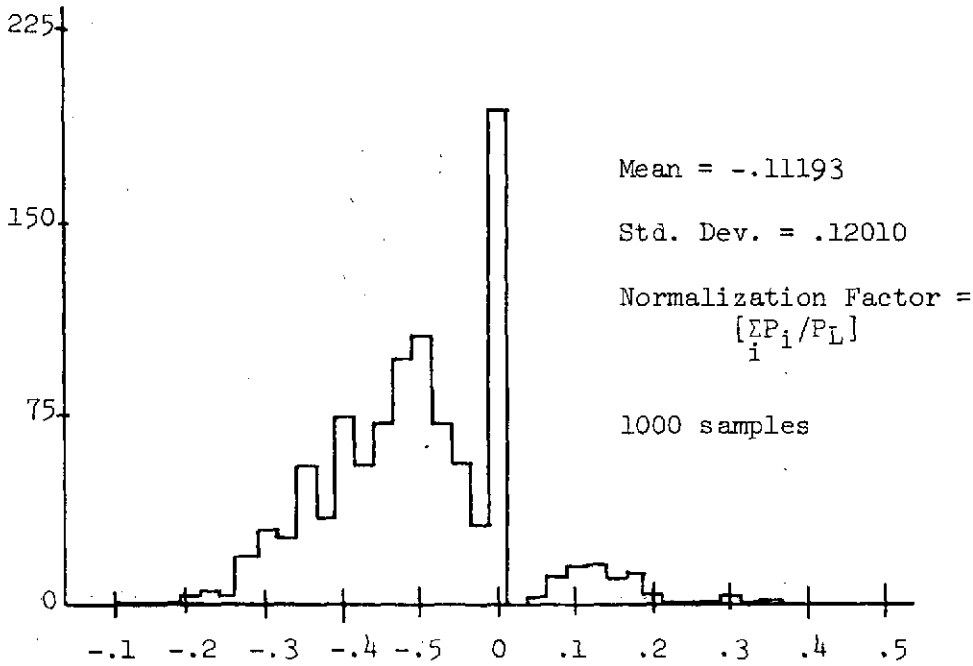


Fig. 5-11 Frequency-Gram for length between 1st and 2nd access by lowest priority job, 5 processors.

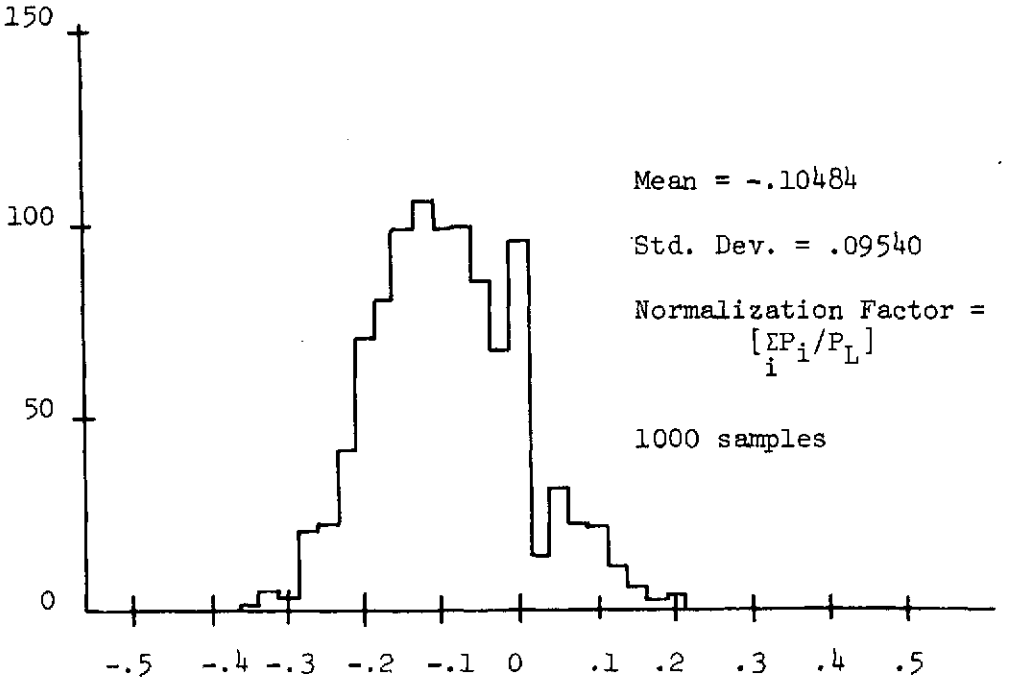


Fig. 5-12 Frequency -Gram for length before lowest priority job obtains first access for 10 processors

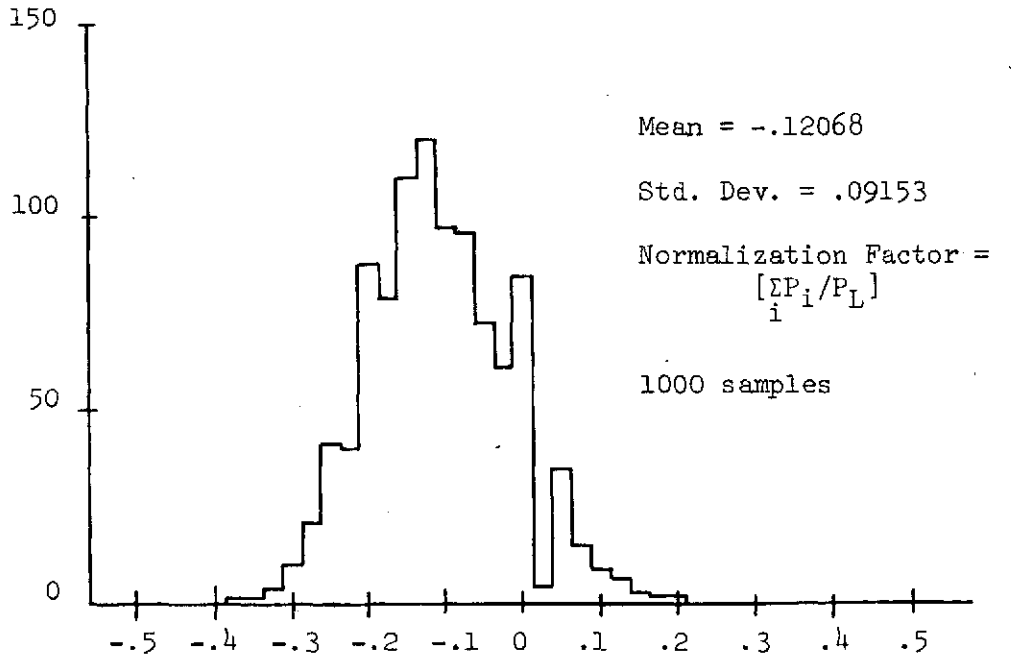


Fig. 5-13 Frequency-Gram for length between 1st and 2nd access by lowest priority job, 10 processors.

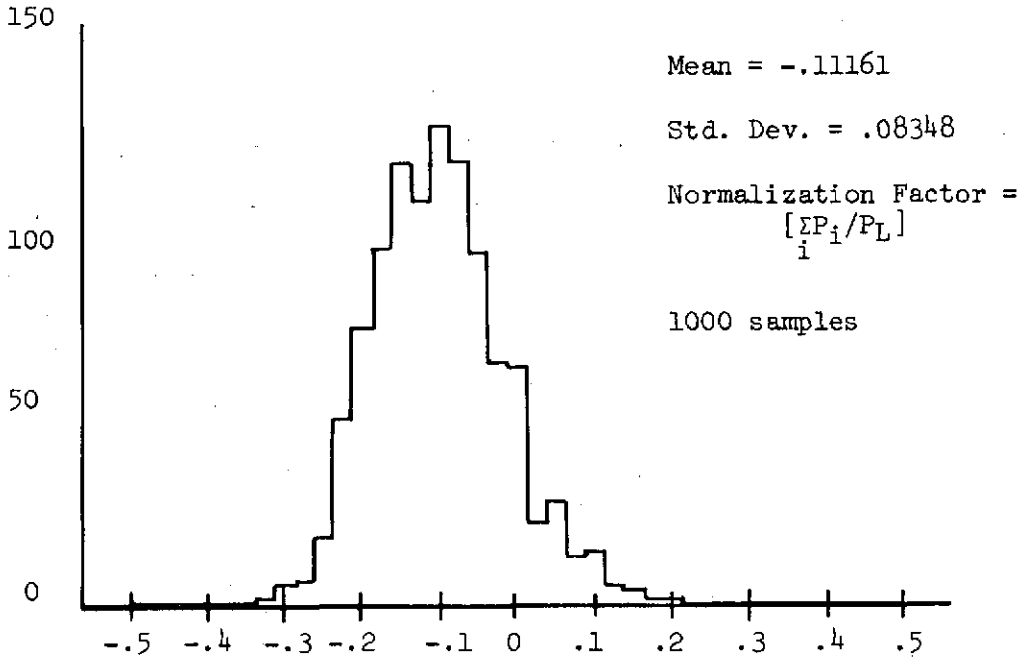


Fig. 5-14 Frequency-Gram for length before lowest priority job obtains first access for 13 processors

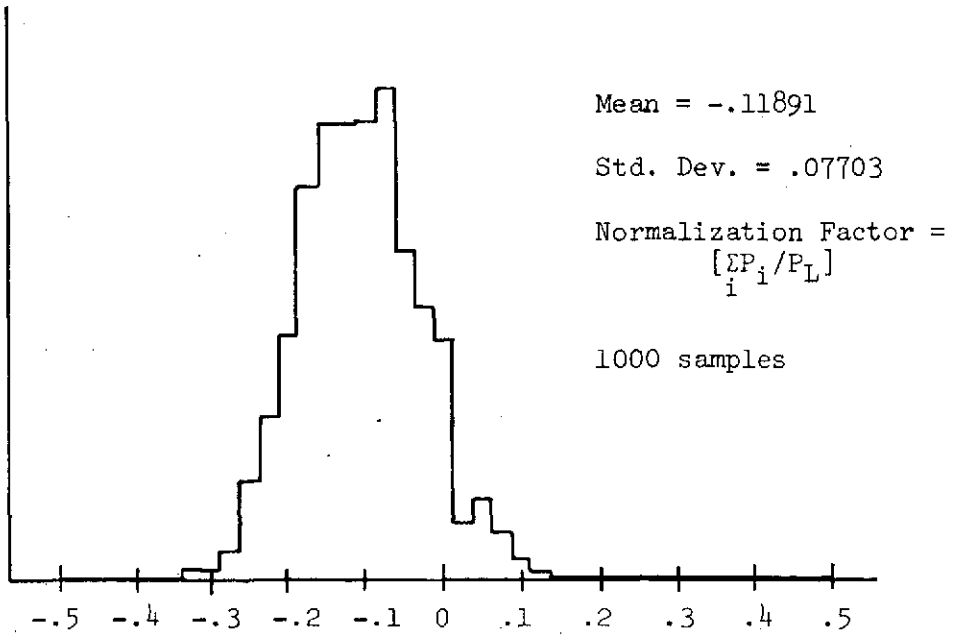


Fig. 5-15 Frequency-Gram for length between 1st and 2nd access by lowest priority job, 13 processors.

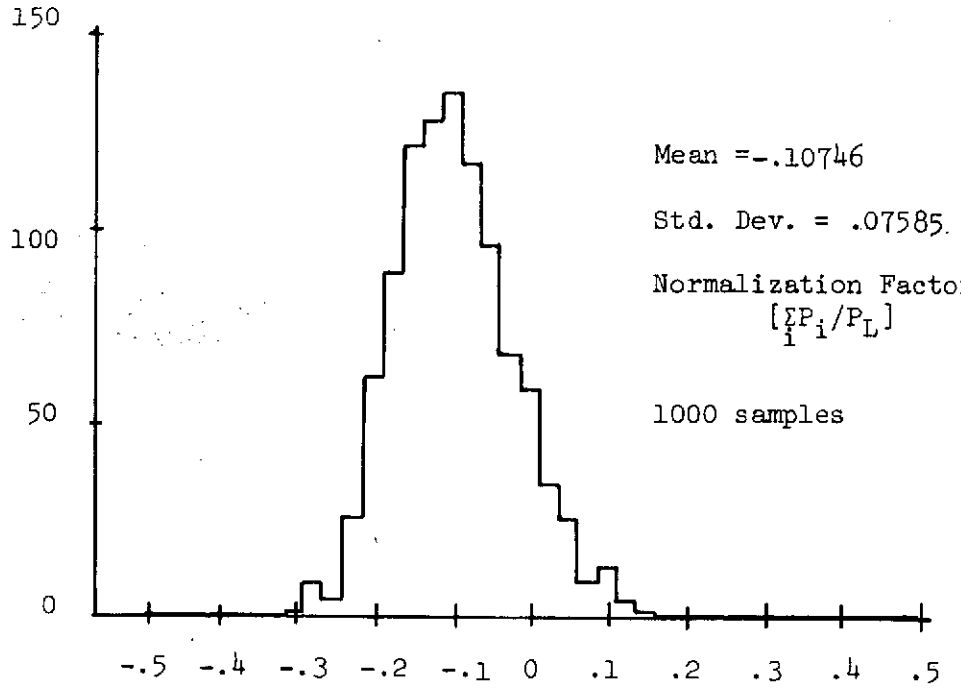


Fig. 5-16 Frequency-Gram for length before lowest priority job obtains first access for 16 processors.

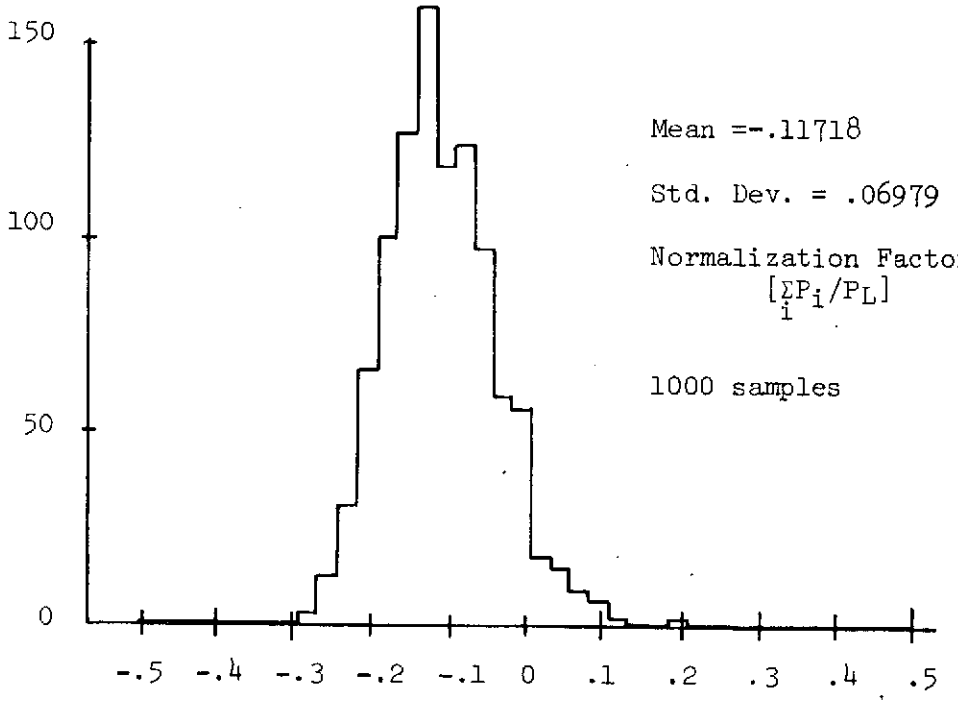


Fig. 5-17 Frequency-Gram for length between 1st and 2nd access by lowest priority job, 16 processors.

executed from a single memory module. The lower priority jobs will tend to obtain accesses at higher rates than their relative priority dictates while the higher priority jobs will tend to receive accesses at rates that are slightly less than their relative priority indicates that they should. This is the expected outcome over the ensemble of priorities and number of contending processors but particular cases may be contrary to this average trend as is indicated by the sequence

80	80	160	80	160	80	160	
53	106	53	106	159	212	53	. . .
53	106	159	212	53	106	159	.

periodic length

Nevertheless, the graphs indicate that the mean length before first access by the lowest priority job ranges from approximately -.052NF with standard deviation of .150NF for three processors to approximately -.107 NF with standard deviation of .076 NF for sixteen processors. The mean and standard deviation for the length between the first and second access by the lowest priority job ranges from approximately -.083 NF and .131 NF for three processors to approximately -.117 NF and .070 NF for sixteen processors. The priorities were always assumed to range from 1 through 255. The two processor case is singular with mean equal to NF and standard deviation of 0.0 since the normalizing factor NF was taken to be the integer part of $\sum_i P_i/P_L$. The curves appear to approach a normal distribution as the number of processors increase.

The results for the means and standard deviations are summarized in Fig. 5-18 and Fig. 5-18A.

A block diagram of the multiprocessor system with the proposed scheme for memory access conflict resolution included is shown in Fig. 5-19. In the context of conflict resolution no explicit distinction has been made between CPU's and IOP's. However, it may be desirable to give the IOP's higher ranking than the CPU's as well as weighting the priority words associated with jobs in the I/O processing state higher than that of jobs in the CPU processing state due to the critical nature of the timing required of input/output transfers. Also, in this context, it appears that each IOP will require a small dedicated buffer storage since the scheme does not guarantee right of memory access to any particular processor at a particular time. However, these requirements are not considered in this paper. As shown in Fig. 5-19 associated with each processor is a memory module address decoder which accepts as input an access request line labeled A and the memory module address lines. If the number of memory modules is a power of 2 and the $\log_2 M$ least significant digits of the address word is taken as the module address then interleaved memory results, but this is not considered to be the case at this point. Thus, we assume that the $[\log_2(M)]$ high order bits of each address word is the module address. From each of the N memory module address decoders is a line to each of the M processor selectors. Therefore, each of the M processor selectors has as input one line from each of the N address

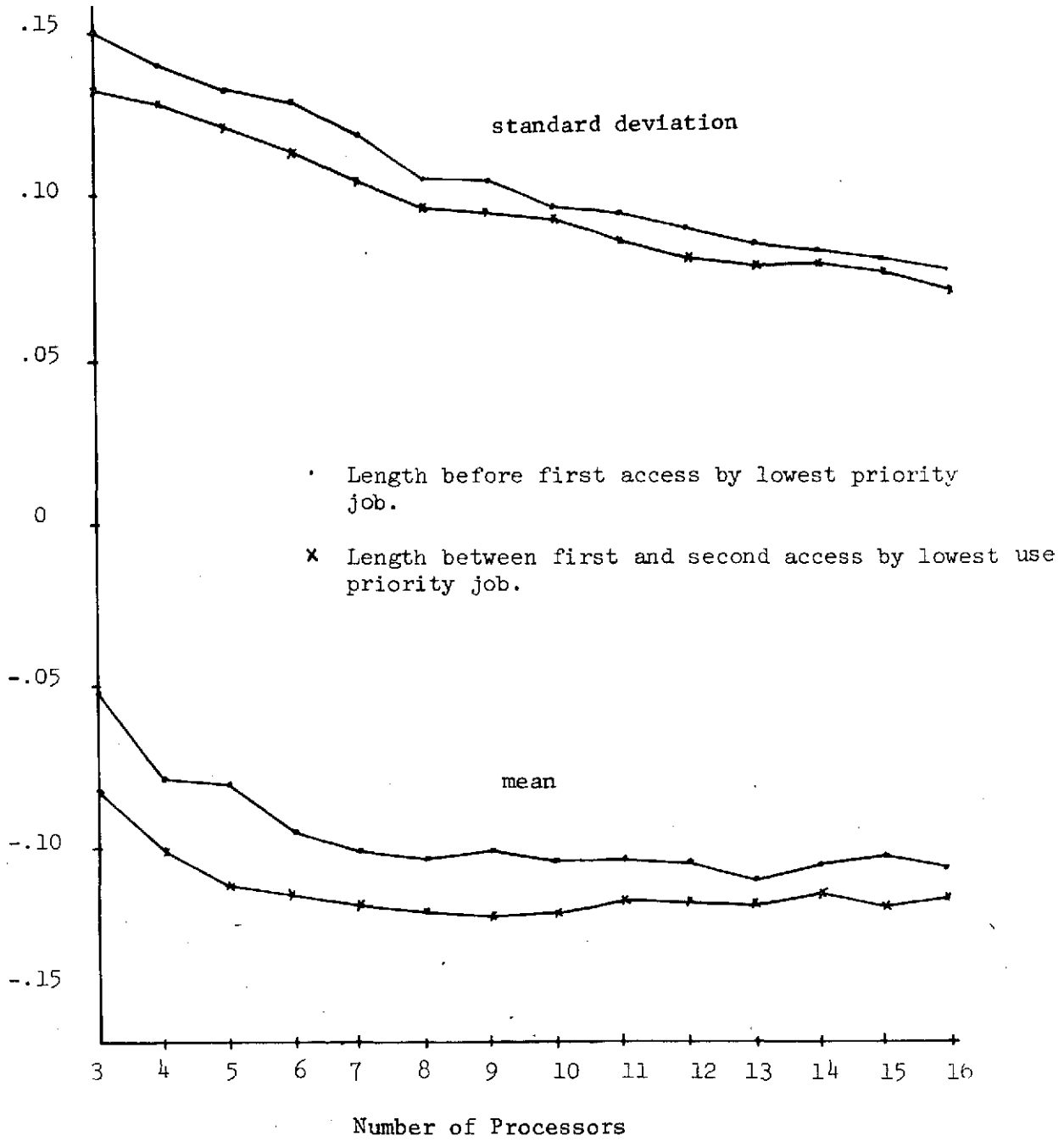


Fig. 5-18 Summary of Means and Standard Deviations normalized about $[\sum_i P_i/P_L]$ for Priority Driven Access Conflict Resolution Scheme.

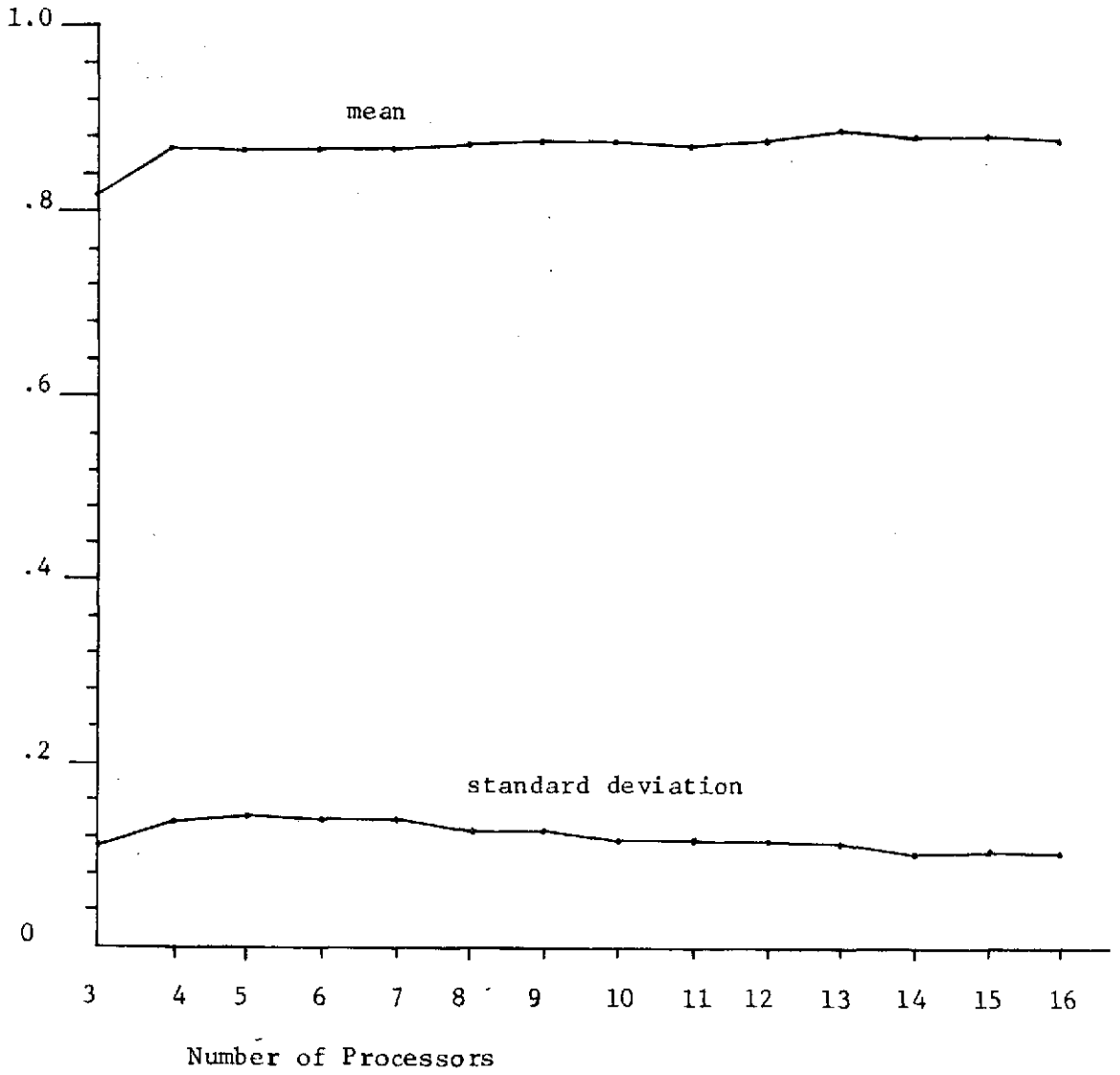


Fig. 5-18A Service to highest priority job relative to that of the lowest priority job under the priority criterion.

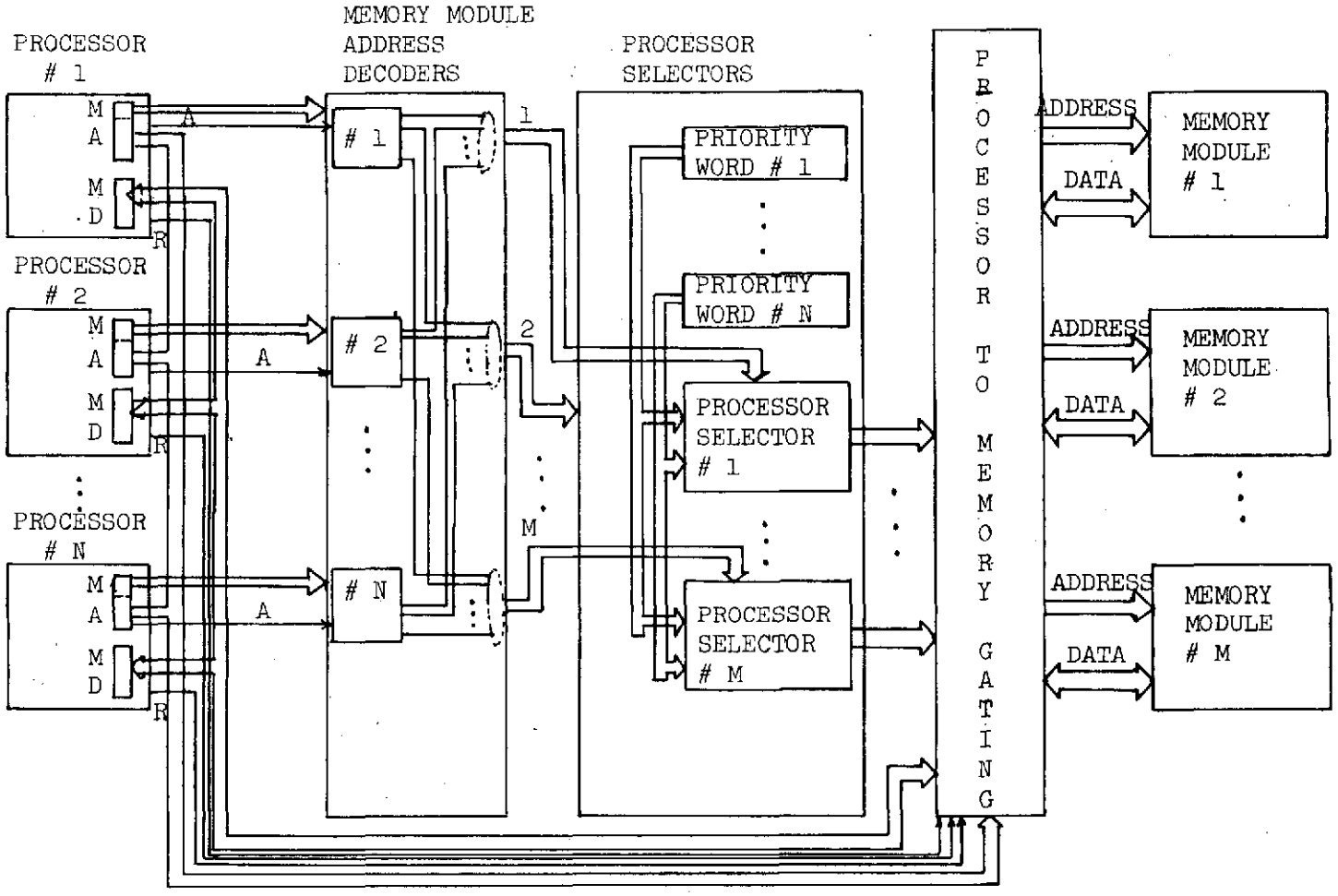
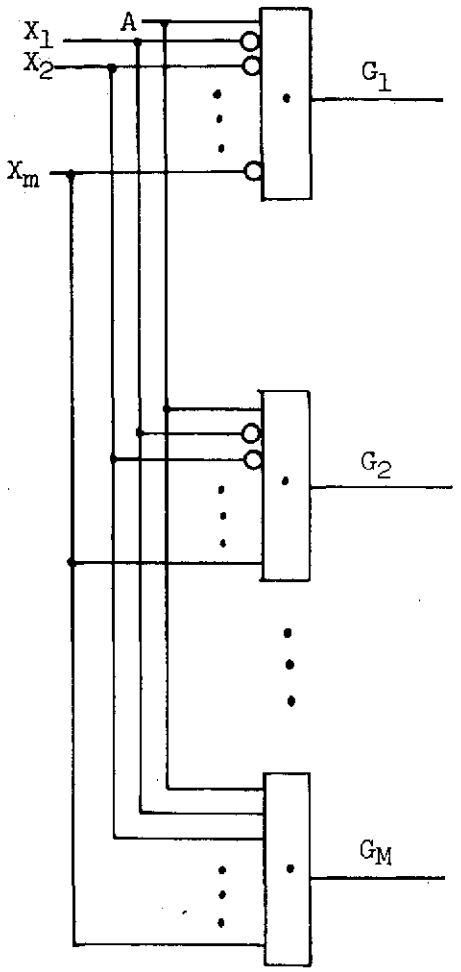


Fig. 5-19. Block Diagram of the Multiprocessor System with Conflict Resolution Scheme Included.

decoders as well as p lines from each of the N priority words. There is one processor selector associated with each of the memory modules which selects at most one of the processors to be given an access at each memory module cycle period. The output of the M processor selectors control the gating between processors and memory modules. Fig. 5-20 shows a memory module address decoder. Fig. 5-21 shows the combinatorial portion of a processor selector. The access word registers are capable of either adding the priority word at their inputs to their present contents or loading the priority word at their inputs at the times controlled by the memory module sequence timing and interconnect controller. The N inputs from the memory module address decoders control the gating of the contents of the access right words into the maximum word detector and only those lines corresponding to processors wanting an access from the memory module associated with the processor selector under consideration will be true. The maximum word detector compares the most significant bit of all contending access words with the maximum size for this bit position. If a particular word's most significant bit is not as large as the largest then the output of the coincidence gate associated with this word and bit position will be zero. Consequently, any words not meeting the largeness test in their most significant bit position will not be considered at the lower bit positions since the output of the coincidence gates are also used as inputs to the input AND gates at the next lower bit position. The same test is then applied to the



$m = \text{Log}_2 \lceil M \rceil$

A = access request line

Fig. 5-20. Memory Module Address Decoder.

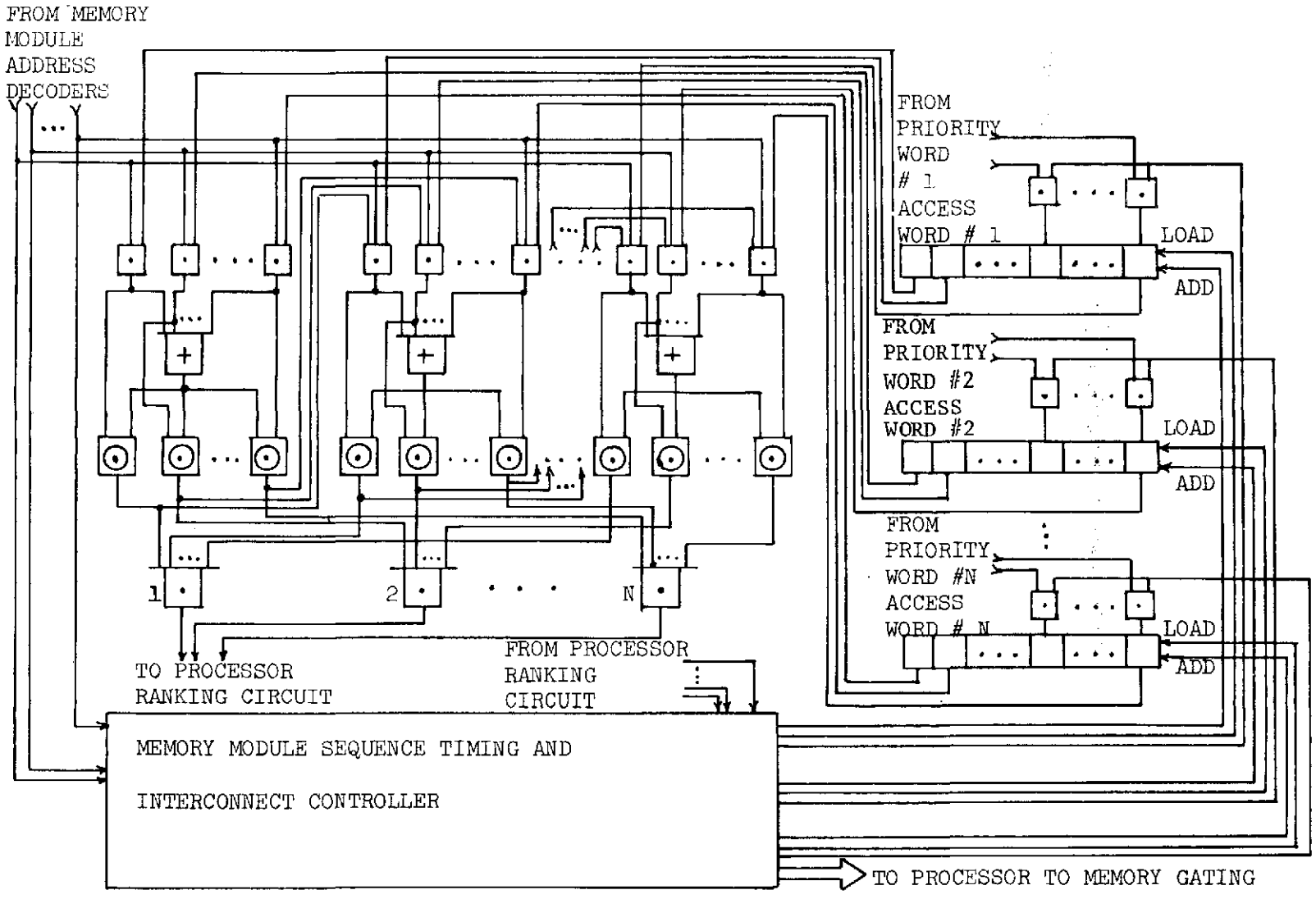


Fig. 5-21. Processor Selector.

next most significant bit position of each contender that has passed the largeness test at its preceeding more significant bit position. Thus, the outputs from the AND gates going to the processor ranking circuit will be true only for those words that were as large as the largest and were contending.

The processor ranking circuit shown in Fig. 5-22 is used to determine a winner in case of a tie from the processor selector. Its inputs consist of the output of the processor selector for gating through the priority word associated with those access words that are as large as the largest. The processor ranking circuit works in a manner identical to that of the max word selector except the outputs are ordered according to processor number i.e. in case two or more contending processor's priority words are equal then the one with lowest number obtains the access provided its associated access right word was equal to the largest access right word. The outputs of the processor ranking circuit is used as inputs to the memory module sequence timing and interconnect controller. The sequence timing and interconnect controller provides timing signals that coordinate the selection of the processor to obtain the next access and timing of the various events associated with the operation of the memory module.

Fig. 5-23 shows the gating of the read/write line from each processor to each memory module. The inputs to these circuits consist of one line from each of the M module sequence timing and interconnect controllers. Fig. 5-24 shows the logic gating for one bit of the data

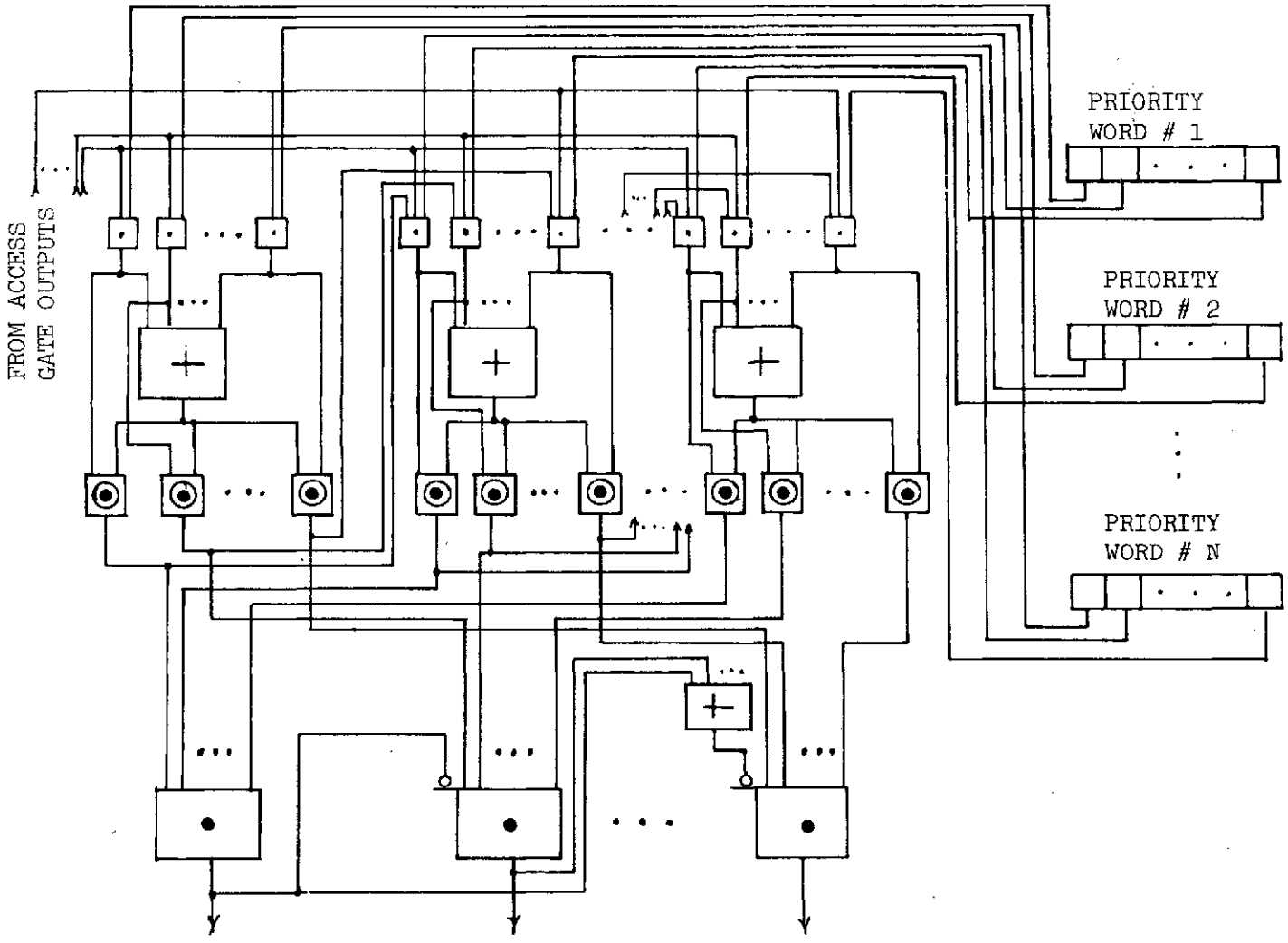


Fig. 5-22. Processor Ranking Circuit.

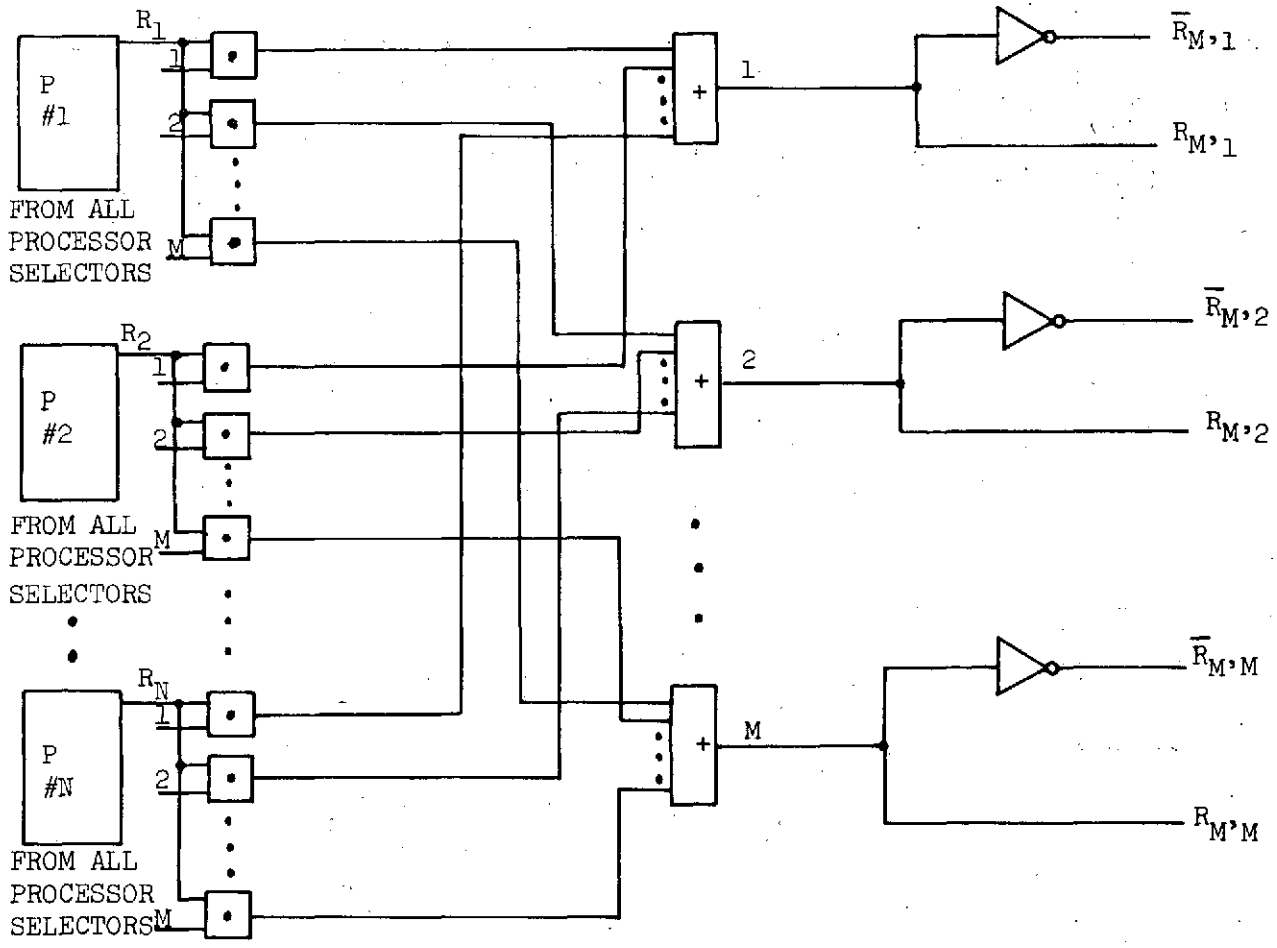


Fig. 5-23. Processor to Memory Gating (Read/Write Lines).

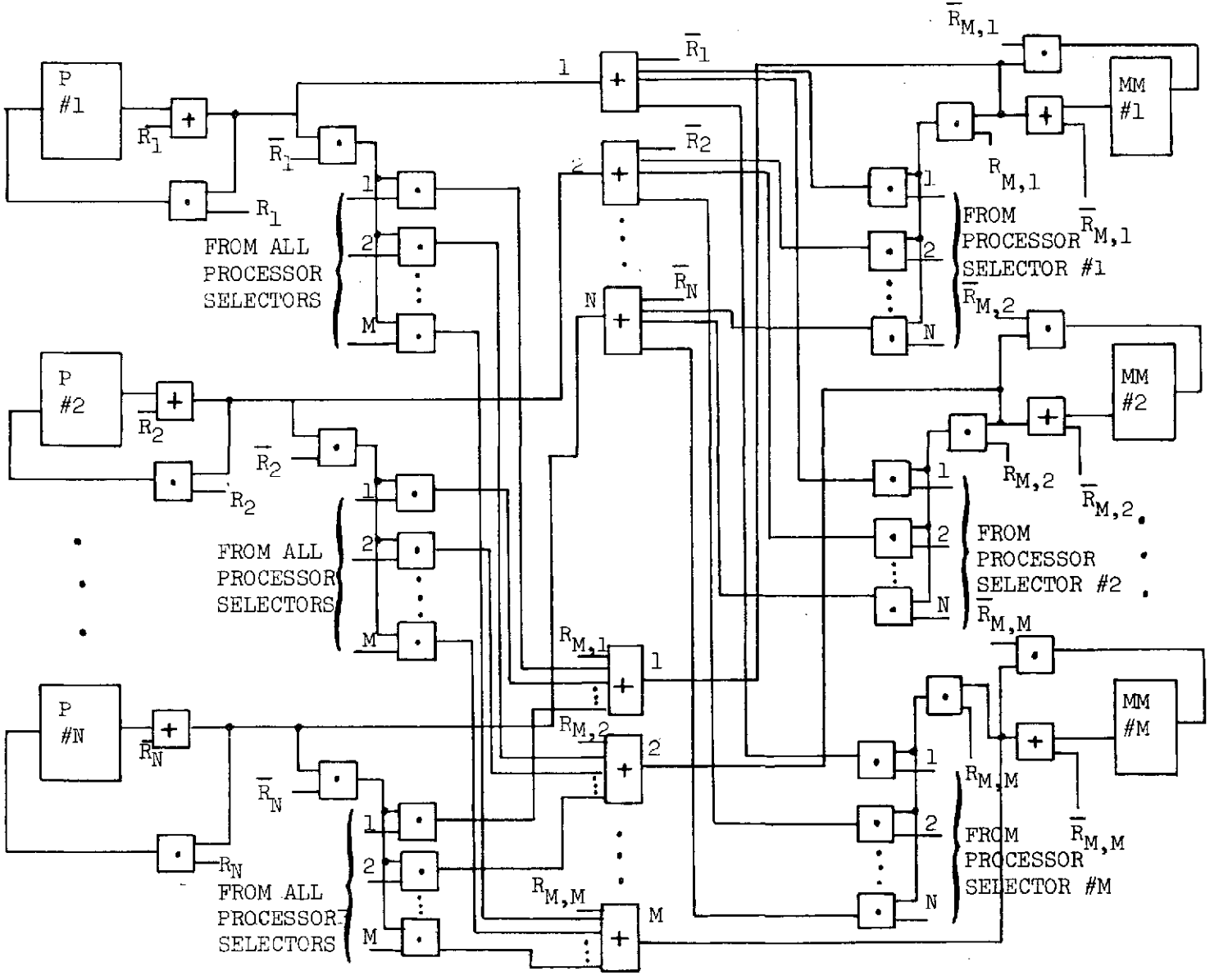


Fig. 5-24. Processor to Memory Gating (One Bit of Data Lines).

lines between the processors and memory modules. In places where the outputs of gates are connected together it is assumed that if one of the gates tends to cause the output to be true and the other tends to cause it to be false then the result is false. Fig. 5-25 shows the gating of the address words from the processors to the memory modules. The inputs to these circuits consist of one line from each of the M sequence timing and control circuits. At most one of the M can be true at a given access period.

Time Required to Resolve Access Conflict

Let us assume the following:

(1) At most 16 processors will be contending for accesses from a single memory module.

(2) The largest priority word is 255.

From (1) and (2) the number of bits required for an access right word is $[\log_2(16x255)] = 12$. If it is assumed that the coincidence gates of Fig. 5-21 are realized with three levels of gating then the ripple time for the max word detector (combinational logic portion of the processor selector) Λ is $5 \times 12 + 1 = 61$ gate delays. Similarly the ripple time through the processor ranking circuit is $5 \times 8 + 2 \times 7 = 54$ gate delays. Thus a total of 115 gate delays is required for resolution of conflict. If we assume a gate delay of 5ns then the total time for access conflict resolution is 575 ns. This amount of time is probably too large for most high speed memory modules. The problem here is that these circuits operate on a sequential bit by bit manner in selecting

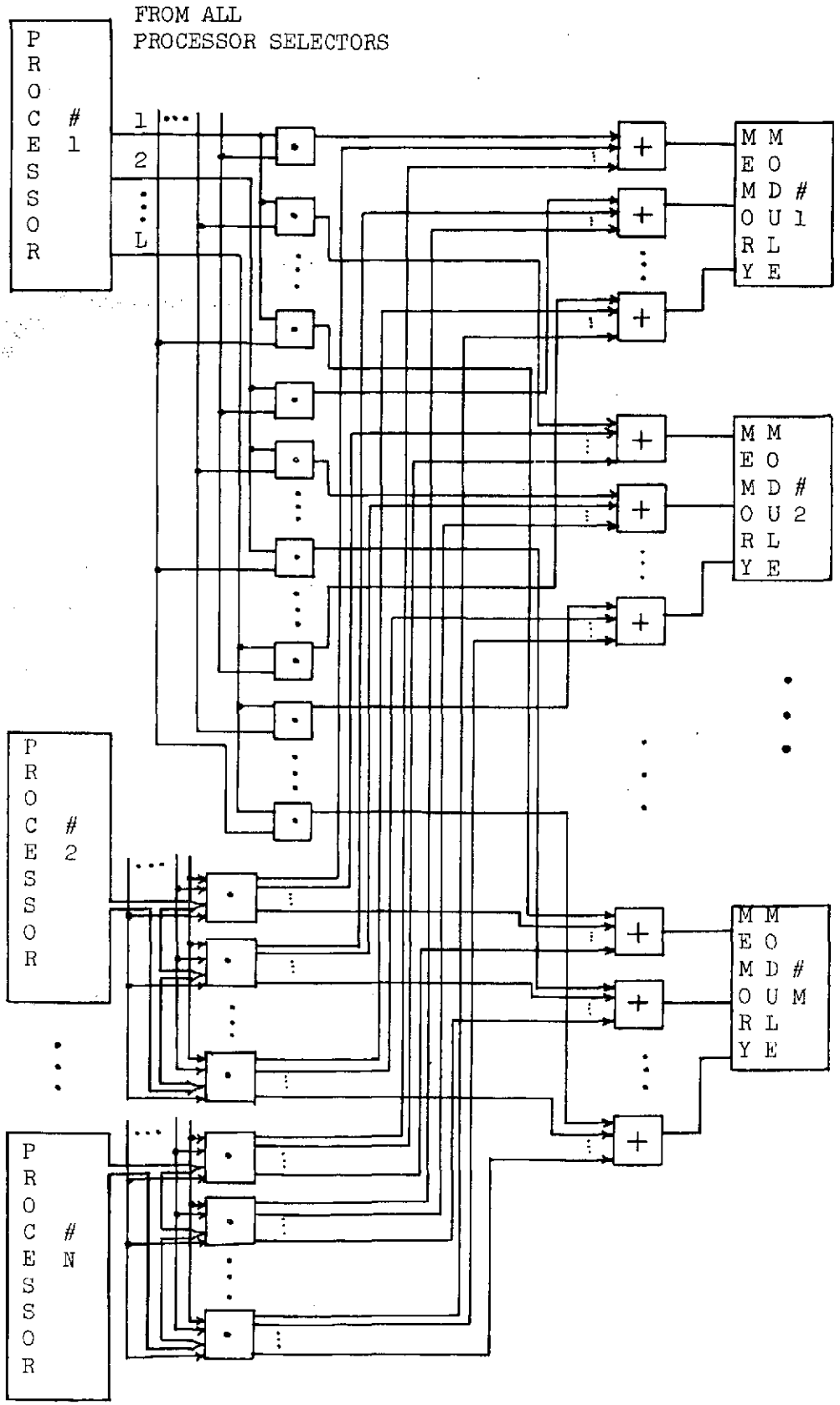


Fig. 5-25. Processor to Memory Gating (Address Lines).

the largest word. If the number of bits in the access right and priority words were smaller then these circuits might be satisfactory. Nevertheless, let us consider an alternate arrangement that makes the selections in an essentially parallel manner. The number of gate delays for the circuits shown in Fig. 5-26 and Fig. 5-27 is independent of the number of bits in the access right or priority words. The exactly 1 out of N and as large as largest circuits are shown in Fig. 5-28. Under the assumption that the coincidence gates are realized with three gate levels then the total number of gate delays for the circuits of Fig. 5-26 and Fig. 5-27 is at most 17. If it is assumed that the amount of time delay per gate is 5ns then these circuits are capable of access conflict resolution in 85 ns.

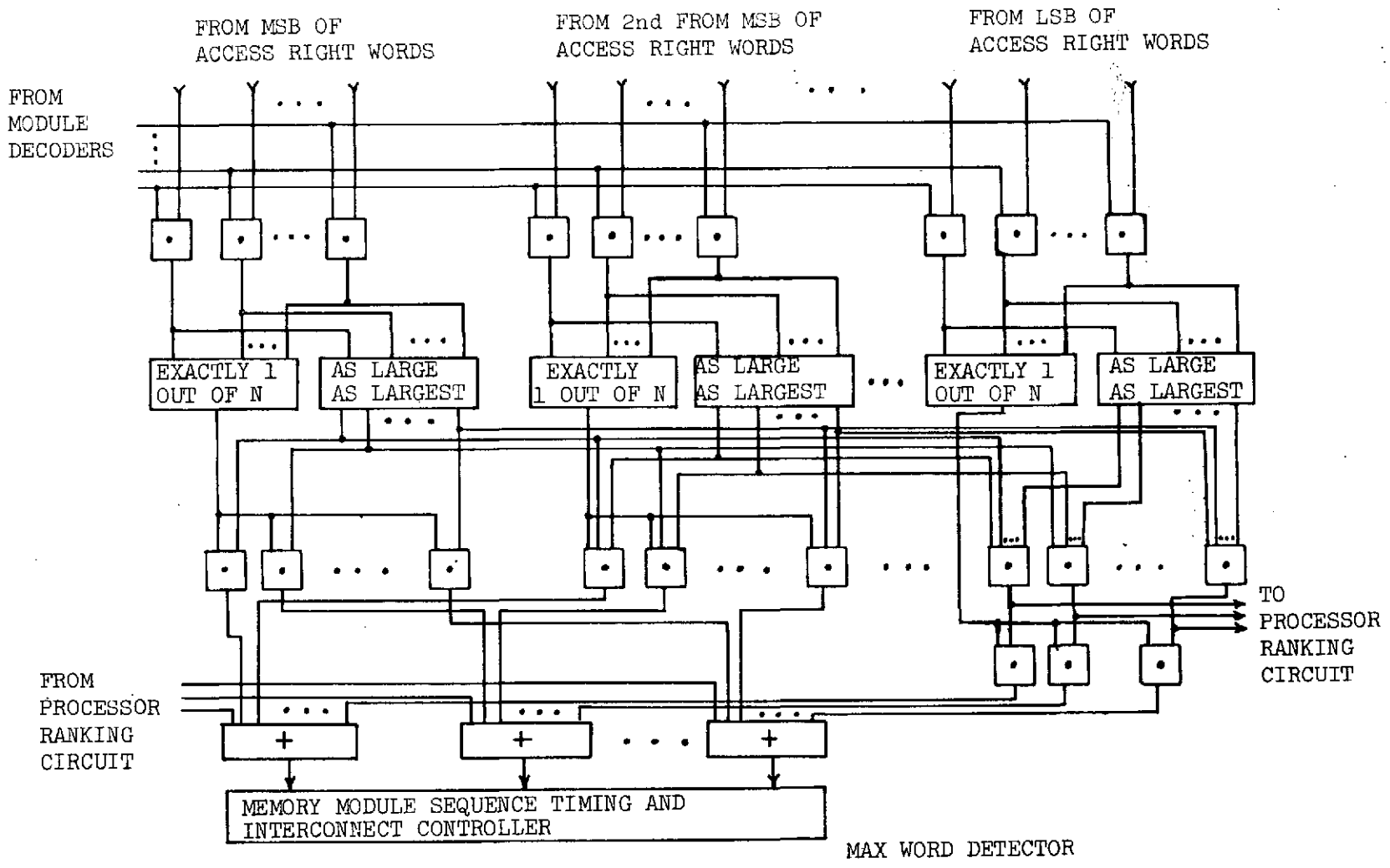


Fig. 5-26 Alternate Processor Selector Circuit

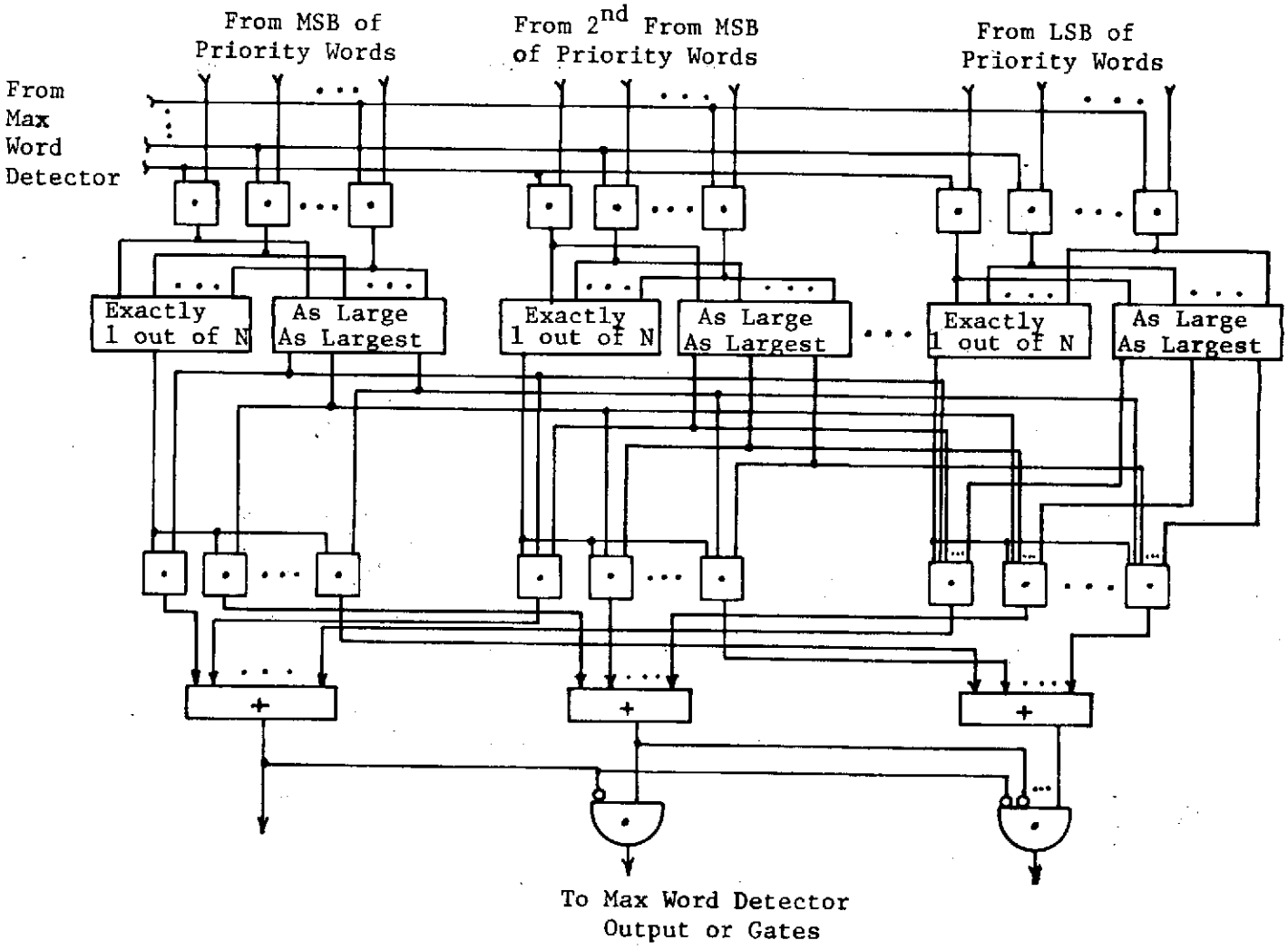


Fig. 5-27. Alternate Processor Ranking Circuit.

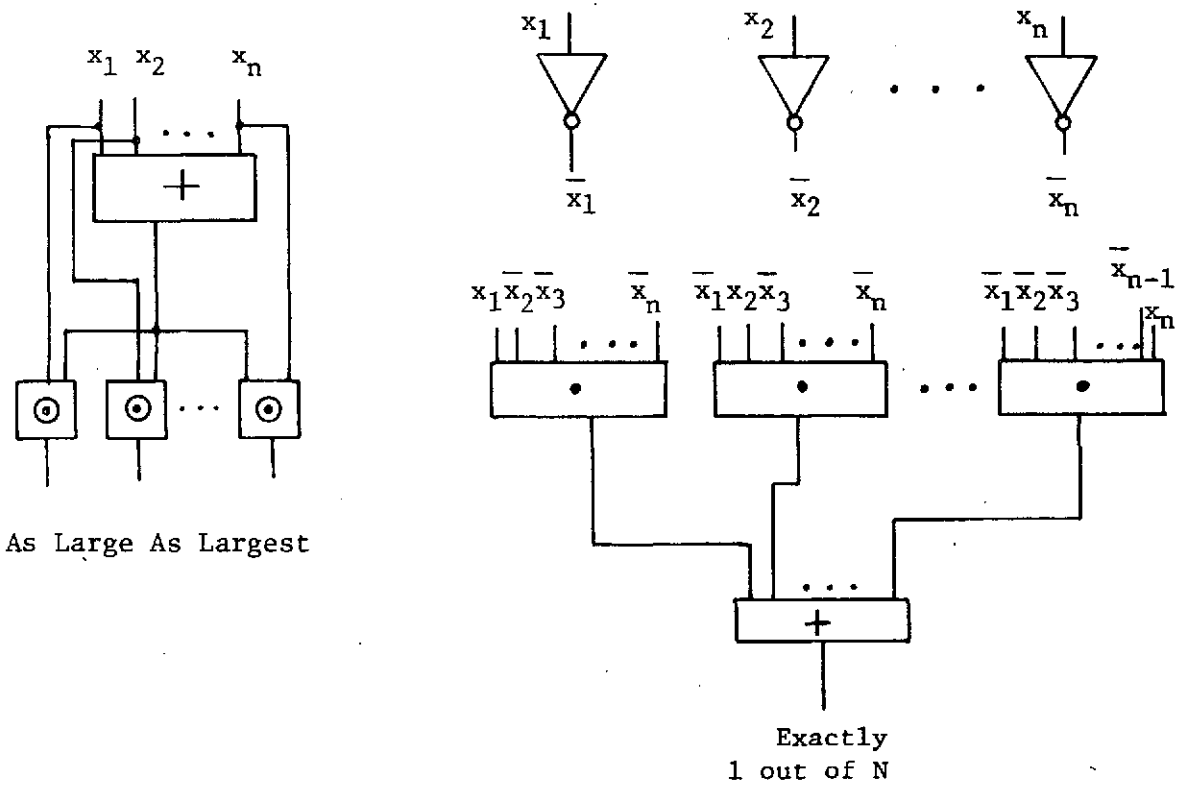


Fig. 5-28. As Large As Largest and Exactly 1 Out of N Circuits.

VI. SOME PROBLEMS FOR THE SIMULATOR

Introductory Remarks

The true value of a simulation model such as is described in this paper is not the ability to provide specific answers to specific questions but rather more often it is more useful in helping to formulate the original questions in such a manner that meaningful answers may be discovered. In other words, the simulation model helps to create an awareness of the problems and then provide the questioner with enough understanding of these problems such that answers to his questions about the problems conveys useful information to him.

The model has a large number of parameters, both explicit and implicit. Quite often, changing one of these parameters in a certain direction does influence the system in the direction that it intuitively appears that it should, but frequently the magnitude of the result is negligible in itself or of little consequence in light of a host of other mitigating factors. Thus the model provides a means for discovering which parameters are more significant under some set of assumed conditions. For these reasons during the investigations which will be conducted in this section of the paper, most of these parameters will be set to values for which past experience in working with the simulation model as it evolved indicates that they should be set in order to either highlight the effects of changes in other parameters or to remove the effects of parameters which have little real bearing on the particular

problem to be studied, but when taken along with a number of other parameters, that may be similarly classified in regards to the problem at hand, tends to dilute the effects of the ones that are being studied. This is also necessary in view of the large number of combinations of values of parameters for which simulation runs would have to be made in order to ascertain the true nature of a particular parameter of interest under the diluted conditions.

It should be apparent that the simulation model is not designed with the idea of obtaining precise results but nevertheless it is felt that useful information can be obtained about a multiprocessor computer of the form assumed in this paper through its use at a cost in time and material that is far lower than through other means.

Since we will be concerned to a considerable extent in determining system resource utilizations in this section let us define memory space, memory bandwidth and processor utilizations in terms of the simulation mode.

Memory Space Utilization - A block of memory space will be considered to be utilized 100% at a simulation step if it is assigned to an active job either in a load condition or processing condition during this step. Its utilization is considered to be zero at all simulation steps at which it is not assigned to an active job. The overall space utilization at a simulation step is then the ratio of the amount of space assigned to active jobs to the total amount of memory space. The mean space utilization over a number of simulation steps in an interval T is then the arithmetic mean of step utilizations in T.

Memory Bandwidth Utilization - Memory bandwidth utilization at a simulation step is simply the ratio of the number of accesses granted during this step to the total number available from the entire memory. The mean utilization of memory bandwidth over an interval T is the arithmetic mean of the step utilizations during this interval.

Processor Utilization - A processor will be considered to be utilized 100% at a simulation step if it is not in an idle state and it is granted as many memory accesses as it requests from the memory. Thus processor utilization is actually the ratio of the number of accesses granted to the number of accesses requested for an active processor. Idle processors are considered to have a utilization factor of zero. The overall utilization of processors of a given type at a simulation step is then the ratio of the sum of the utilization factors for all the processors of this type to the total number of processors of this type in the system. The mean utilization of processors of a particular type over an interval T is then the arithmetic mean of the step utilizations in this interval.

Although the model handles jobs on an individual basis and prints out statistics concerning service to each individual job these individual statistics will not be presented in the problems to be considered due to the bulk of material that would be required. Thus we will concentrate on the overall responses to and characteristics of the total job set.

In the simulations which were performed in this section no precedence relations were specified among the jobs in the input job sets,

each simulation run was made for 1300 steps, all jobs had an external priority of one and the following system parameters were left set as indicated:

```
ICECSZ(CESZ) = 40
ICON(CON) = 4000
IN1(N1) = 20
IFECSZ(FECZ) = 40
IPCT(PCT) = 100 (scaled internally to 1.0)
IP1(P1) = 27
IP2(P2) = 85
IQ1(Q1) = 15
IQ2(Q2) = 8
IQ3(Q3) = 3
IR(R) = 800
IS(S) = 0
ITA(TA) = 300 (scaled internally to 3.0)
MINBLK(MNBK) = 1024
MMST = 64
NI = 15
MMSPF(MSPF) = 16
MMAPF(MAPF) = 32
RP01 = 0.85
RP02 = 0.30
RP03 = 1.0
RP04 = 10.0
RP05 = 50.0
RP06 = 3.0
RP07 = 1.0
RP08 = 1.0
RP09 = 0.33333
RP11 = 0.0
```

Problems to be Studied

In this section five problems will be studied by means of the simulator. These problems include the following:

(1) Memory fragmentation

(2) System resource utilization comparisons between the feedback and nonfeedback scheduler/allocators.

(3) Aggregate system response to the input job set and frequency of schedules as a function of total memory size.

(4) Relative comparisons of system response for various system configurations and numbers of recurrent jobs.

(5) The impact of TMR jobs on the memory allocation problem.

Memory Fragmentation

In the absence of some automatic hardware mapping scheme (and an elaborate memory protection device associated therewith) for mapping a logically contiguous name space into a noncontiguous absolute address space it is highly desirable to make memory allocations to each job from a single addressably contiguous block of storage locations. Memory fragmentation is a condition whereby the available memory space becomes separated into many small noncontiguous blocks. Thus memory fragmentation can, to a certian extent, thwart the ability to make allocations from single addressably contiguous blocks, especially large allocations.

The simulation model coalesces blocks of storage as they are released with any addressably adjacent available blocks. Scheduling is then done in small batches and as much of the available memory space as possible is allocated to jobs in the schedule that require memory. This method of operation has the advantage of choice for optimization over a scheme that reallocates the memory space as soon as it becomes available or as soon as there is a demand for it. If this latter option of immediate reallocation were adhered to then the system would try to become a first-in first-served device. While first-in first-served devices are esthetically appealing for queues involving people, they do not lend themselves to optimization in the utilization of resources nor do they necessarily ensure fastest service to individual jobs in a computer enviroment and can be atrocious in their appetite for system overhead. For example, under a first-in first-served policy (policy but not in

fact realizable) the jobs in the system queue that are waiting for memory space would tend to be from the population requiring the larger amounts of space. Concomitantly the jobs releasing space would tend to be from the population requiring the smaller amounts of space. Thus, quite often, following release of space the system would search through the entire list of jobs that were waiting for memory without finding one that would fit into the released space. Therefore all three of the evils mentioned above are realized.

It has been suggested [28] that the memory fragmentation problem is tolerable under conditions similar to those described here for the simulation model, however, no quantative figures are available. Therefore the simulator will be used to ascertain the amount of memory fragmentation. Since in this regard we are more interested in the average utilization of memory space when there is great demand for it let us adjust the input demands on the system's memory space such that it remains high at all times and observe the total amount of available storage space immediately before and after each new schedule. This may be done for several different average request sizes and dispersions relative to that of the total memory space. Actually a number of cases were observed concerning the fragmentation problem but the following two cases are typical.

Case 1

- Total number of recurrent jobs in the system = 40
- Each job's memory space size from a uniform random distribution between 4 and 32
- Actual average request size = 21.7

The data shown in Fig. 6-1 was then obtained from five simulation runs with total space of 128, 192, 256, 320, and 384. A plot of this data is shown in Fig. 6-2.

Case 2

- Total number of recurrent jobs in the system = 40.
- Total memory size fixed at 256
- Job space size varying from job to job

The data shown in Fig. 6-3 was then obtained from four simulation runs with job space size varying in the ranges of 12-24, 8-32, 4-64 and 1-128. A plot of this data is shown in Fig. 6-4.

The data of Fig. 6-1 and the curve of Fig 6-2 indicates that the portion of total space available immediately after schedules was almost constant and less than .05 for all five of the total memory sizes. From the data of Fig. 6-3 and the curve of Fig. 6-4 this same parameter varied in the range from approximately .04 to .12. The larger portion of total space available in the second case may be only partially attributed to the relative demand for space. This demand is roughly measured by the product of the average space requirement per job and the average number of jobs waiting memory before each schedule. However, the system's ability to make maximum utilization of the total memory space is also controlled by the relationships between the space requirements of the jobs and the sizes of the available blocks of space as well as whether or not the system is biased in favor of the smaller size jobs. From the data of Fig. 6-3 it appears that the system was biased in favor of the smaller size jobs, especially in the case of the largest requirements range.

	Total memory space				
	128	192	256	320	382
No. of schedules	30	19	16	13	10
Av. no. of available blocks before schedule	2.83	3.58	4.56	4.92	5.60
Av. total available space before schedule	52.77	101.10	139.12	180.54	224.2
Av. no. of jobs waiting memory before schedule	32.40	31.53	30.75	28.77	28.80
Av. no. of available blocks after schedule	2.55	2.94	3.38	3.70	4.10
Av. total space after schedule	5.22	6.68	7.31	13.15	13.30
Av. no. of jobs waiting memory after schedule	29.73	26.21	23.94	20.38	18.4

Fig. 6-1 Tabulation of data concerning memory fragmentation--Job space requirements from uniform distribution between 4 and 24 with mean = 21.7, varying total space.

A Portion of total jobs waiting memory before schedule
B Portion of total jobs waiting memory after schedule
C Portion of total space available before schedule
D Portion of total space available after schedule

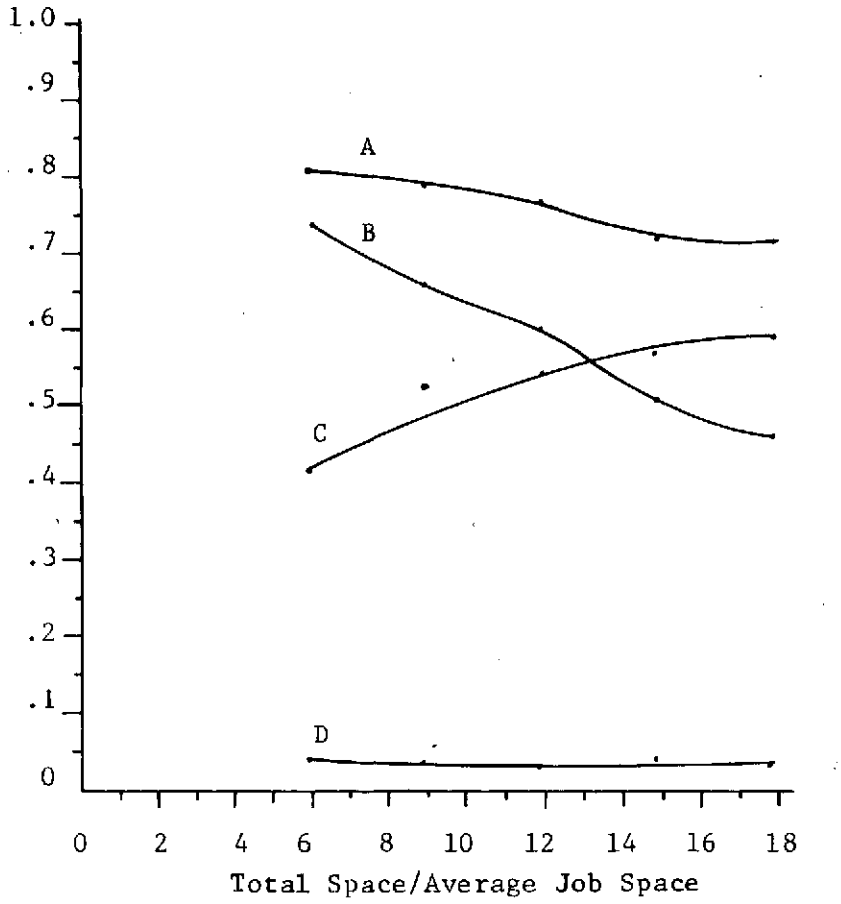


Fig. 6-2 A plot of data concerned with memory fragmentation - Constant job space requirements, varying total space.

	MEMORY SPACE REQUIREMENTS RANGE			
	12-24	8-32	4-64	1-128
No. of schedules	69	74	64	28
Av. space requirement	19.33	22.15	38.75	80.38
Av. No. of available blocks before schedule	4.0	3.57	2.83	1.79
Av. total available space before schedule	115.80	100.74	95.17	116.43
Av. No. of jobs waiting memory before schedule	5.84	8.53	21.44	31.57
Av. No. of available blocks after schedule	3.55	3.23	2.47	1.46
Av. total space after schedule	26.60	30.25	24.60	9.68
Av. No. of jobs waiting memory after schedule	1.87	5.30	19.44	29.50

Fig. 6-3 Tabulation of data concerning memory fragmentation for four space requirement ranges, total memory size = 256, 40 jobs

A - Portion of jobs waiting memory before each new schedule.
B - Portion of jobs waiting memory after each new schedule.
C - Portion of total space available before new schedule.
D - Portion ot total space available after new schedule.

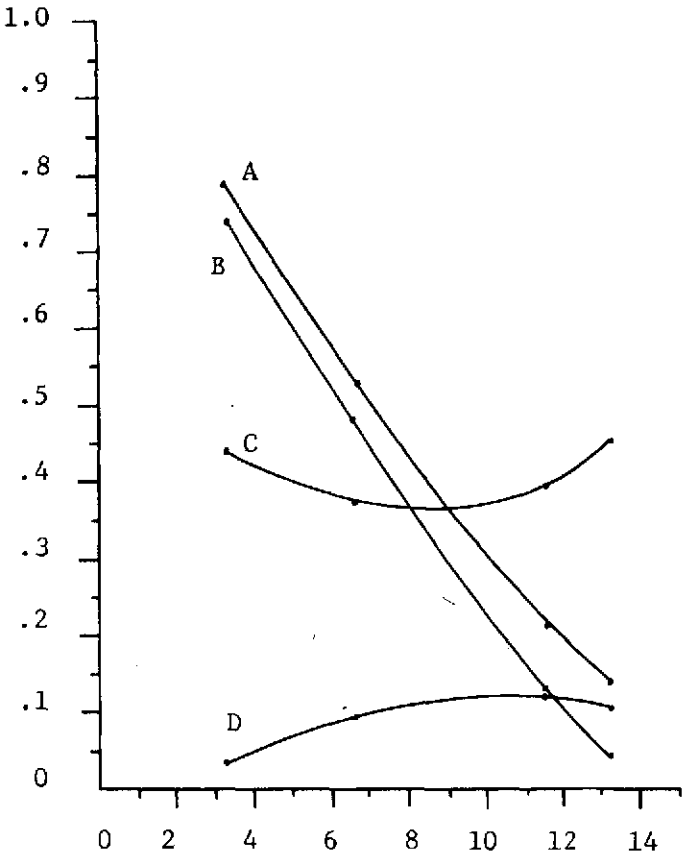


Fig. 6-4 A plot of data concerned with memory fragmentation - Varying job space requirements, constant total space.

System Resource Utilization Comparisons Between the Feedback and Non-feedback Scheduler/Allocators

As mentioned before in sections I and II the system may be configured such that memory allocation and internal priorities of each job are influenced by goodness of fit in both space and bandwidth between requirements of jobs and an estimate of what is available from the main memory at the time of memory allocation. In making these comparisons a number of simplifications were made but the most notable ones were that the numbers of input/output operations for each job were adjusted such that these operations had a small influence on the overall system and that the memory space for each job was entirely within a single memory module. Four simulation runs each of 1300 simulation steps were then made with the system configurations and job characteristics listed in Fig. 6-5. The results of these runs are shown in Fig. 6-6. These results warrant some discussion. First of all, it should be pointed out that the nonfeedback configuration tends to be biased against jobs with large bandwidth requirements since if priorities of a large bandwidth job and a small bandwidth job are equal and assuming large overall bandwidth demands then the program executer will tend to grant memory accesses to the low bandwidth job in accordance with the number that it requests. Conversely, it will tend to grant memory accesses to the high bandwidth job in accordance with the priority of the high bandwidth job relative to that of other active jobs having space in the same memory module. Thus

	Run Number			
	1	2	3	4
Number of CPUs	10	10	9	8
CPU Speed	1	1	1	1
Number of IOPs	12	12	12	12
IOP Speed	5	5	5	5
Number of Memory Modules	10	5	3	2
Memory Module Speed	16	16	16	8
Total Number of recurrent jobs	40	40	40	40
Number of job sets	1	2	3	4
Job bandwidth requirements	B_m	$.1xB_m$, $.9xB_m$	$.1xB_m$, $.3xB_m$, $.6xB_m$	$.1xB_m$, $3/20xB_m$, $.3xB_m$, $9/20xB_m$
Job space requirements	S_m	$1/2xS_m$	$1/3S_m$	$1/4xS_m$

All speeds are in numbers of basic clock periods per cycle time of device.

Fig. 6-5 System configurations and job characteristics for comparisons between feedback and nonfeedback scheduler/allocators.

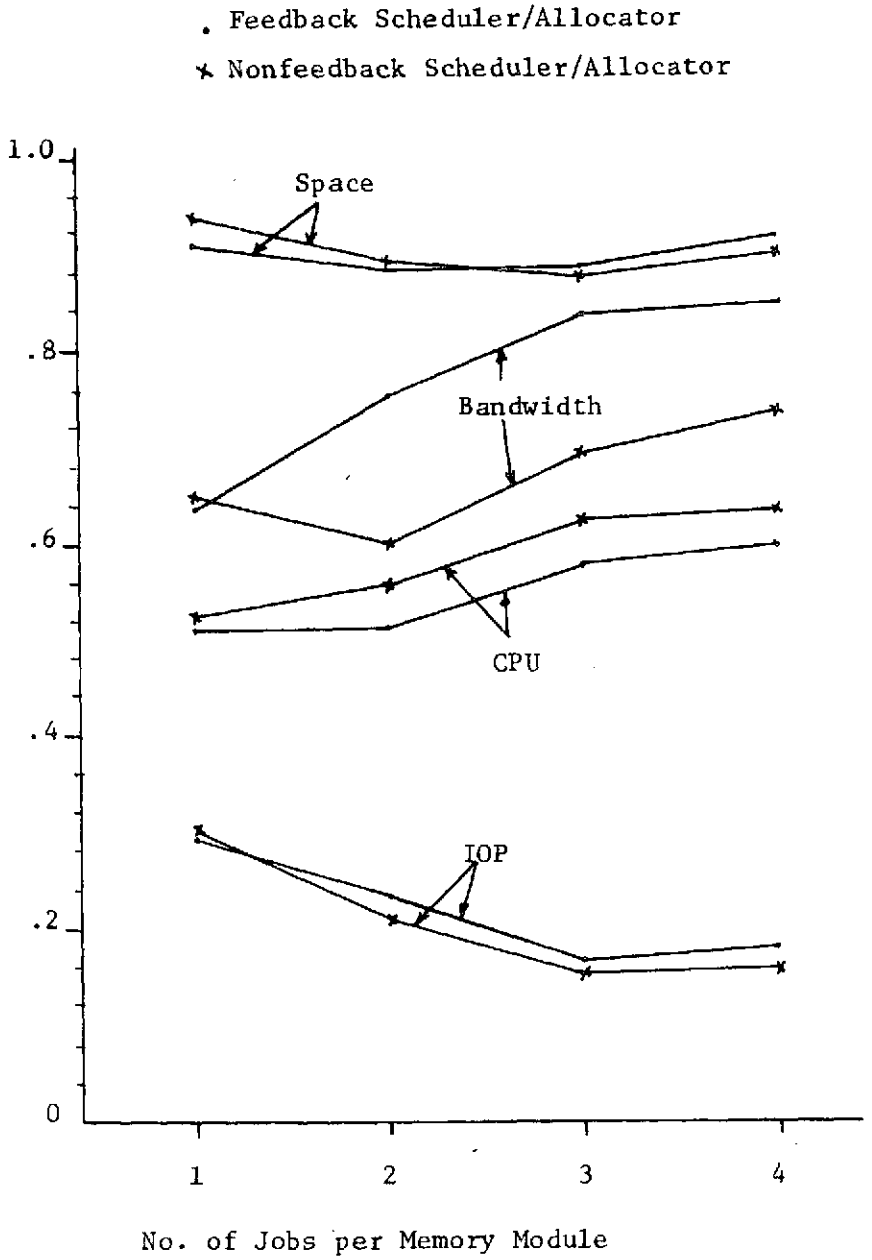


Fig. 6-6 System resource utilization comparisons between feedback and nonfeedback scheduler/allocators.

with a fixed direct relationship between bandwidth requirements and total number of accesses for job completion the low bandwidth jobs will tend to run in a shorter length of time than the high bandwidth jobs. With approximately equal populations of high and low bandwidth jobs the net result is that the low bandwidth jobs are run with higher frequency than the high bandwidth jobs, especially with large total demand on the system. If a CPU is not idle during some period of time T and is granted all the memory accesses that it requests during this interval T then we must assume that its utilization is 100% during this interval. This explains the higher utilization of CPUs during all four runs for the nonfeedback configuration. The memory bandwidth utilization was higher for the feedback configuration in all runs except run #1. In run #1 there was no real choice for bandwidth matching since all the jobs had the same space and bandwidth requirements. The number of executions for each job was not the same for the feedback and nonfeedback configurations and since the jobs did not have the same processing shape curves a small difference in bandwidth utilization can be expected during this run.

Memory bandwidth utilization is a rough measure of the total processing done by the system if relative service to individual jobs can be ignored and it is assumed that each program executed by the system has been optimized for the particular system configuration. Alternately, inefficiencies inherent in particular programs need not be considered in relative comparisons between the feedback and nonfeedback versions of the scheduler/allocator since these could theoretically be removed in either case. Under these assumptions the feedback configuration appears to have performed more work than the nonfeedback version.

Aggregate System Response to the Input Job Set and Frequency of Schedule as a Function of Total Memory Size

The total space in the main memory appears to have a considerable influence on the response of the system to individual jobs as well as the amount of overhead required to keep the system utilization at reasonably high levels. The frequency at which new schedules must be initiated in order to attempt to maintain some level of resource utilization is a fairly good relative indicator of overhead associated with scheduling jobs and allocating memory space. For the present problem three groups of four sets of four simulation runs were made corresponding to a total space that was roughly .1, .2, .4 and .8 times that required by all the recurrent jobs in the system. The system's total memory bandwidth was held constant at approximately .52 times the amount that would be demanded if all jobs were executed at their desired repetition rates. In this manner the total demands on the system were maintained at a high level throughout each run. In each set of runs four runs were made corresponding to 2, 4, 8, and 16 memory modules. Total memory speed and space was held constant in each of the four runs of a set by adjustment of module size and speed. This resulted in module speed being directly related to size. This is of course contrary to what one would expect in regard to maximum capability from real memory modules all of the same technology but of different sizes. However, we may consider the largest and fastest memory modules to be the highest quality and consequently the highest cost. Then by substituting larger numbers of smaller and slower modules we may observe the changes in the system's performance.

If at each simulation step each job's memory access requests are distributed over its total memory space in a linear manner then in those instances in which the mean job size is larger than the space of a memory module the results are not valid. This is true because real computer programs are far from random in regard to the sequence in which accesses are made to locations of their memory space. In fact, the success of virtual memory systems is predicated on the 'locality of reference' [25] phenomenon of real computer programs. Thus if the contingencies mentioned above actually held then the speed at which a processor could obtain accesses from a single memory module would be an upper bound on the speed of execution of a particular program. This problem could be overcome somewhat by interleaving the memory to a depth such that the space spanned by each interleaved set of modules is greater than the mean job space requirement. In the present problem for those cases in which the mean job size is actually larger than module size runs will be made for both random and sequential distributions of access requests so that comparisons may be made.

For the first group of four sets of four runs each, the bandwidth requirements were matched to the amount of bandwidth available from the memory space occupied by the average size job. By holding total available memory bandwidth fixed and increasing total memory space, the amount of bandwidth that is available to a job of a fixed size tends to decrease directly with total memory size. Thus by adjusting the bandwidth requirements of each job inversely in relation to total memory size the amount of bandwidth that is available to a job of the mean size should remain fairly well matched over all total memory sizes. In this manner we may

remove the effects of bandwidth availability to individual jobs on CPU utilization. Thus CPU utilization becomes an adjusted indicator of system thruput and we are able to observe the effects of space alone on this indicator. With more space more jobs may be in the memory at the same time and CPU utilization should increase as space increases to some point at which the number of jobs in the memory is somewhat larger than the number of CPUs. With a total memory size of sixty-four and a mean job size of sixteen the system may on the average have four jobs either being loaded or in execution. At a total memory size of 512 this average number is increased to thirty-two. However with the number of CPUs fixed at eight and the number of IOP's at sixteen we should expect to see CPU utilization to range from something less than one-half to close to unity depending upon how high the system tries to maintain CPU utilization through rescheduling.

For this problem we will assume that the total IOP load is light at all times. The result of all this is that the factor determining sytem thruput, as reflected in CPU utilization, should pass from a low value at the smaller memory size in which the CPUs can be utilized only lightly due to an insufficient number of jobs in the memory to an asymptotic high value in which more jobs are in the memory than the fixed number of CPUs can adequately service. The system configurations and job characteristics for the present problem are shown in Fig. 6-7. The results for the first group of runs in which job bandwidth requirements were matched to that available from the space occupied by the mean size job is shown in Fig. 6-8 through Fig. 6-13.

For the second group of runs the bandwidth requirements were held fixed at the value that was used for the smallest memory size in the first group of runs. The results for this group of runs are shown in Fig. 6-14 through Fig. 6-18. In the third group of runs the bandwidth requirements were allowed to vary from job to job over the range from .06 times the memory's total bandwidth to .25 times the memory's total bandwidth. The results for this group of runs is shown in Fig. 6-19 through Fig. 6-24. In Fig. 6-8 the CPU utilization curve has the expected shape. It indicates that a total memory size in the range of 256 is adequate for the assumed number of CPUs and mean job sizes. Comparisons between Fig. 6-9 and Fig. 6-15 show that the lowering of bandwidth requirements directly with total memory size had a considerable influence on the steepness of the slope of the bandwidth utilization curves of Fig. 6-9 but had little effect on the space utilization curves. The downward slope of CPU, memory bandwidth and job completions between memory sizes of 256 and 512 in Fig. 6-14, Fig. 6-15, and Fig. 6-16 was caused by localized bandwidth limiting due to packing of a disportionate number of jobs into one end of the memory. The effects of this phenomenon are greater for larger numbers of smaller memory modules as is indicated by these curves. The relatively large amount of space available immediately after schedules in Fig. 6-17 for the largest total memory size is due to the smallness of space demand relative to the amount that is available and is not a true indicator of the fragmentation problem. Sequential distribution of access requests, as indicated by the dashed lines, resulted in general in lower utilization of system resources.

Total Memory Size		(Number of Modules, Module Speed) (2,8)	(4,16)	(8,32)	(16,64)
512	A MIX	1/16 .568	1/8 .568	1/4 .568	1/2 .568
256	A MIX	1/8 .785	1/4 .785	1/2 .785	1 .785
128	A MIX	1/4 .915	1/2 .915	1 .915	2 .915
64	A MIX	1/2 .980	1 .980	2 .980	4 .980

Module speed is in basic clock periods per module cycle period.

A is the bandwidth requirement of each job expressed in units of the bandwidth of a memory module, B_m.

MIX is the fraction of short instructions for each job. In these runs a short instruction requires eight CPU cycle periods and a long instruction requires 256 CPU cycle periods.

Number of CPUs = 8, Number of IOPs = 16
Number of recurrent jobs = 40
Job space range = 4-24, Mean = 16.3
Desired mean job interarrival period = 275
Mean number of memory accesses per execution: CPU = 7500
IOP = 2750.

Mean job bandwidth requirement:

Group #1 - $\frac{\text{(Mean space requirement)}}{\text{Total space (see above listing)}} \times$ Total Memory bandwidth

Group #2 - .25 x total memory bandwidth

Group #3 - varying over range .06 to .25 times total memory bandwidth

Fig. 6-7 System response as a function of total memory size - System configurations and job characteristics.

A α E 2 memory modules
B α F 4 memory modules
C α G 8 memory modules
D α H 16 memory modules

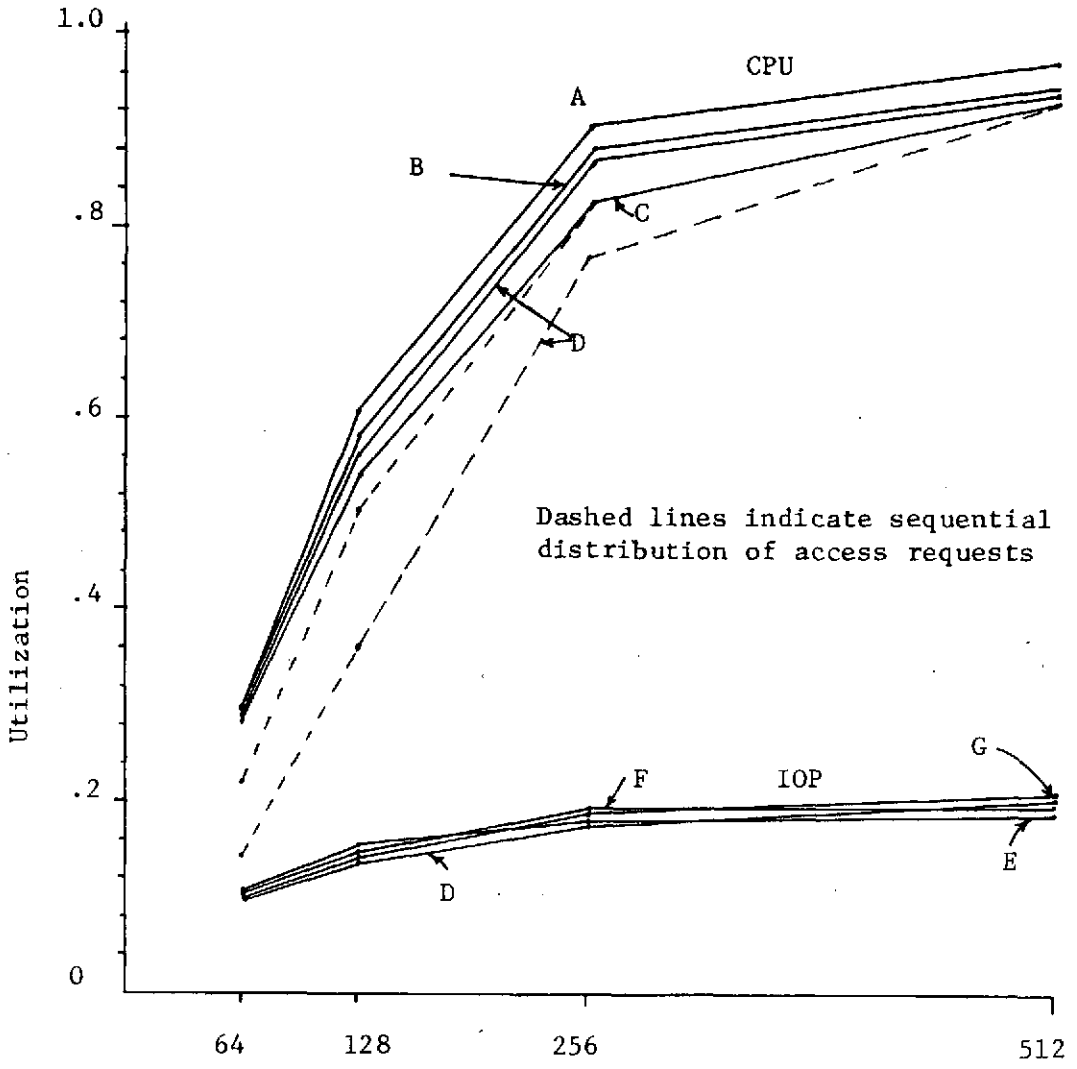


Fig. 6-8 System response as a function of total memory size with matched bandwidth requirements - CPU and IOP utilizations.

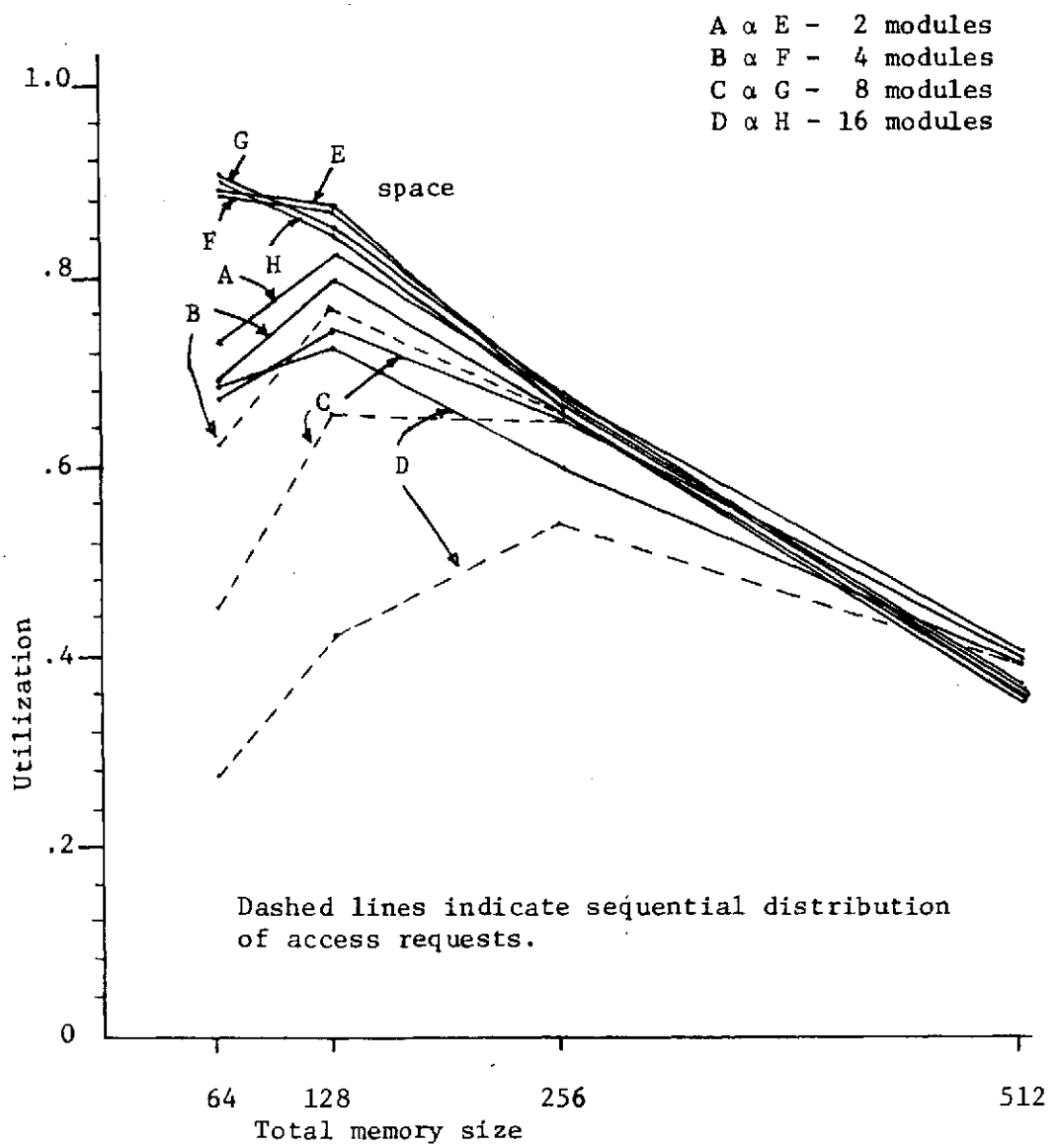


Fig. 6-9 System response as a function of total memory size with matched bandwidth requirements - Memory space and bandwidth utilizations.

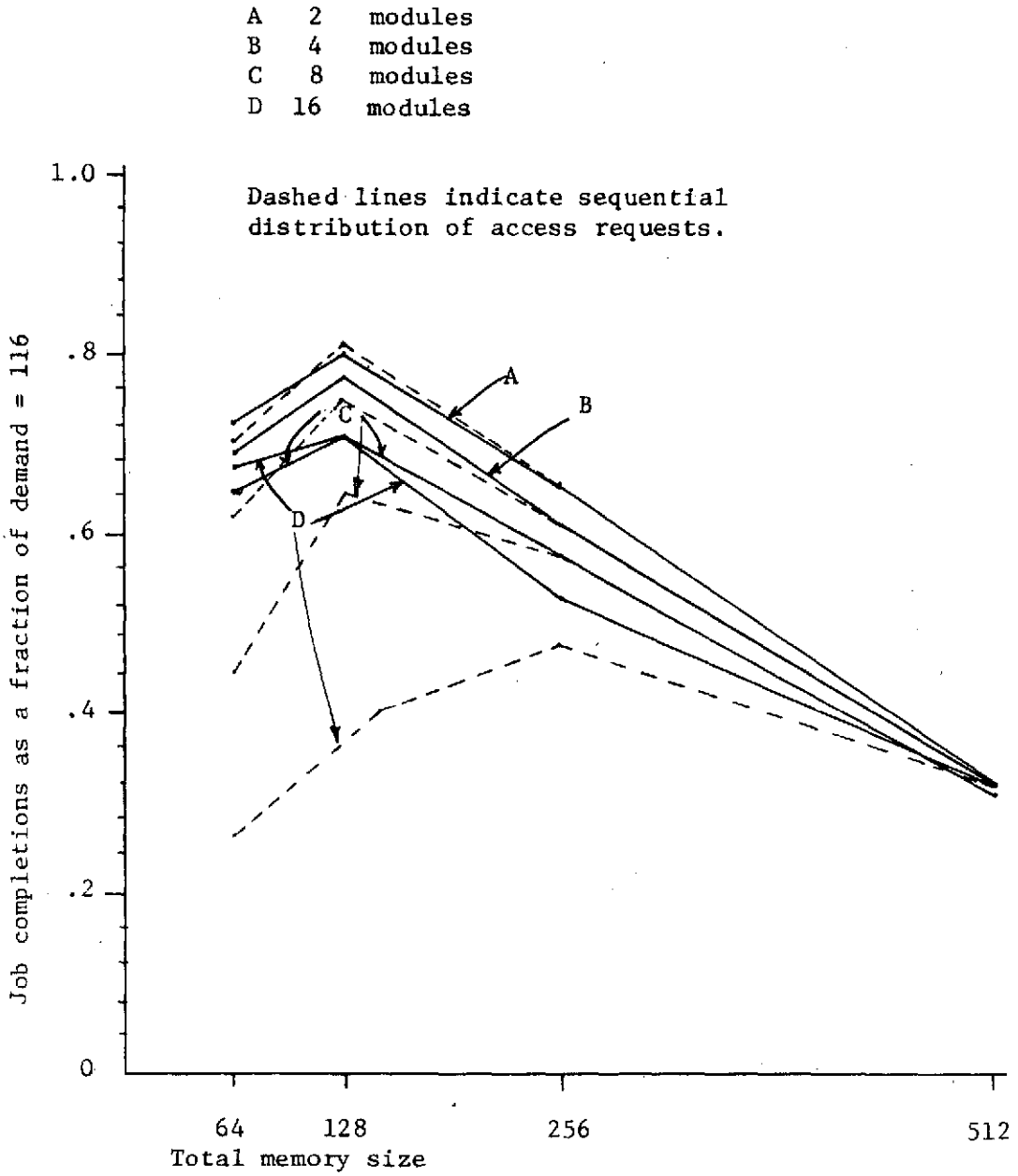


Fig. 6-10 System response as a function of total memory size with matched bandwidth requirements - Job completions.

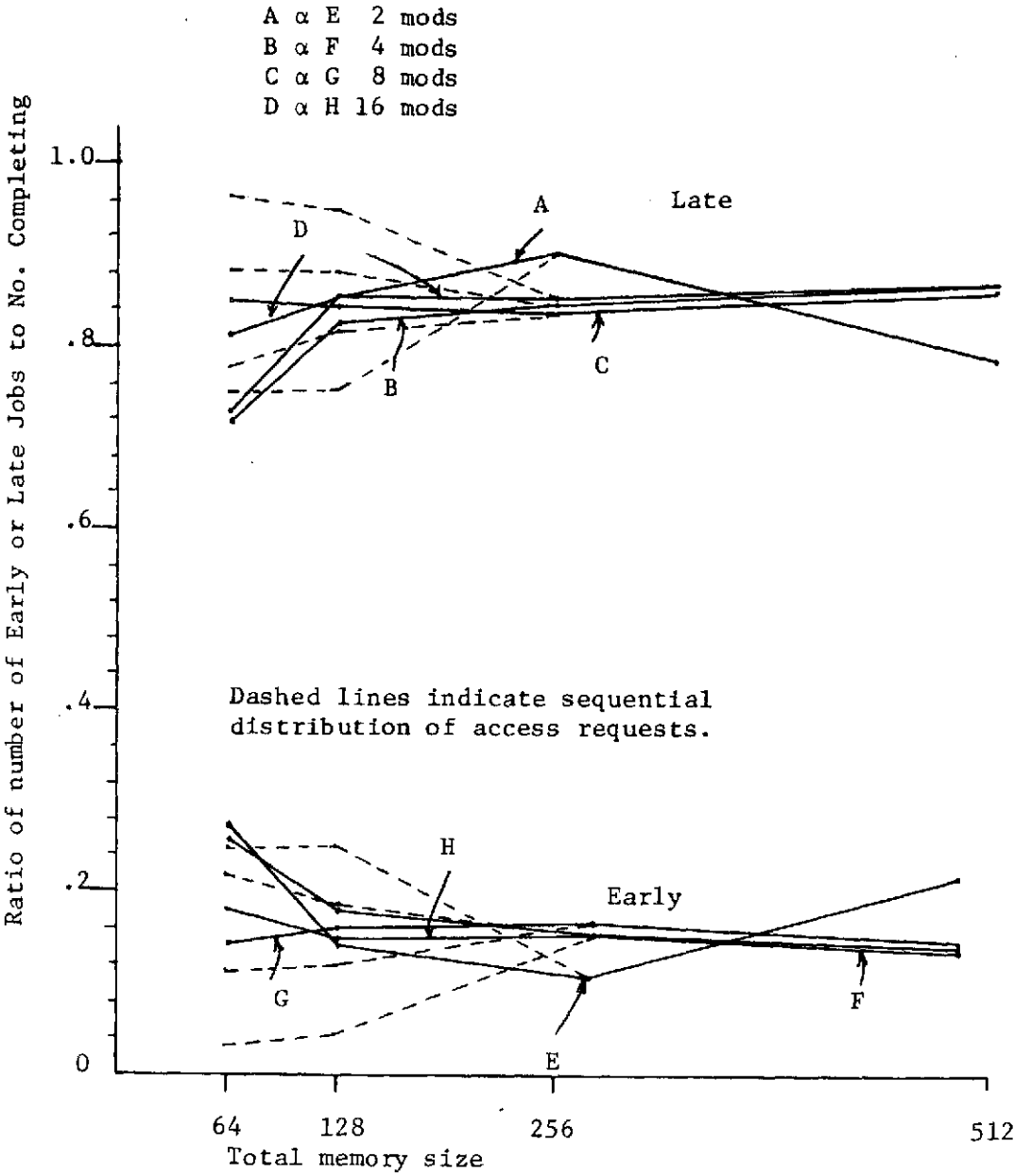


Fig. 6-11 System response as a function of total memory size with matched bandwidth requirements - Late and early jobs.

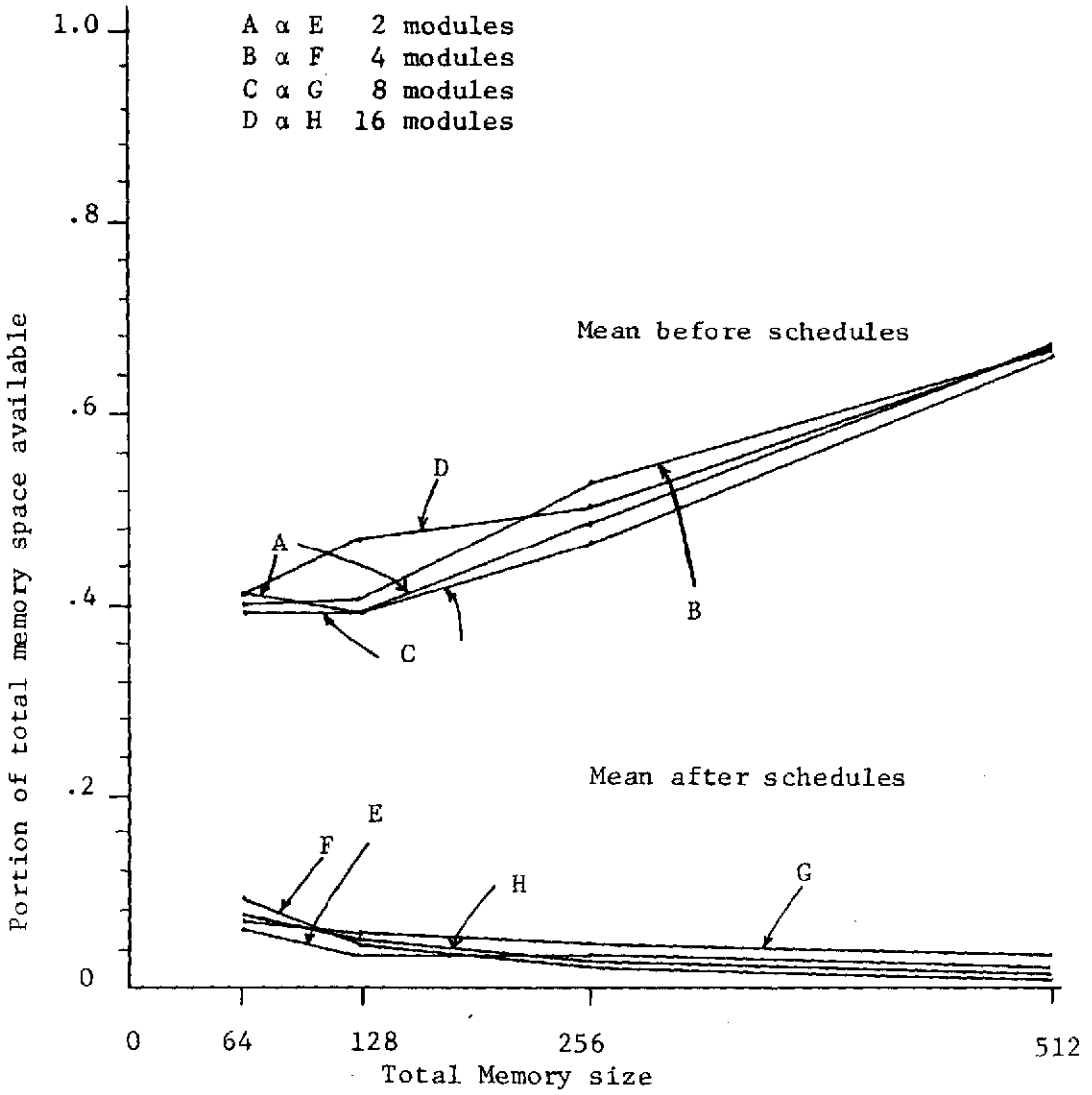


Fig. 6-12 System response as a function of total memory size with matched bandwidth requirements - Available space before and after schedules.

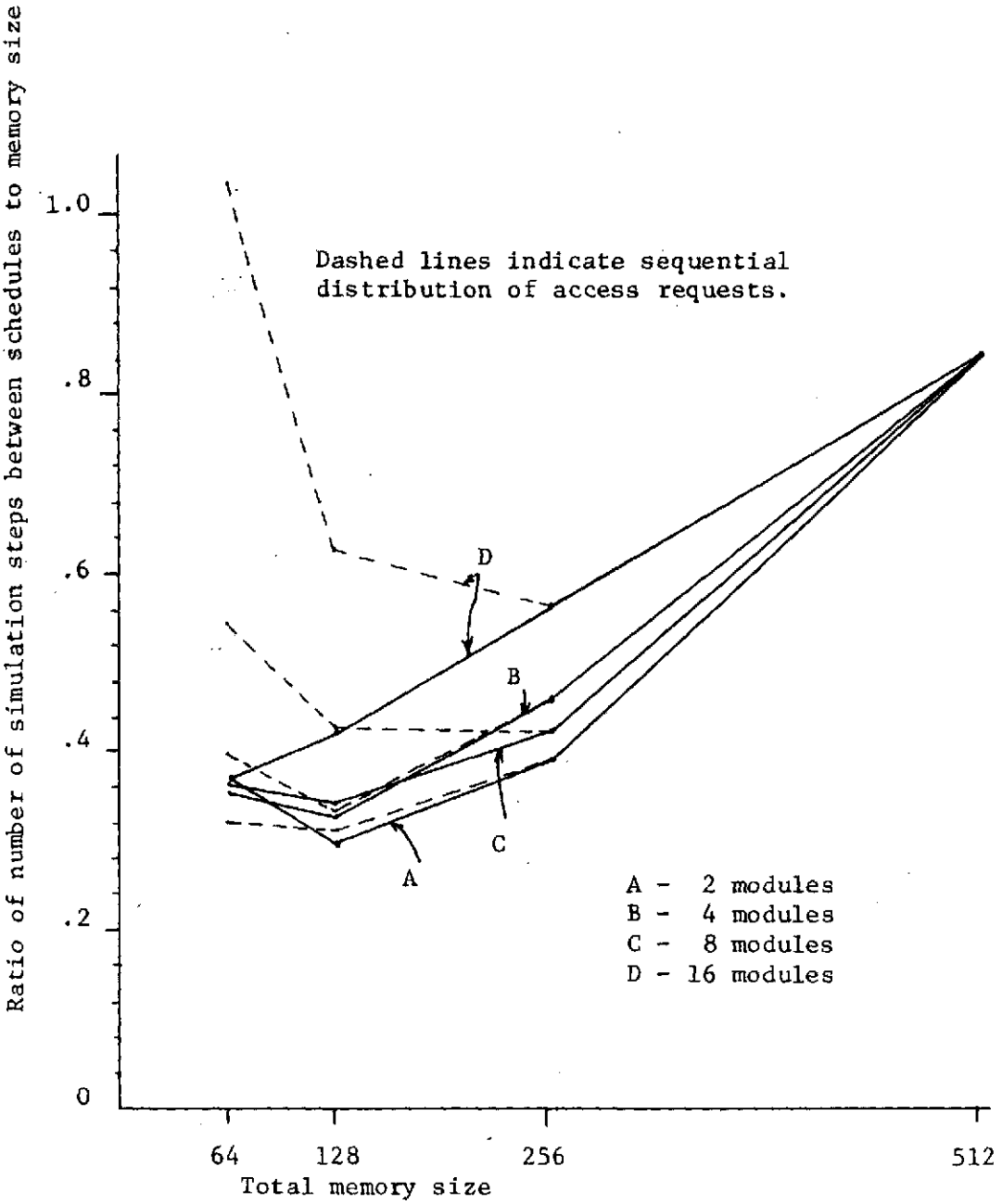


Fig. 6-13 System response as a function of total memory size with matched bandwidth requirements - Frequency of schedules.

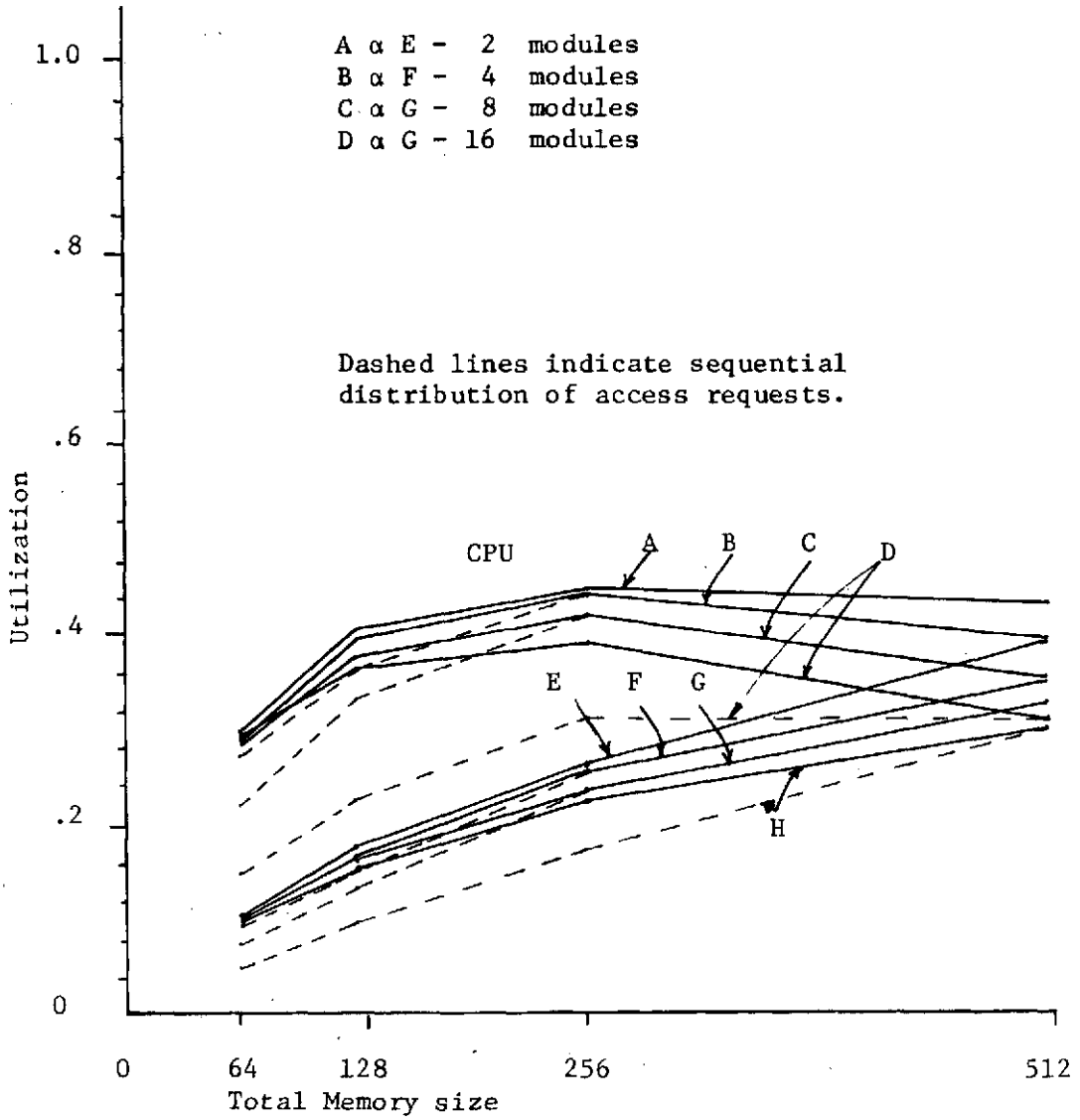


Fig. 6-14 System response as a function of total memory size with constant bandwidth requirements - CPU and IOP utilizations.

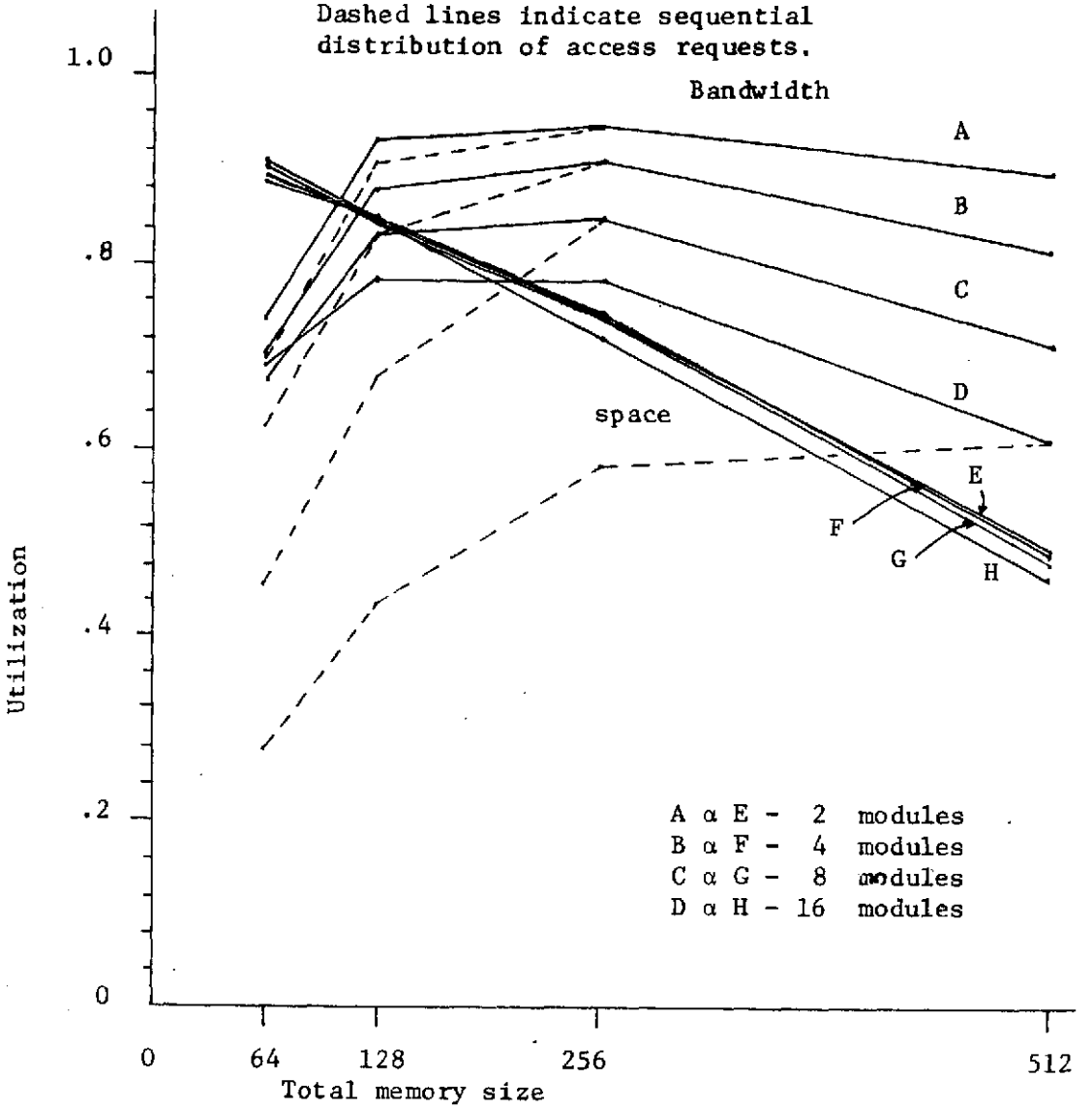


Fig. 6-15 System response as a function of total memory size with constant bandwidth requirements - Memory space and bandwidth utilizations.

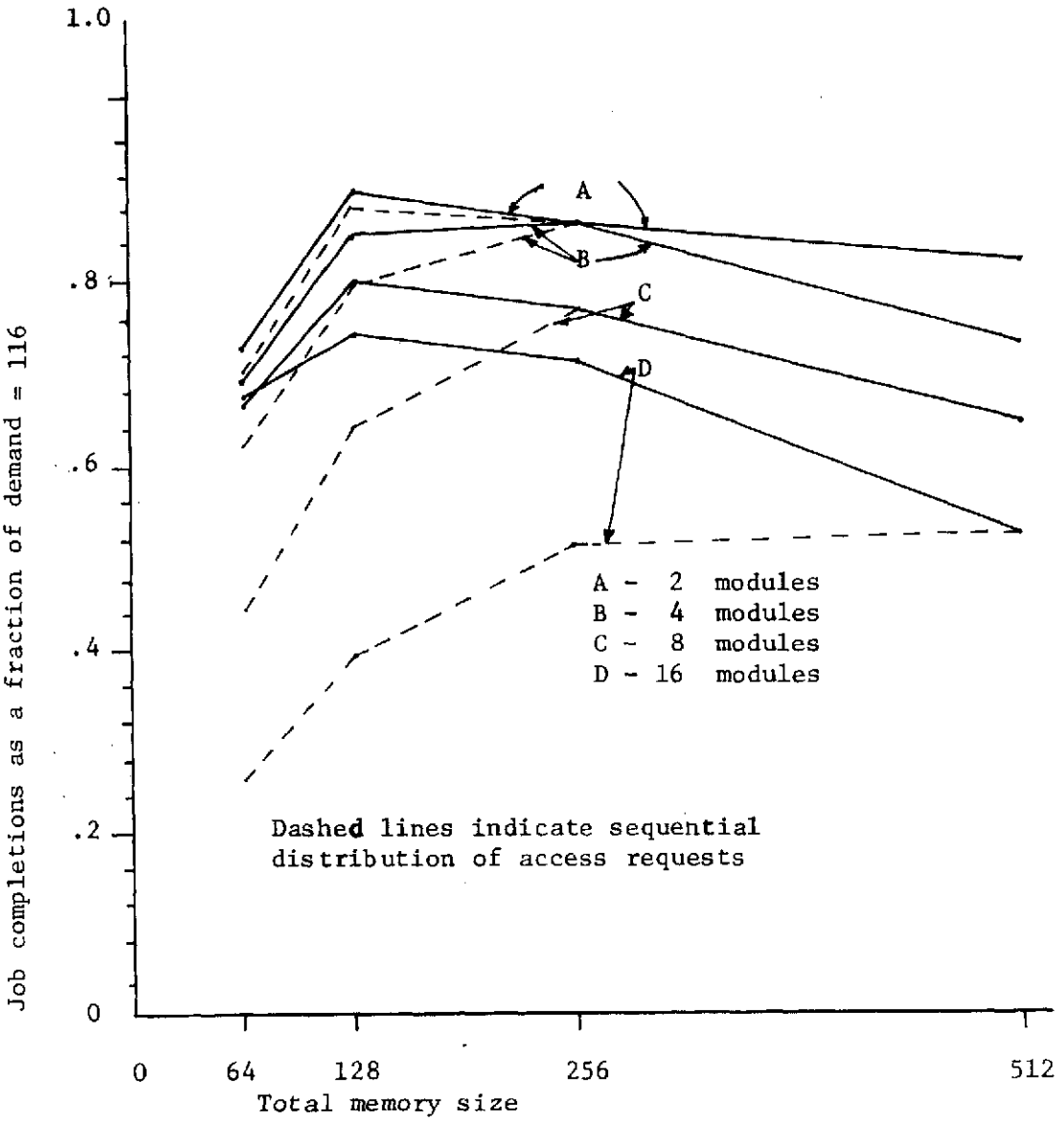


Fig. 6-16 System response as a function of total memory size with constant bandwidth requirements - Job completions.

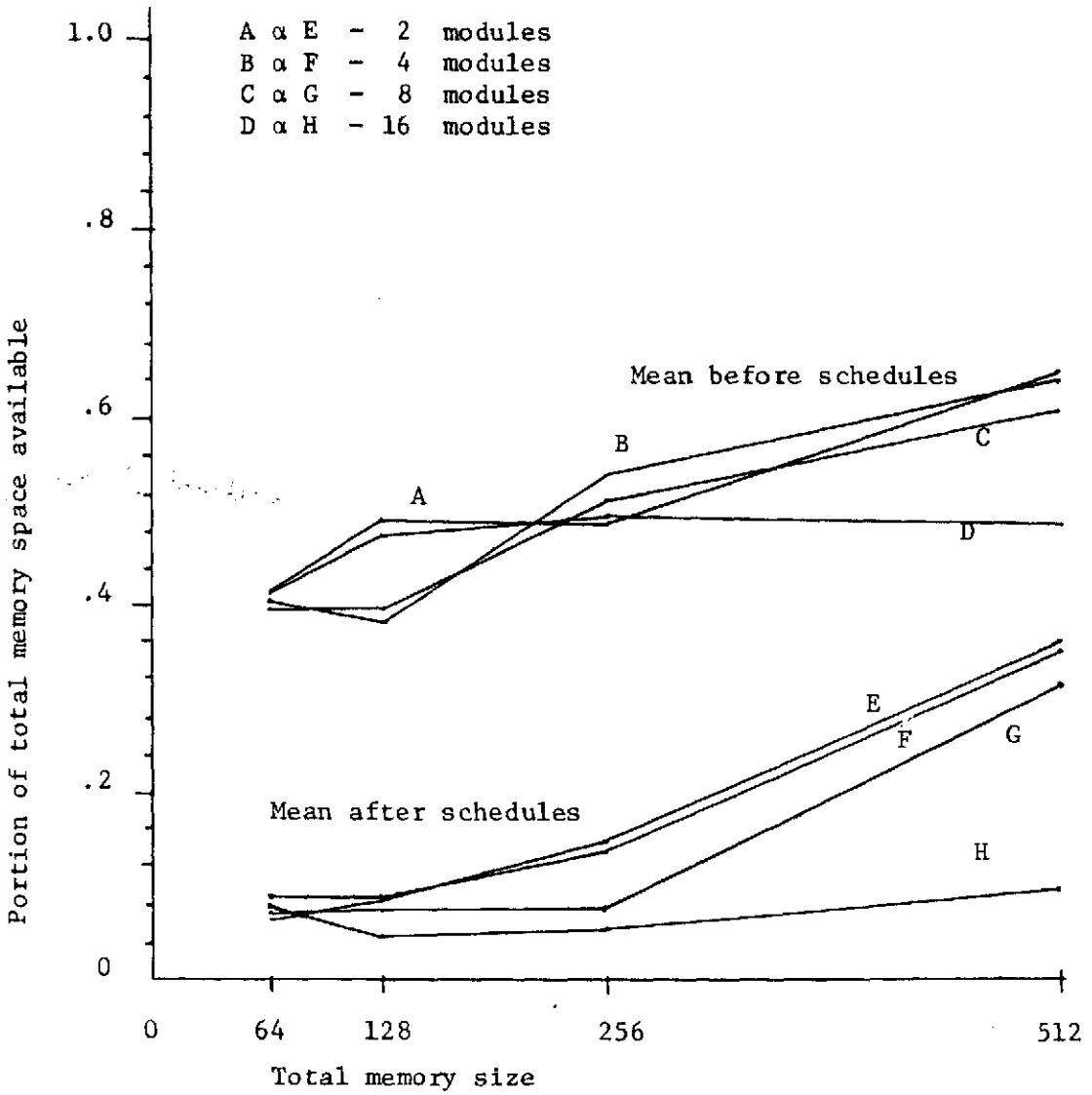


Fig. 6-17 System response as a function of total memory size with constant bandwidth requirements - Available space before and after schedules.

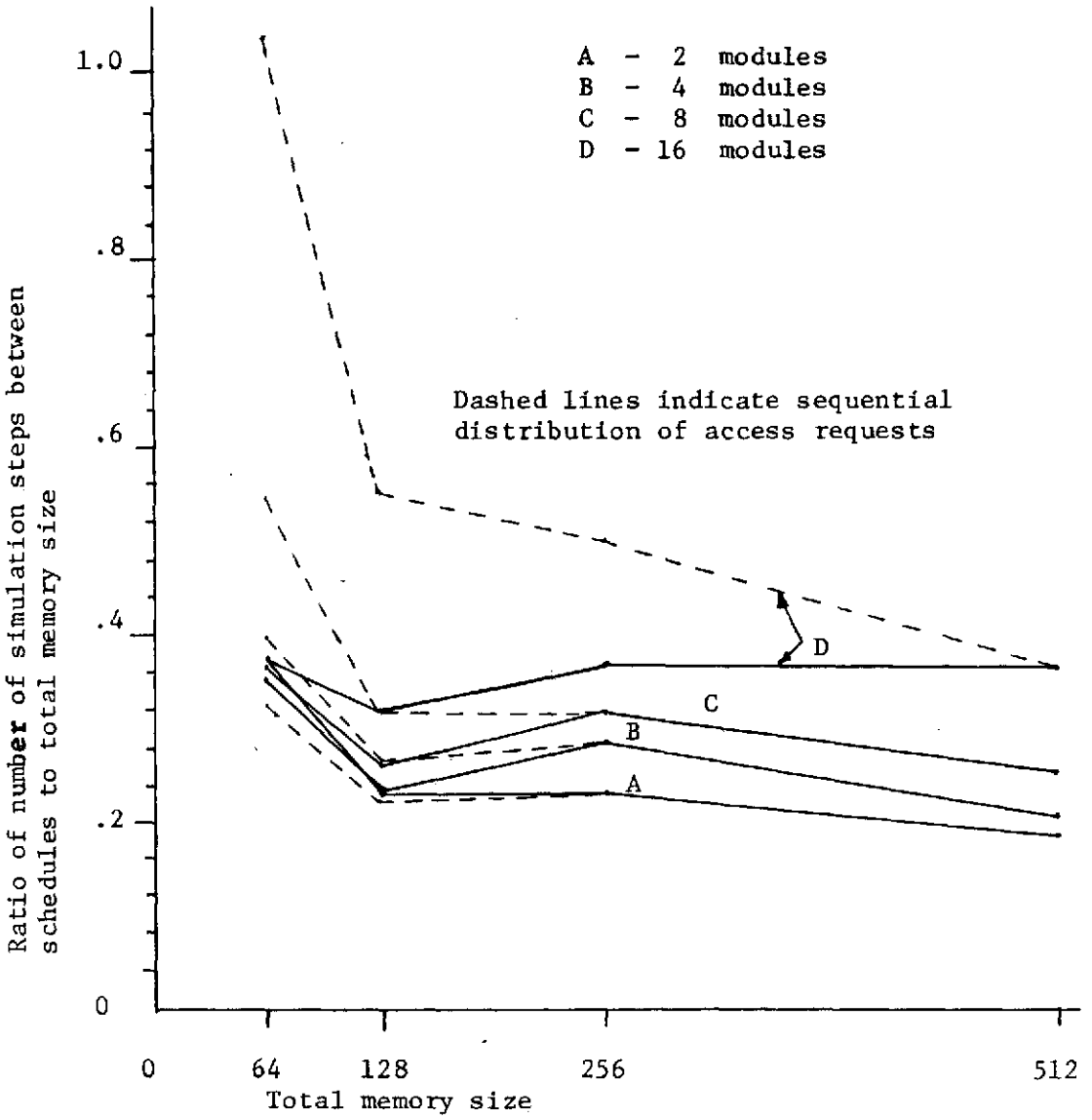


Fig. 6-18 System response as a function of total memory size with constant bandwidth requirements - Frequency of schedules.

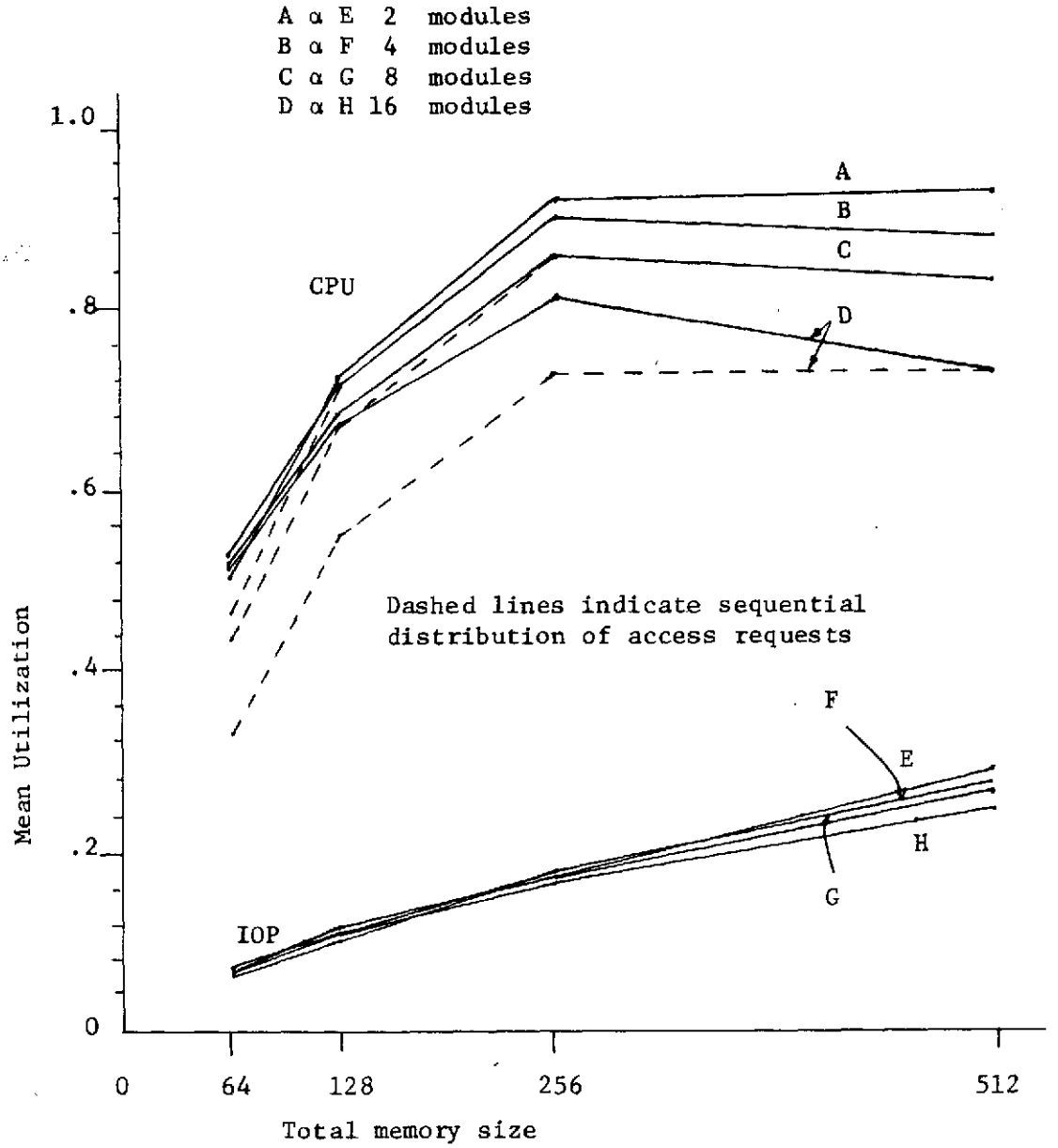


Fig. 6-19 System response as a function of total memory size with widely dispersed bandwidth requirements-CPU and IOP utilizations.

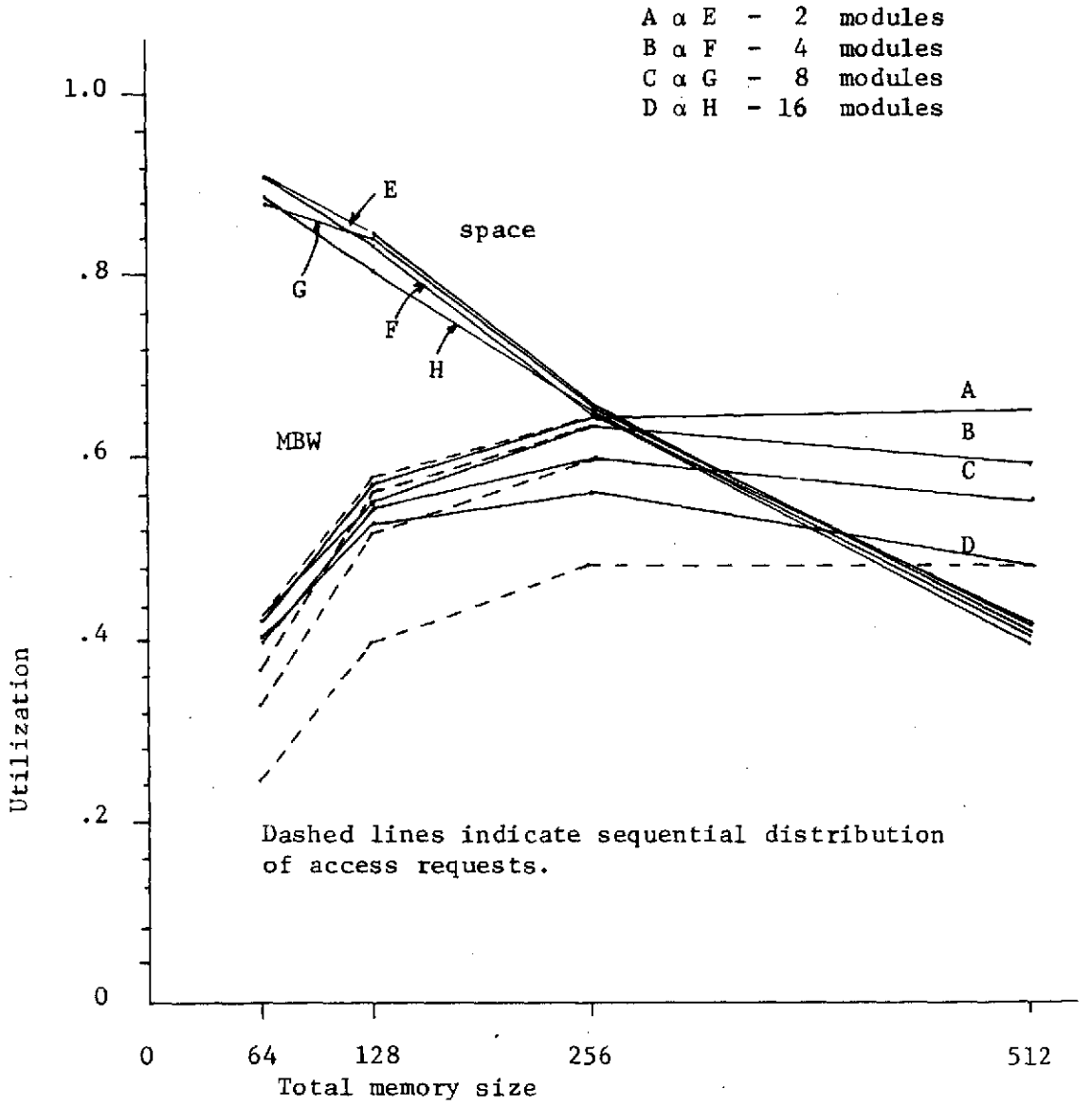


Fig. 6-20 System response as a function of total memory size with widely dispersed bandwidth requirements memory space and bandwidth utilizations.

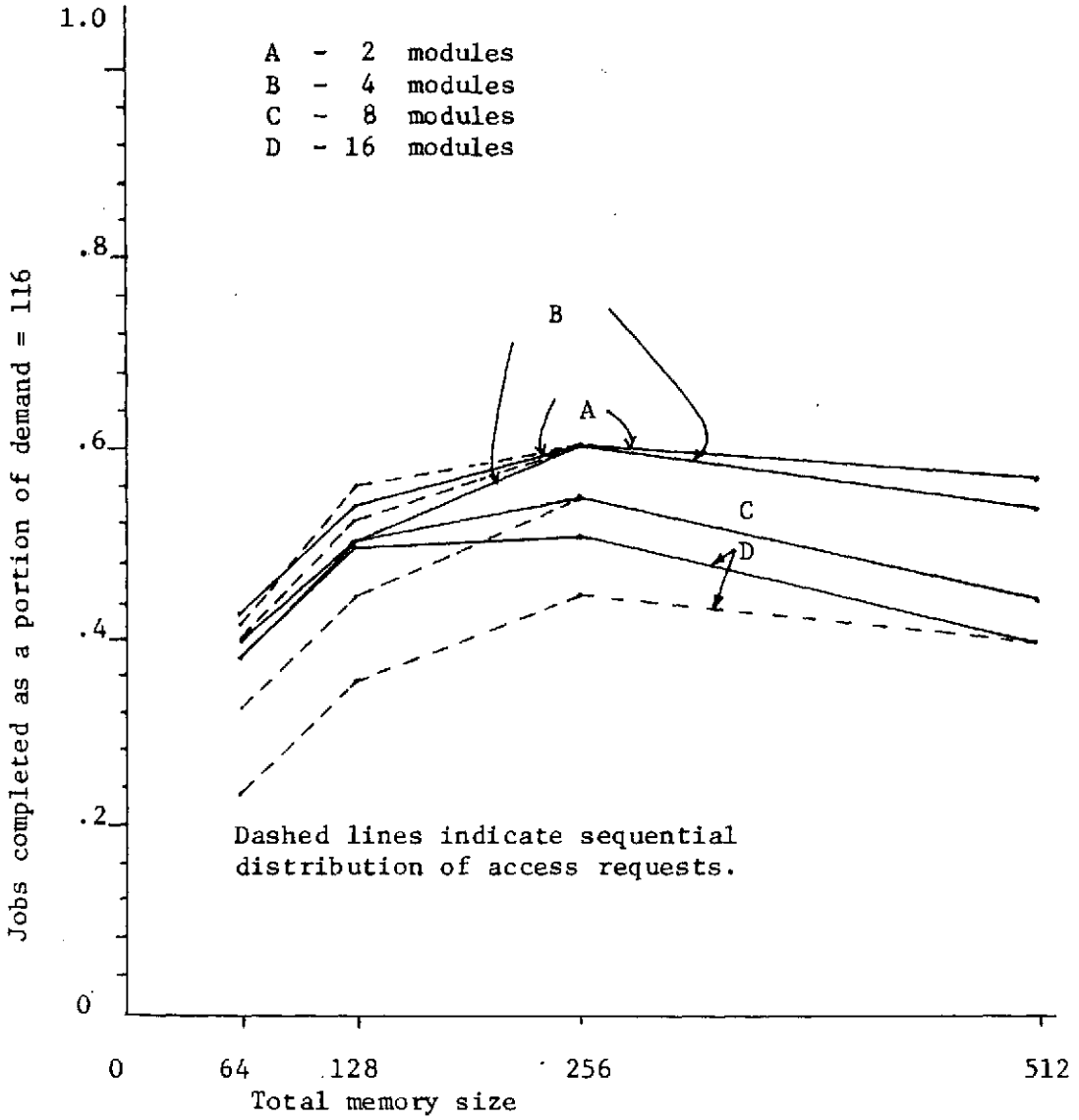


Fig. 6-21 System response as a function of total memory size with widely dispersed bandwidth requirements-Job completions.

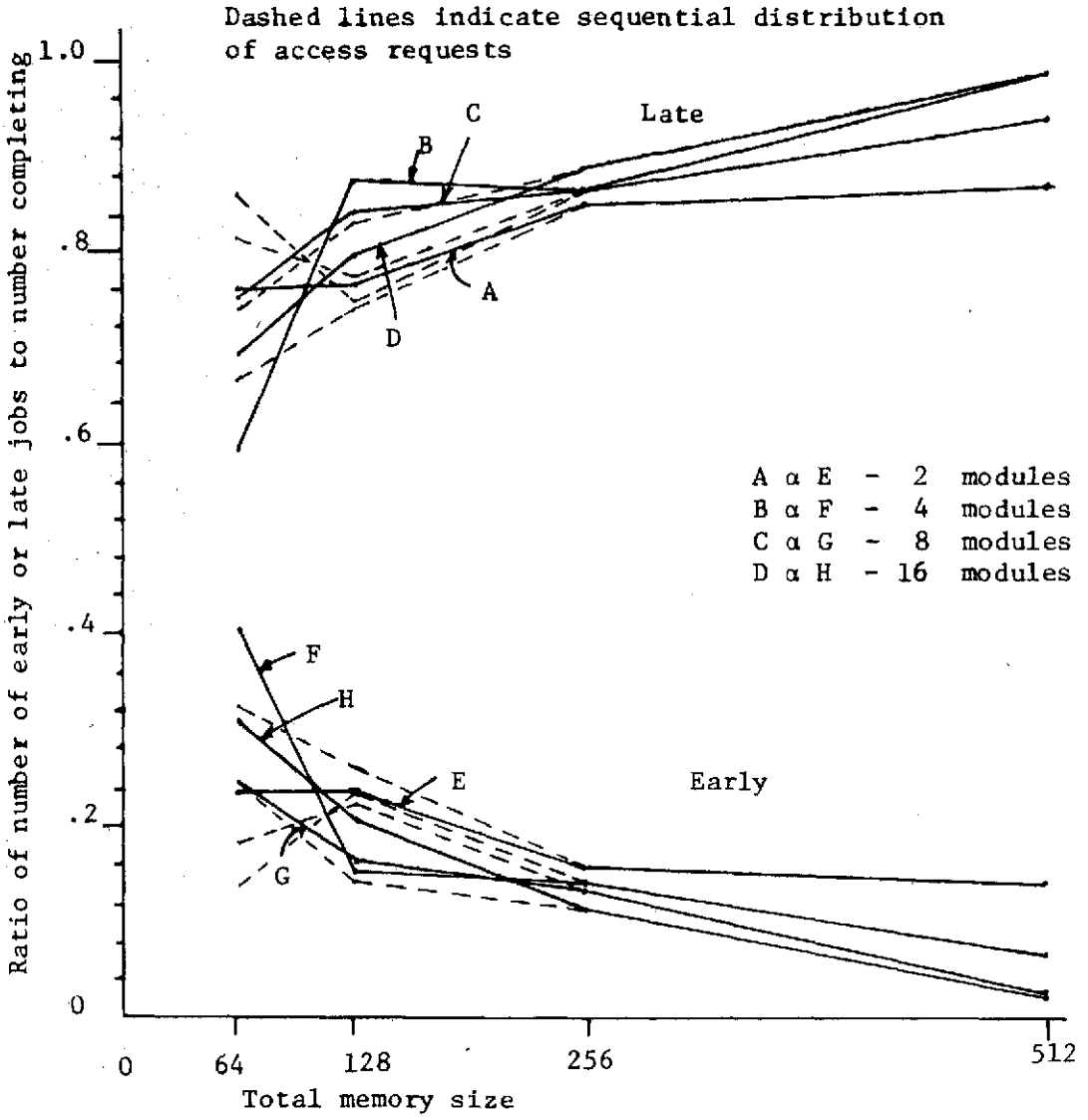


Fig. 6-22 System response as a function of total memory size with widely dispersed bandwidth requirements - Late and early jobs.

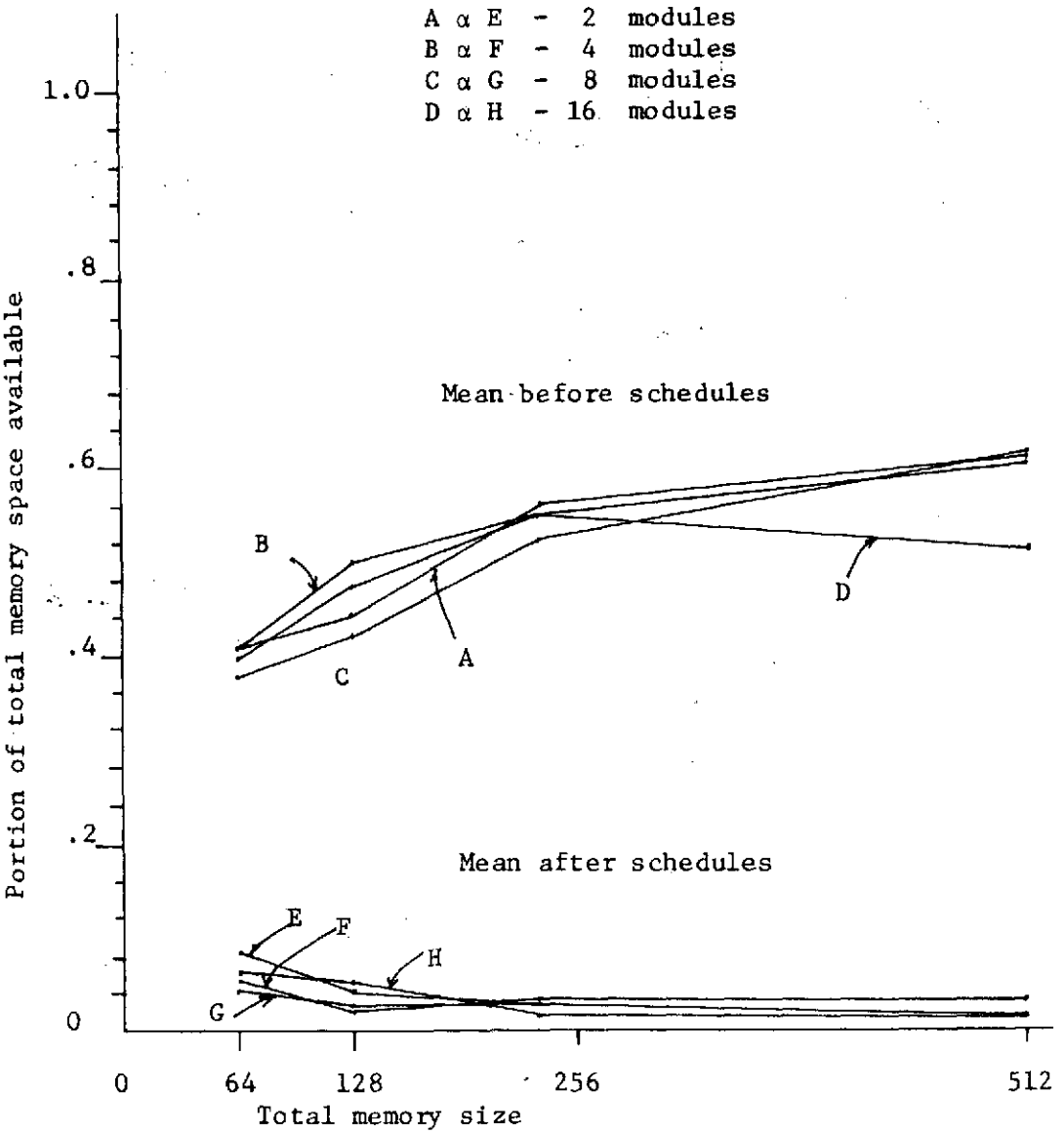


Fig. 6-23 System response as a function of total memory size with widely dispersed bandwidth requirements - Available space before and after schedules

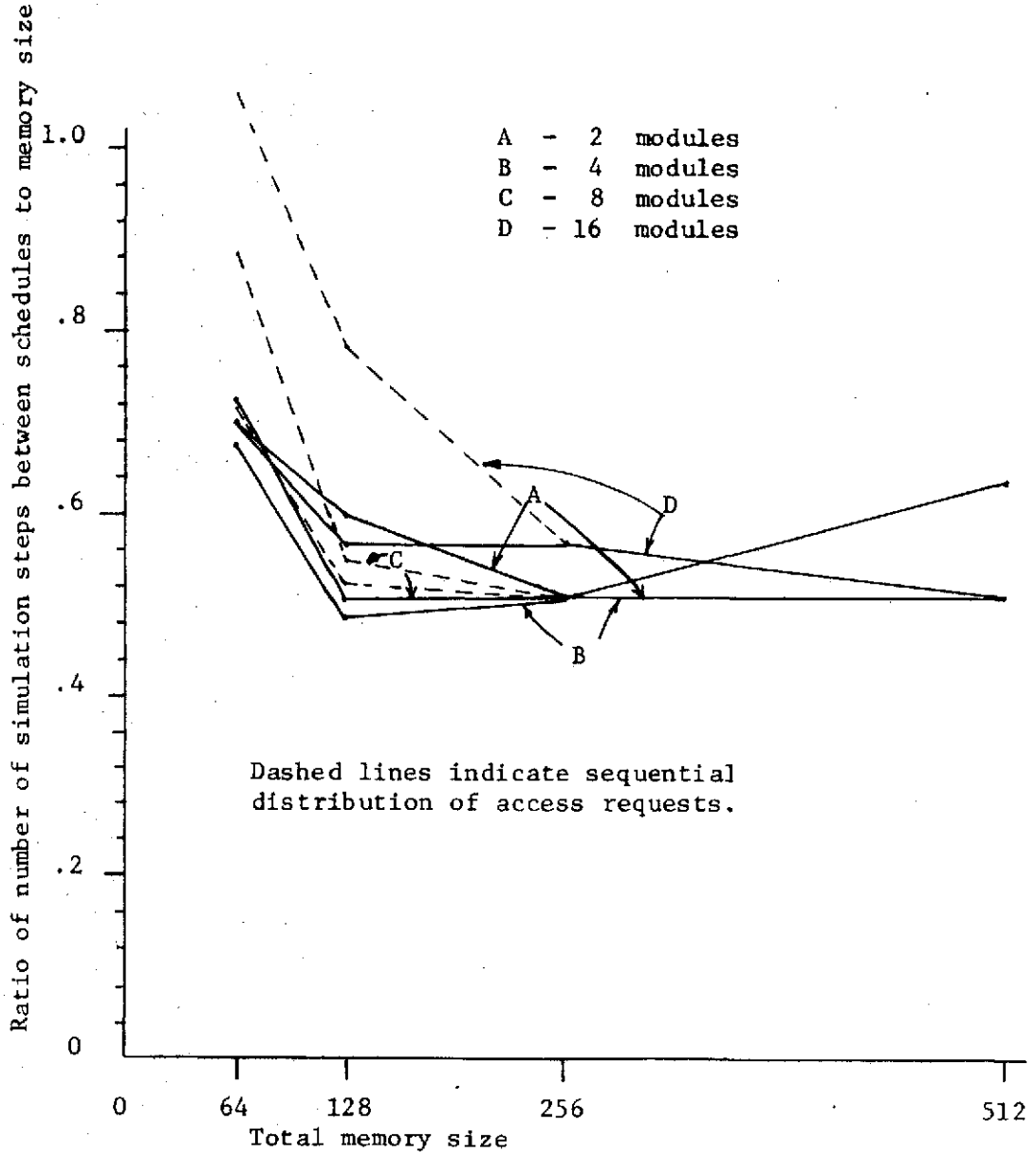


Fig. 6-24 System response as a function of total memory size with widely dispersed bandwidth requirements - Number of steps between schedules.

Relative Comparisons of System Response for Various System Configurations and Numbers of Recurrent Jobs

In this problem two sets of runs were made. In the first set of runs the memory's bandwidth was adjusted to be approximately five and one-half times that required by the job set based on the numbers of jobs, desired interarrival times and total memory accesses. In the second set of runs the memory's total bandwidth was adjusted to be approximately seventy percent of desired total demand. Both sets of runs were made with both feedback and nonfeedback scheduler/allocators for the system configurations shown in Fig. 6-25. The results for these sets of runs are shown in Fig. 6-26 through Fig. 6-35. From these figures it is apparent that the feedback configuration tends to select the jobs requiring larger amounts of space and bandwidth if relatively larger amounts are available and jobs of smaller space and bandwidth requirements if relatively smaller amounts are available, i.e., it works approximately as it should. However, it tends to deviate considerably from the dictates of external priority even under light demands. A certain amount of this tendency is to be expected since maximizing resource utilization is somewhat incompatible with response to individual job demands. Nevertheless the feedback configuration appears to have performed better than the nonfeedback configuration in the limited bandwidth availability case.

	Total memory size				
	32	64	128	192	256
Number of CPUs	1	2	4	6	8
Number of IOPs	2	4	8	12	16
Number of Memory Modules	2	4	8	12	16
Number of recurrent jobs	5	10	20	30	40
Job completion demands	116	108	71	35	21
Mean total access demands per step	176.41	358.71	706.41	1102.24	1432.17
Bandwidth Availability per step — large	1000	2000	4000	6000	8000
Bandwidth Availability per step — limited	125	250	500	750	1000
Job space requirements Range	4000 to 24000	4000 to 24000	4000 to 24000	4000 to 24000	4000 to 24000
Actual mean space Requirement per job	15.6	20.3	18.3	17.0	16.8

Fig. 6-25 Relative comparisons of system response for various system configurations and number of recurrent jobs - System configurations and job characteristics.

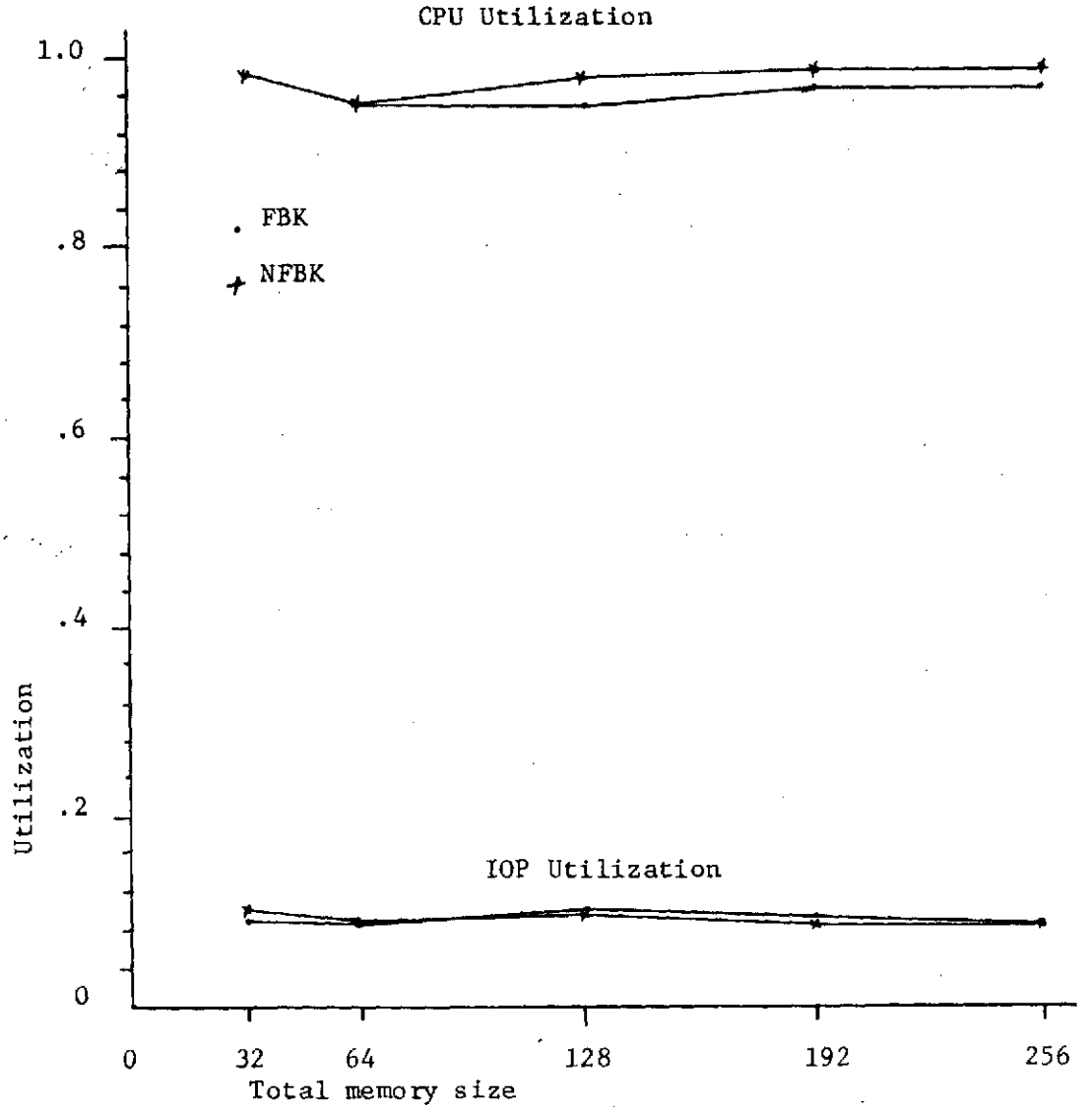


Fig. 6-26 Relative comparisons of system response for various system configurations and number of recurrent jobs, large bandwidth availability - Processor utilizations.

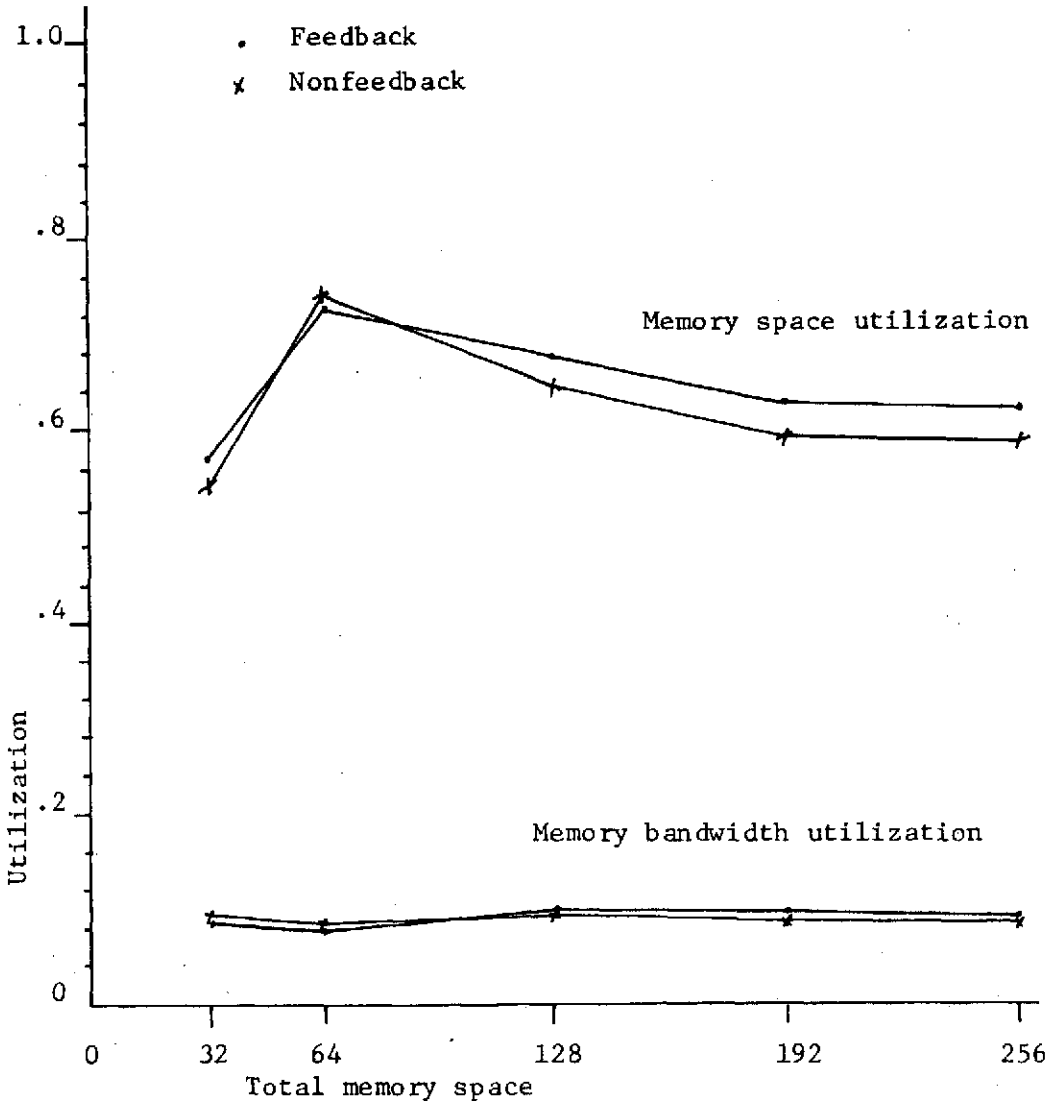


Fig. 6-27 Relative comparisons of system response for various system configurations and number of recurrent jobs, large bandwidth availability - Memory space and bandwidth utilizations.

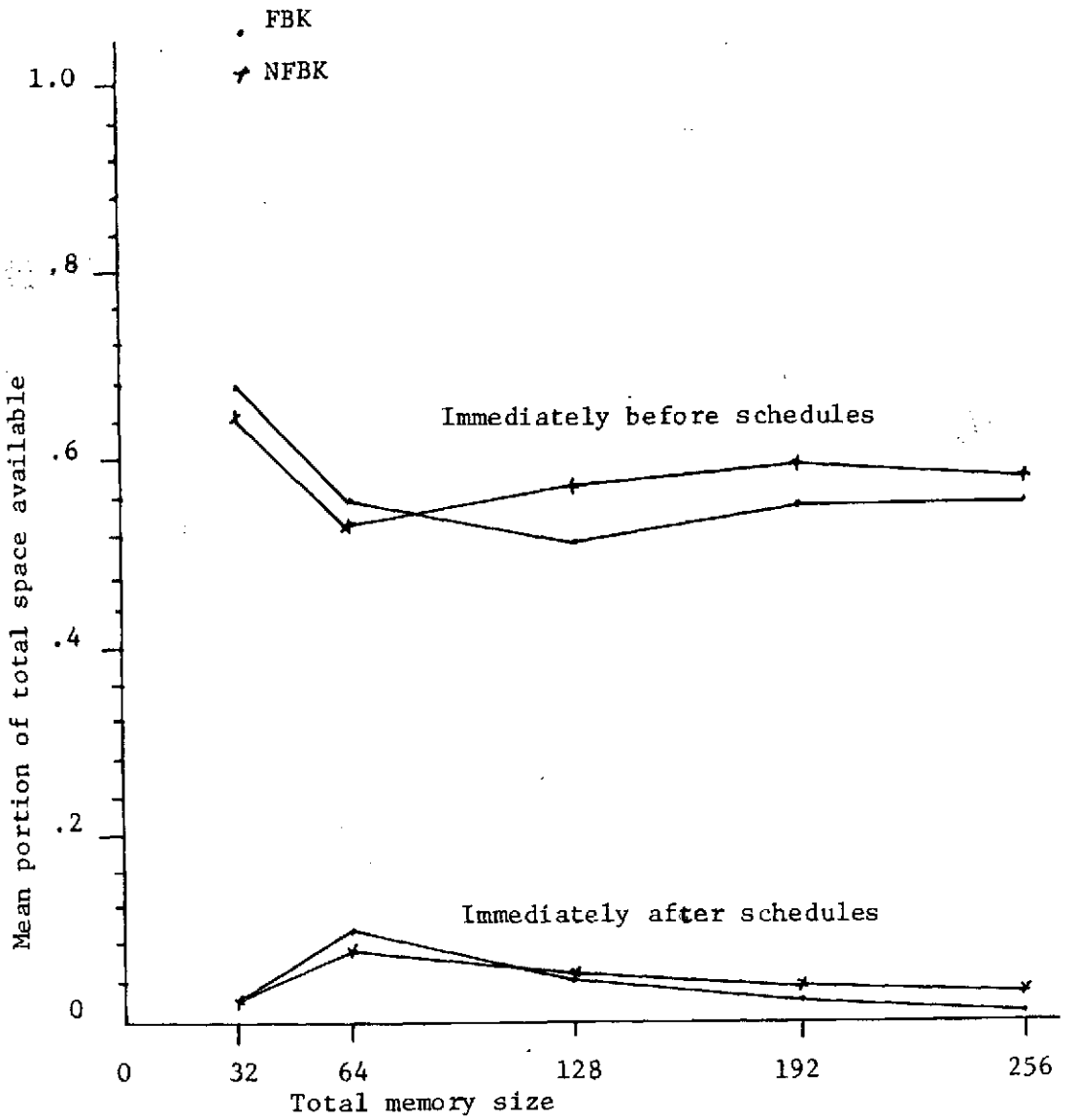


Fig. 6-28 Relative comparisons of system response for various system configurations and number of recurrent jobs, large bandwidth availability - Memory space availability.

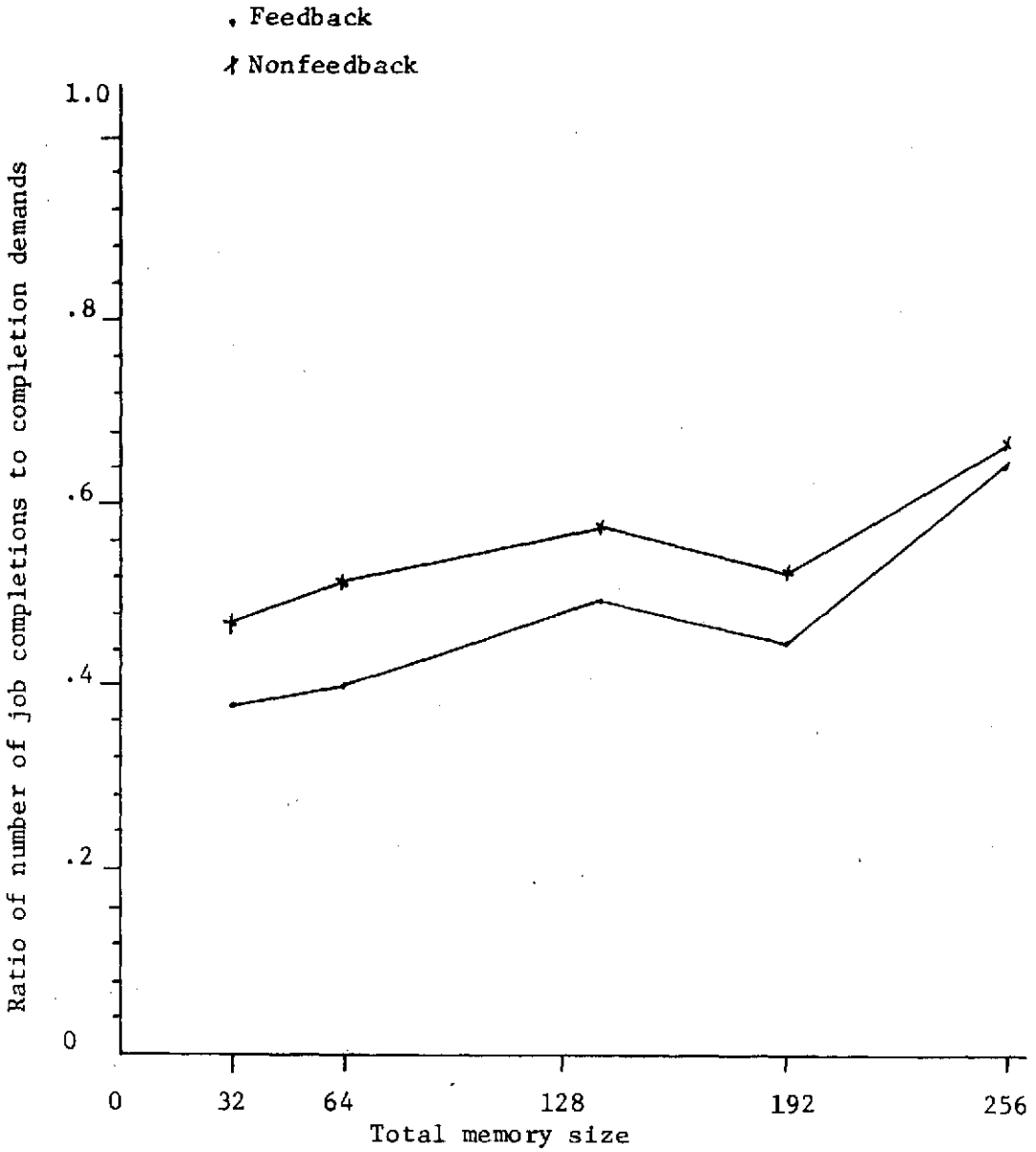


Fig. 6-29 Relative comparisons of system response for various system configurations and number of recurrent jobs, large bandwidth availability - Job completions.

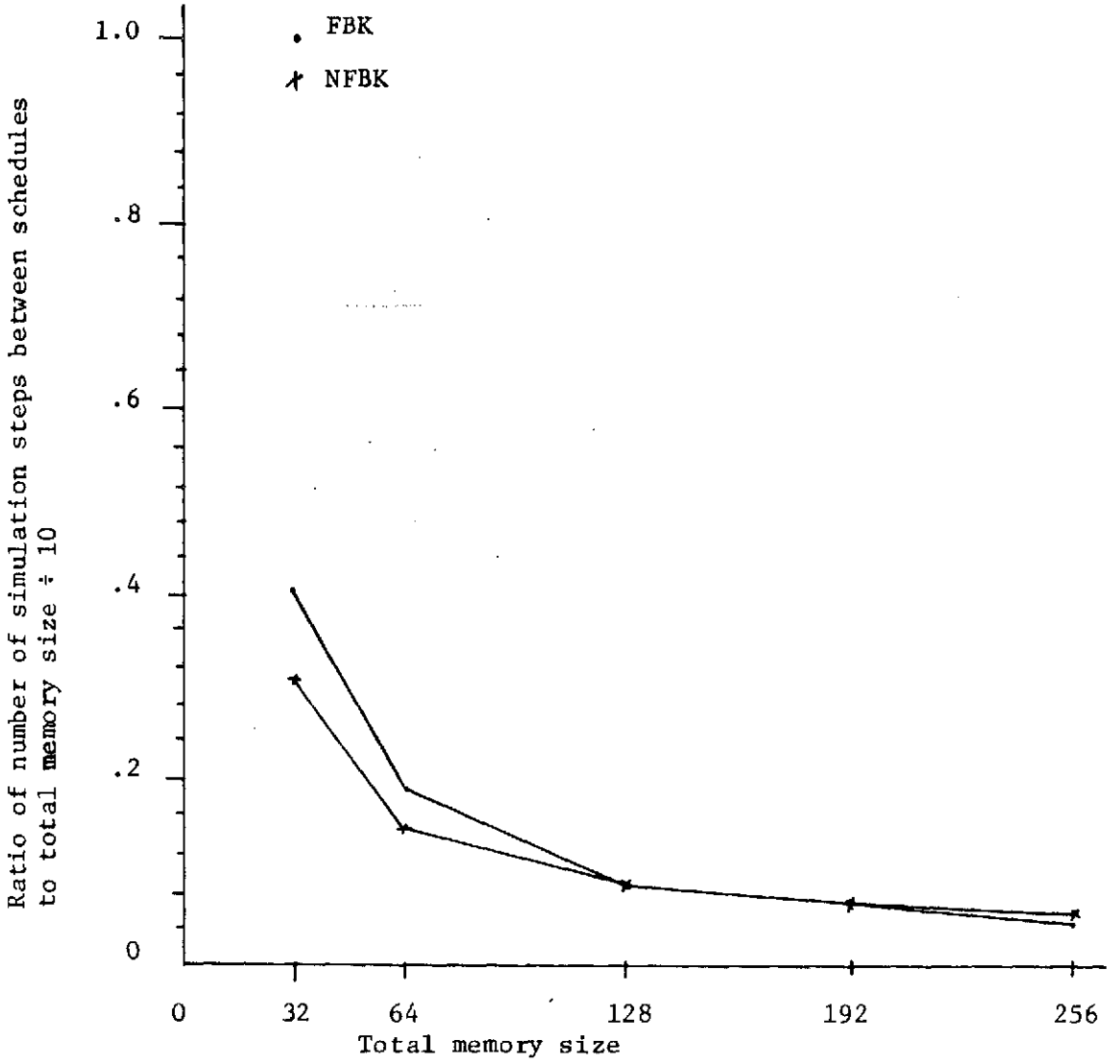


Fig. 6-30 Relative comparisons of system response for various system configurations and number of recurrent jobs, large bandwidth availability - Number of steps between schedules.

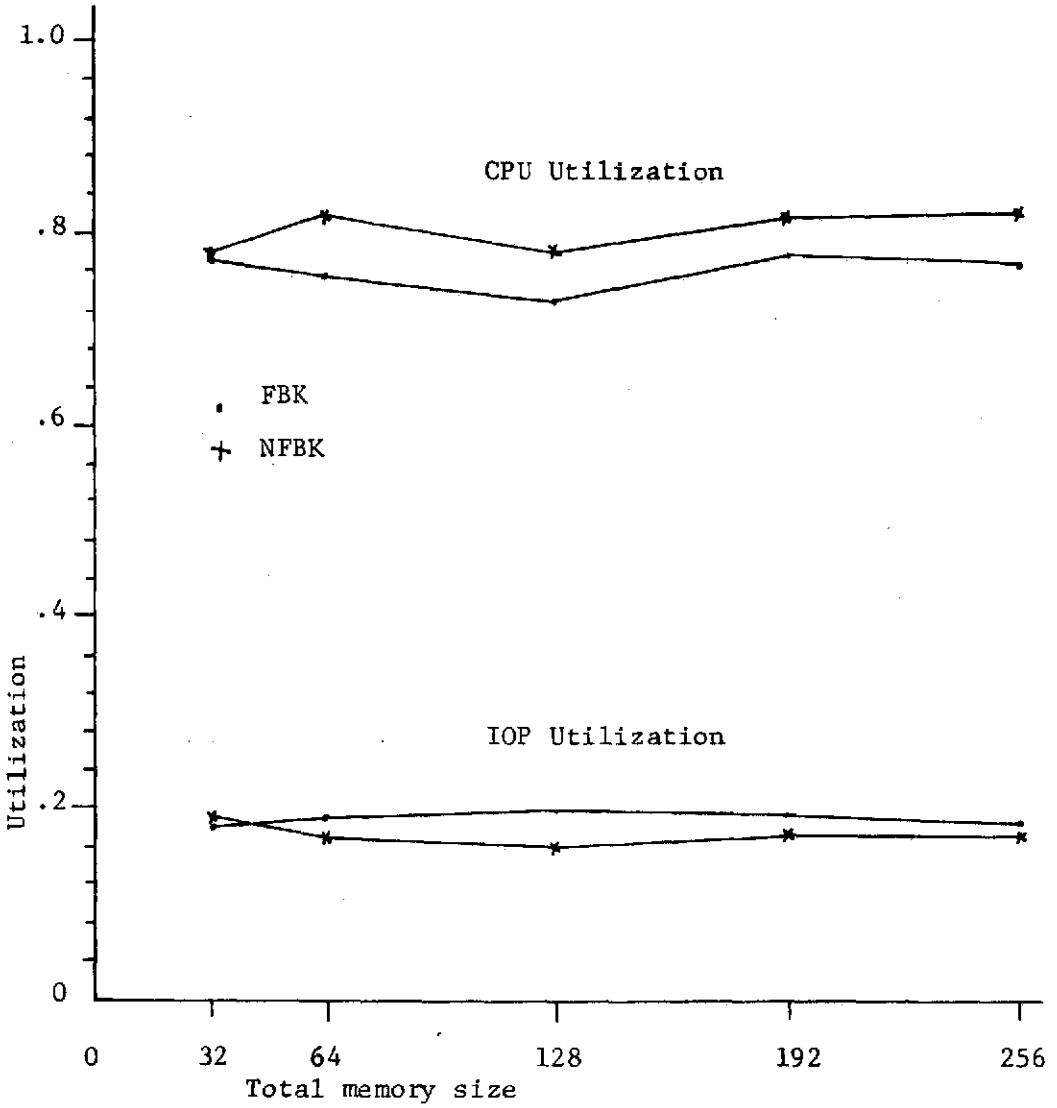


Fig. 6-31 Relative comparisons of system response for various system configurations and number of recurrent jobs, limited bandwidth availability - Processor utilization.

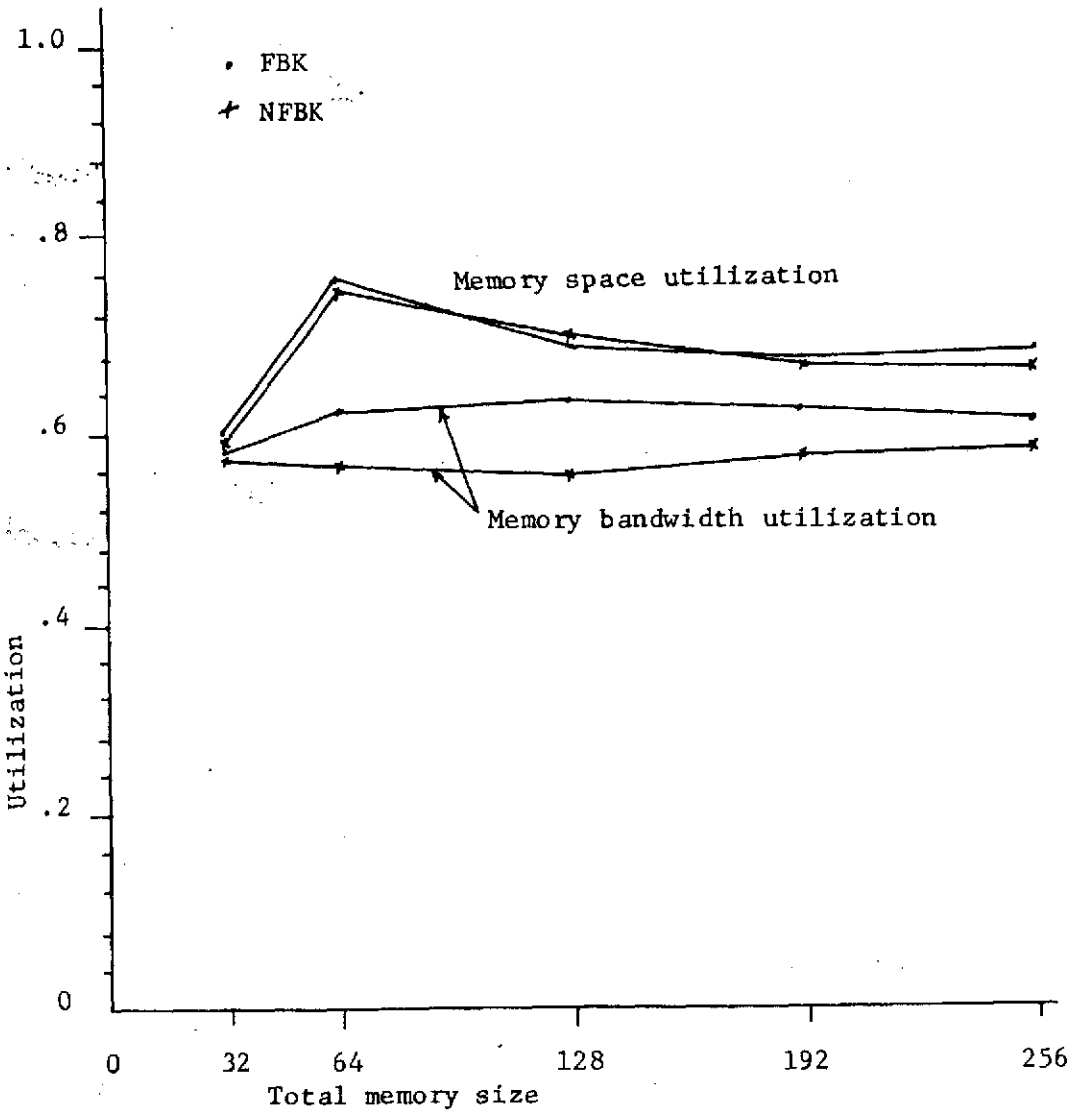


Fig. 6-32 Relative comparisons of system response for various system configurations and number of recurrent jobs, limited bandwidth availability - Memory space and bandwidth utilization.

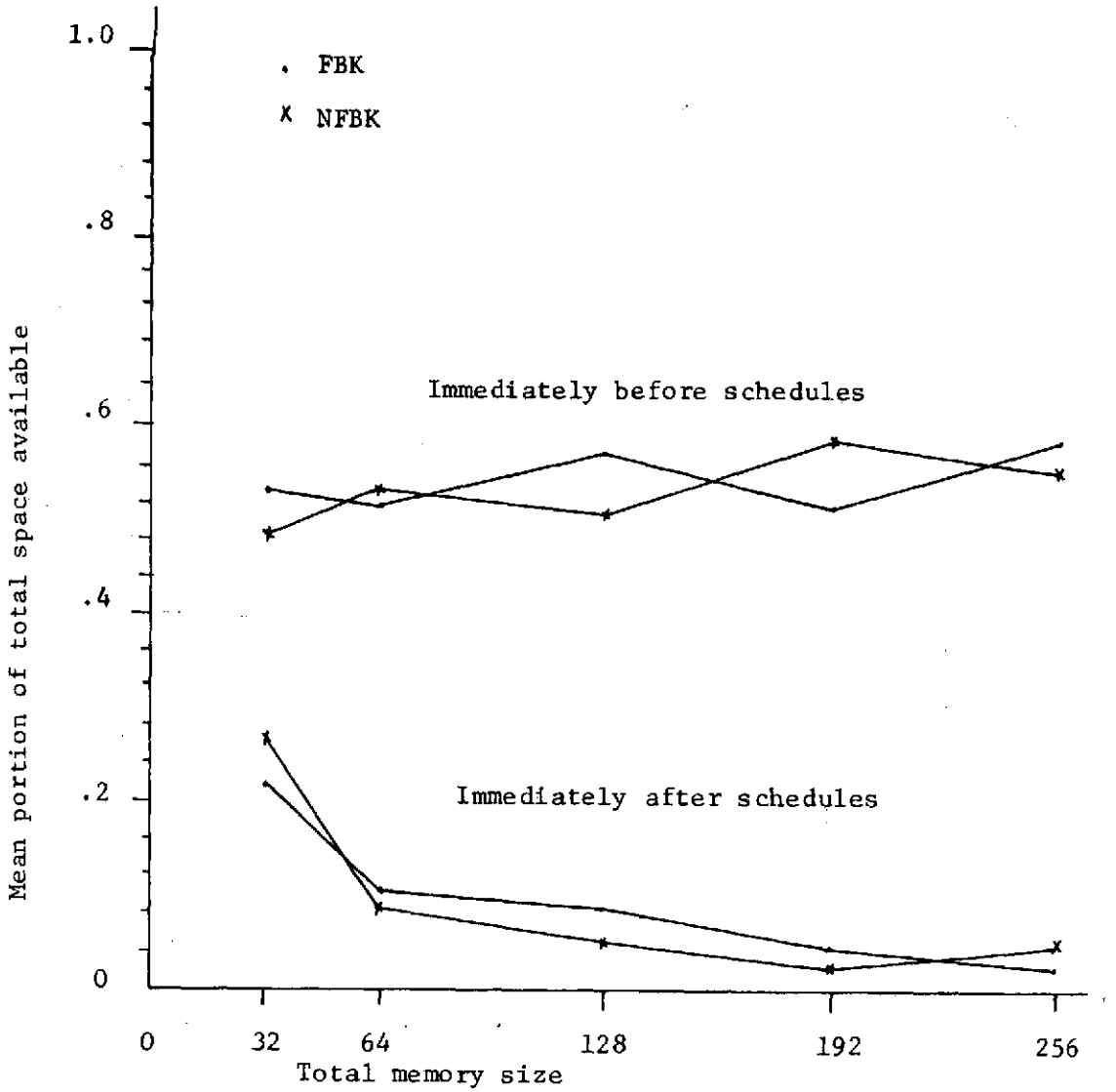


Fig. 6-33 Relative comparisons of system response for various system configurations and number of recurrent jobs, limited bandwidth availability - Memory space availability.

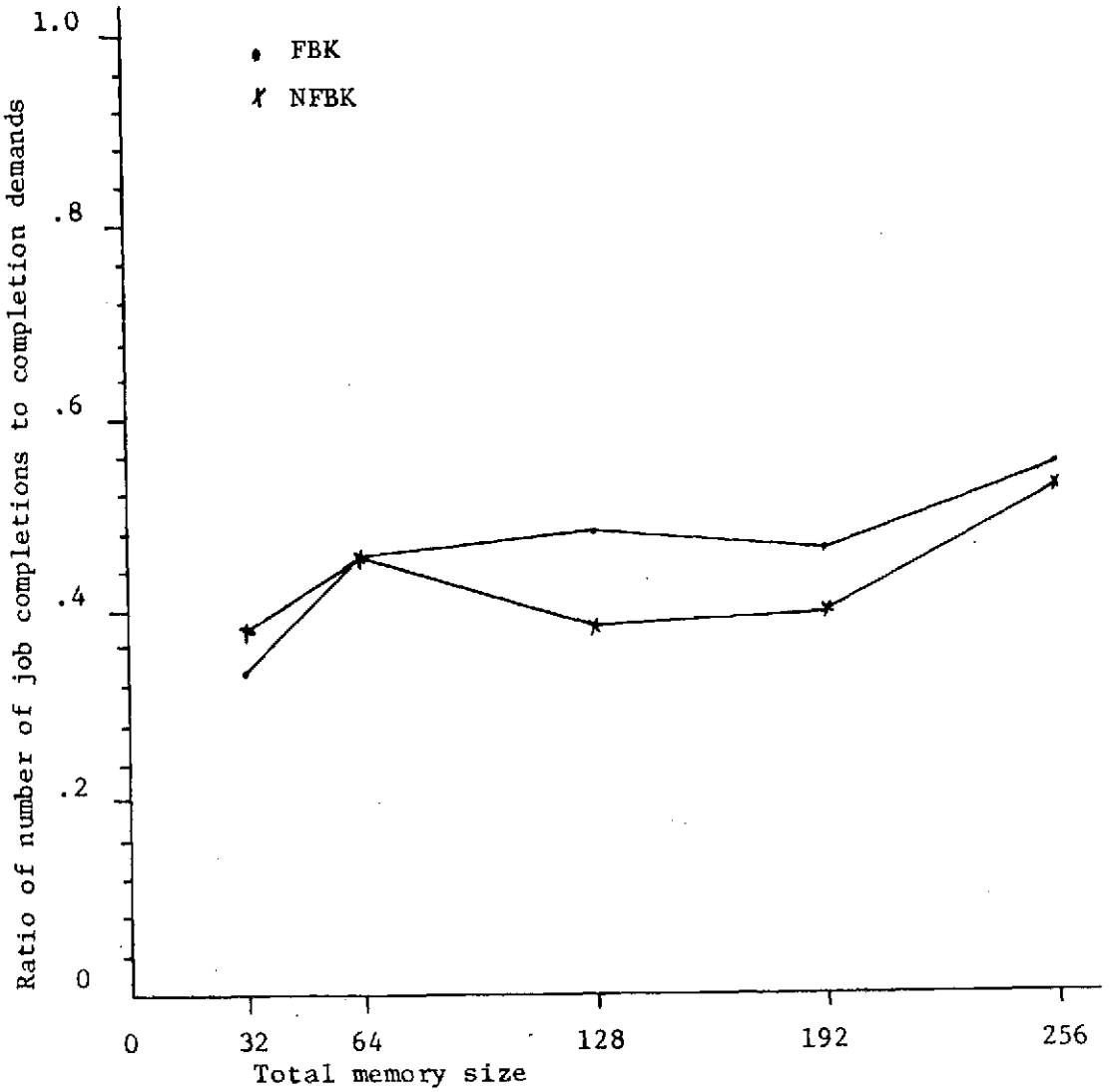


Fig. 6-34 Relative comparisons of system response for various system configurations and number of recurrent jobs, limited bandwidth availability - Job completions.

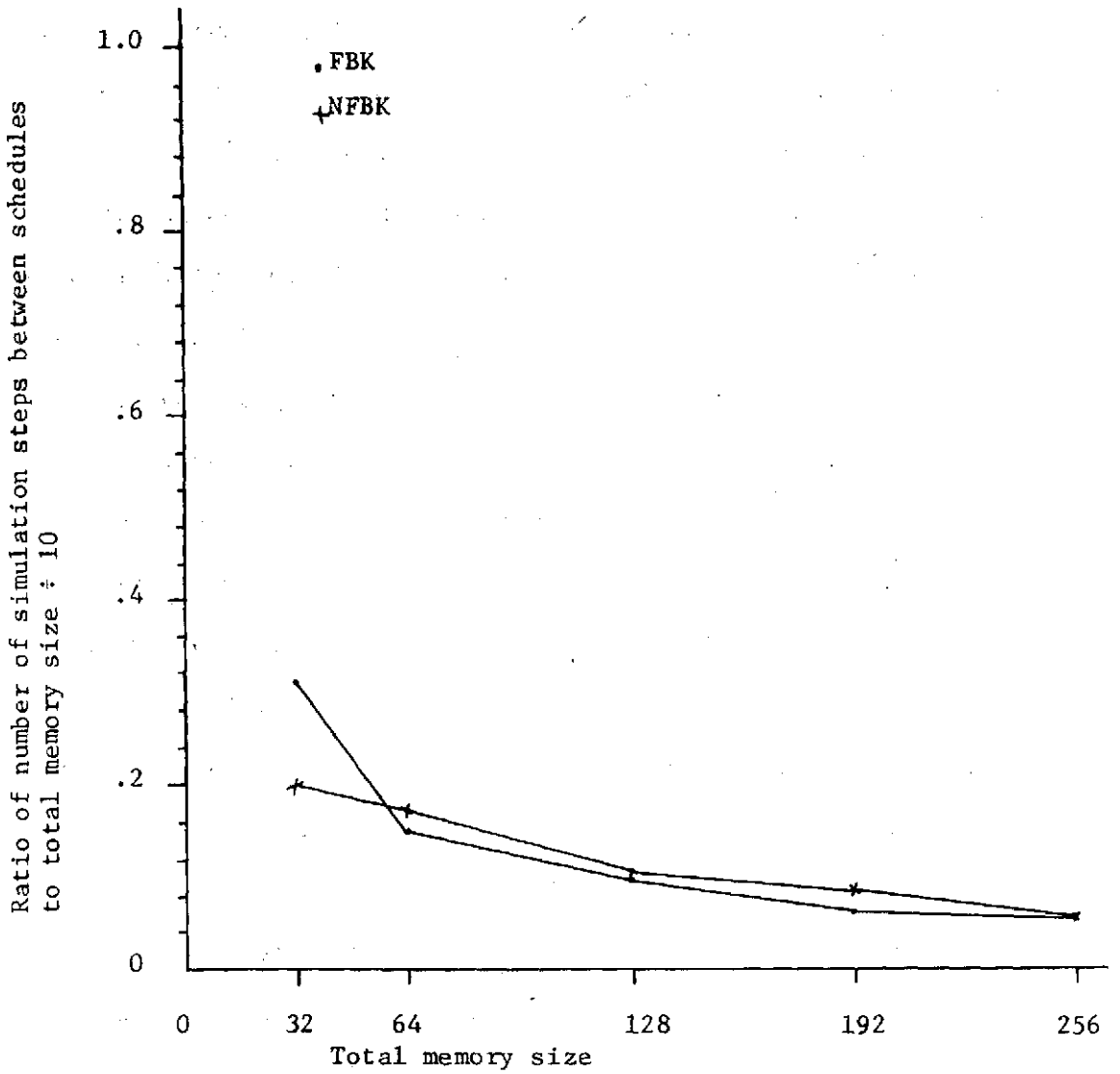


Fig. 6-35 Relative comparisons of system response for various system configurations and number of recurrent jobs, limited bandwidth availability - Simulations steps between schedules.

The Impact of TMR Jobs on the Memory Allocation Problem

One method for improvement of reliability is through triple modular redundancy. [29], [30] The model does not assume that there is a fixed configuration of the memory into three or more parallel segments into which the three components of a TMR job are placed and executed in a lock-step fashion. The model assumes a much more general situation in that both TMR and SIMPLEX jobs may be in progress in the system simultaneously and that allocations of memory to TMR jobs can be made in triplicate from any available memory blocks so long as the three corresponding allocations to a job is made from three disjoint sets of memory modules. This creates a number of problems in the allocation of memory and assignment of processors to the TMR jobs. With factors of external priority equal for both SIMPLEX and TMR jobs the TMR jobs are at a very distinct disadvantage in being able to obtain all the resources that they need, especially with a large overall demand and a limited amount of resources.

For this problem three sets of runs were made. In the first set all the jobs were TMR. In the second set approximately one-half of the jobs were TMR and in the third set one-third of the jobs were TMR. The total number of recurrent jobs was fixed at forty for all three sets of runs. In each set of runs total memory size and bandwidth were fixed but the numbers and speeds of memory modules were varied. In order to keep space demands nearly constant the mean space requirement of jobs was adjusted so that the run with the largest fraction of SIMPLEX jobs

had the largest mean job requirement for space. For all three sets of runs the number of memory accesses required for job completion was adjusted for a small value and bandwidth requirements were adjusted for a high value in order to complete a larger number of jobs so that a large number of allocations would be made.

The system configurations and job characteristics are listed in Fig. 6-36. The results are shown in Fig. 6-37 through Fig. 6-42. These figures indicate that the memory fragmentation problem tends to be greater with larger percentages of TMR jobs. However, Fig. 6-42 indicates that service to TMR jobs relative to that of SIMPLEX jobs tends to be poorer for relatively smaller percentages of the numbers of TMR jobs in the system.

	ALL TMR	½ TMR	1/3 TMR
Job space requirement range	4-24	6-36	8-43
Mean job space requirement	16.8	25.1	29.8
Actual number of TMR jobs	40	22	14
Total space requirement	2016	2108	2026
(Numbers of Modules, speeds of Modules)	(3,6) (6,12), (9,18), (12,24), (15,30)		
Total memory size	360		
Number of CPUs	10		
Number of IOPs	15		

Fig. 6-36 The impact of TMR jobs on the memory allocation problem -- Job characteristics and system configurations.

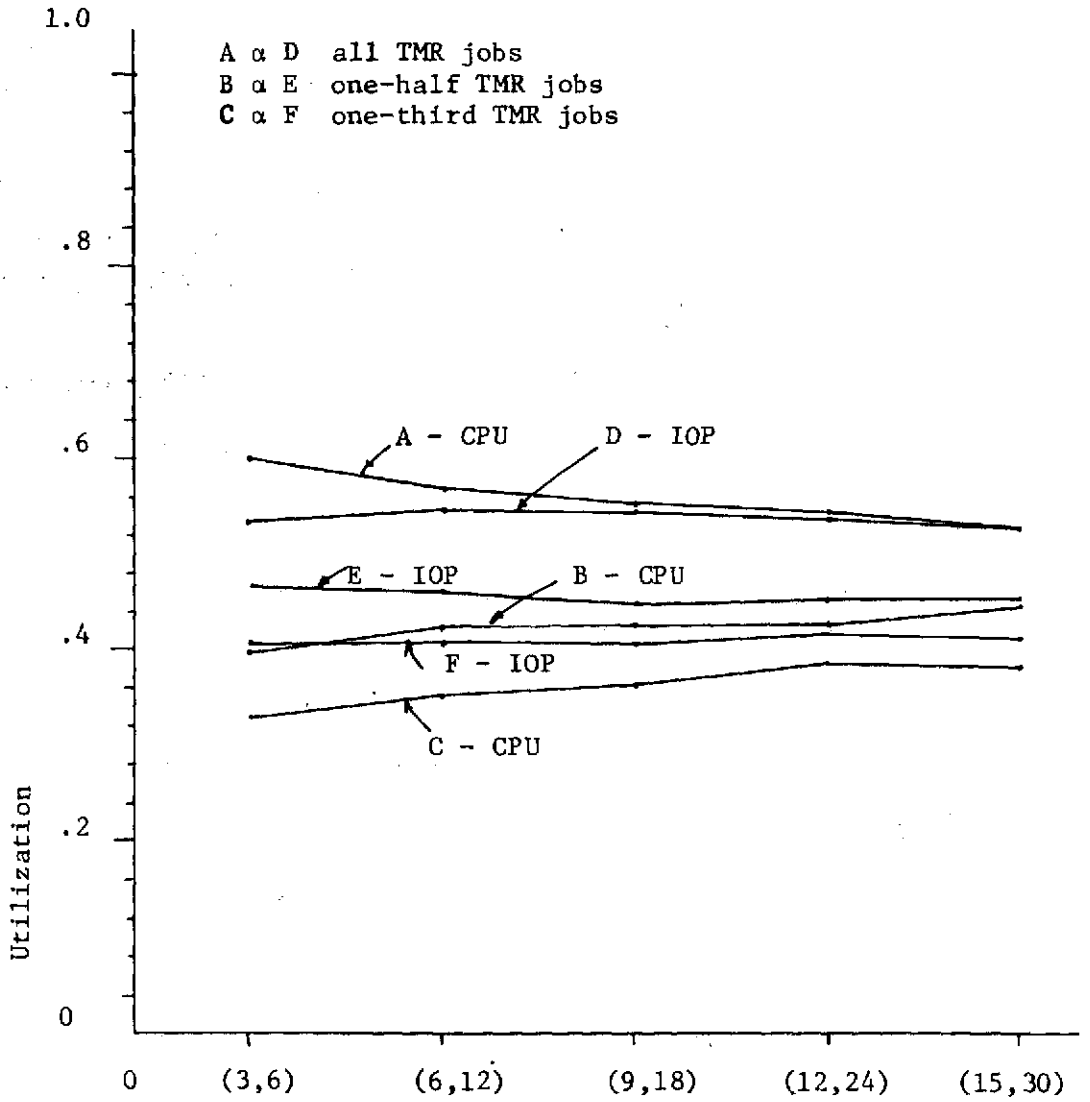
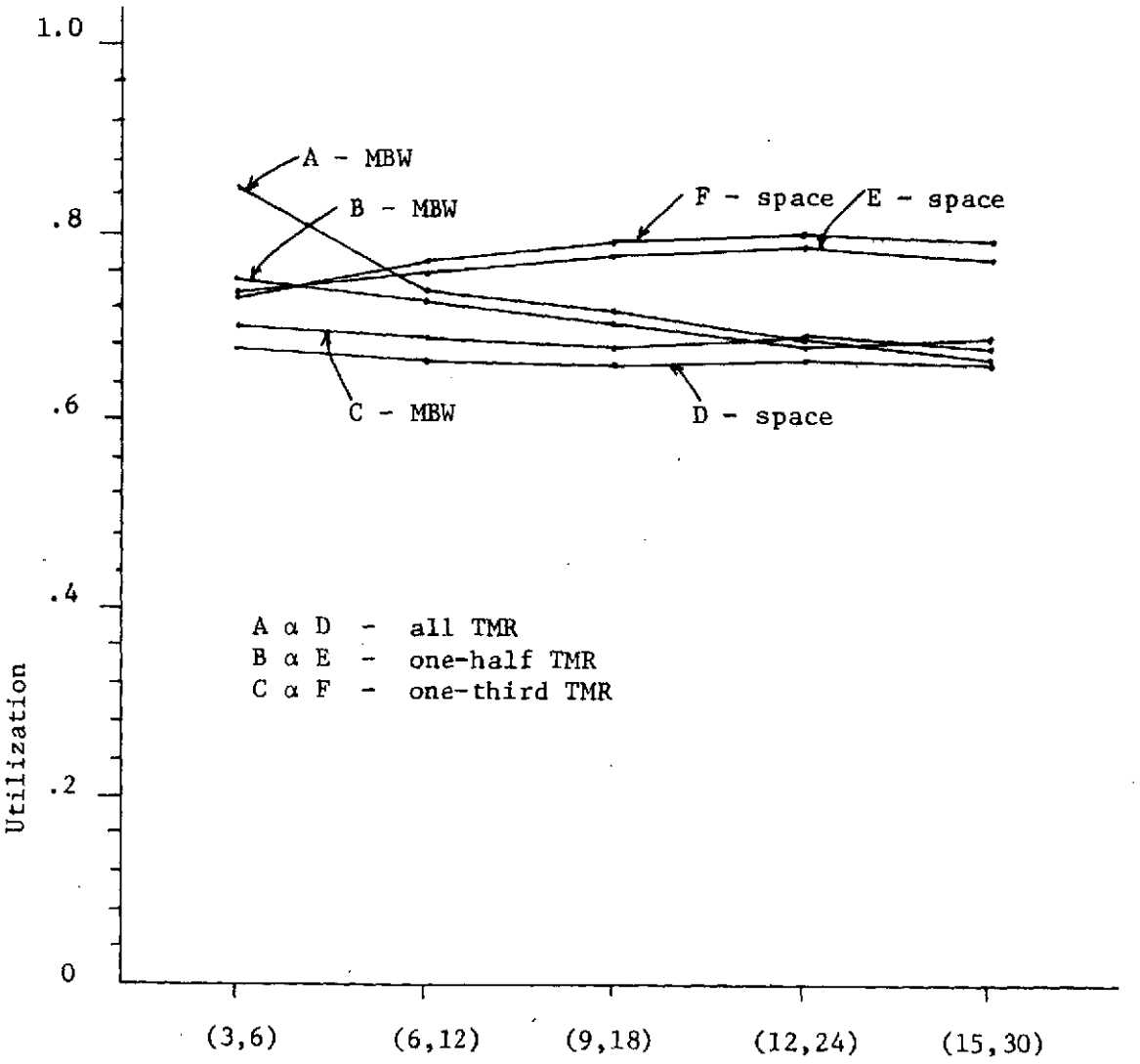


Fig. 6-37 The impact of TMR jobs on the memory allocation problem - Processor utilization.



(Number of Modules, Module speed), Module speed is in basic clock periods per module cycle period.

Fig. 6-38 The impace of TMR jobs on the memory allocation problem - Memory space and bandwidth utilizations.

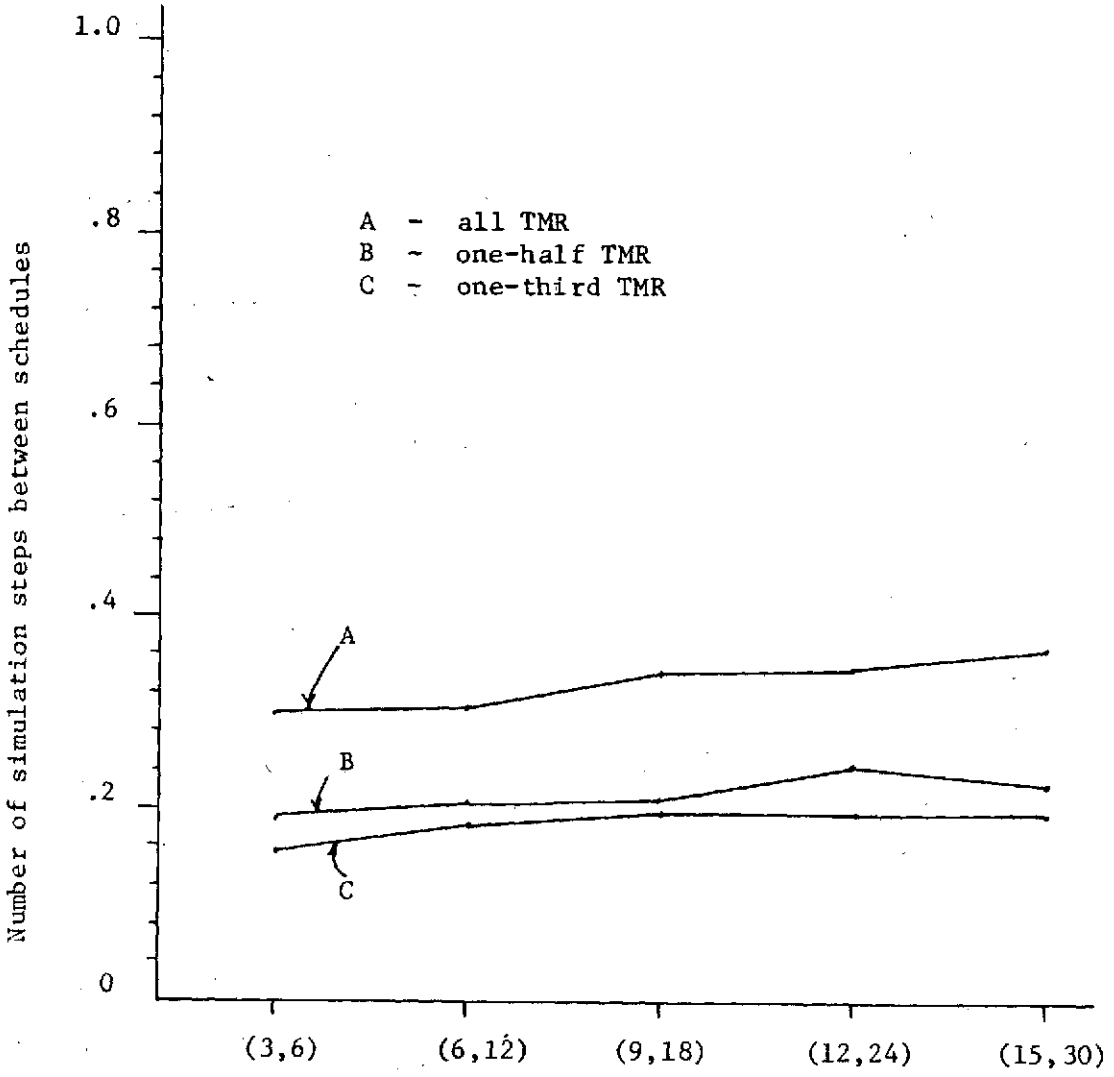
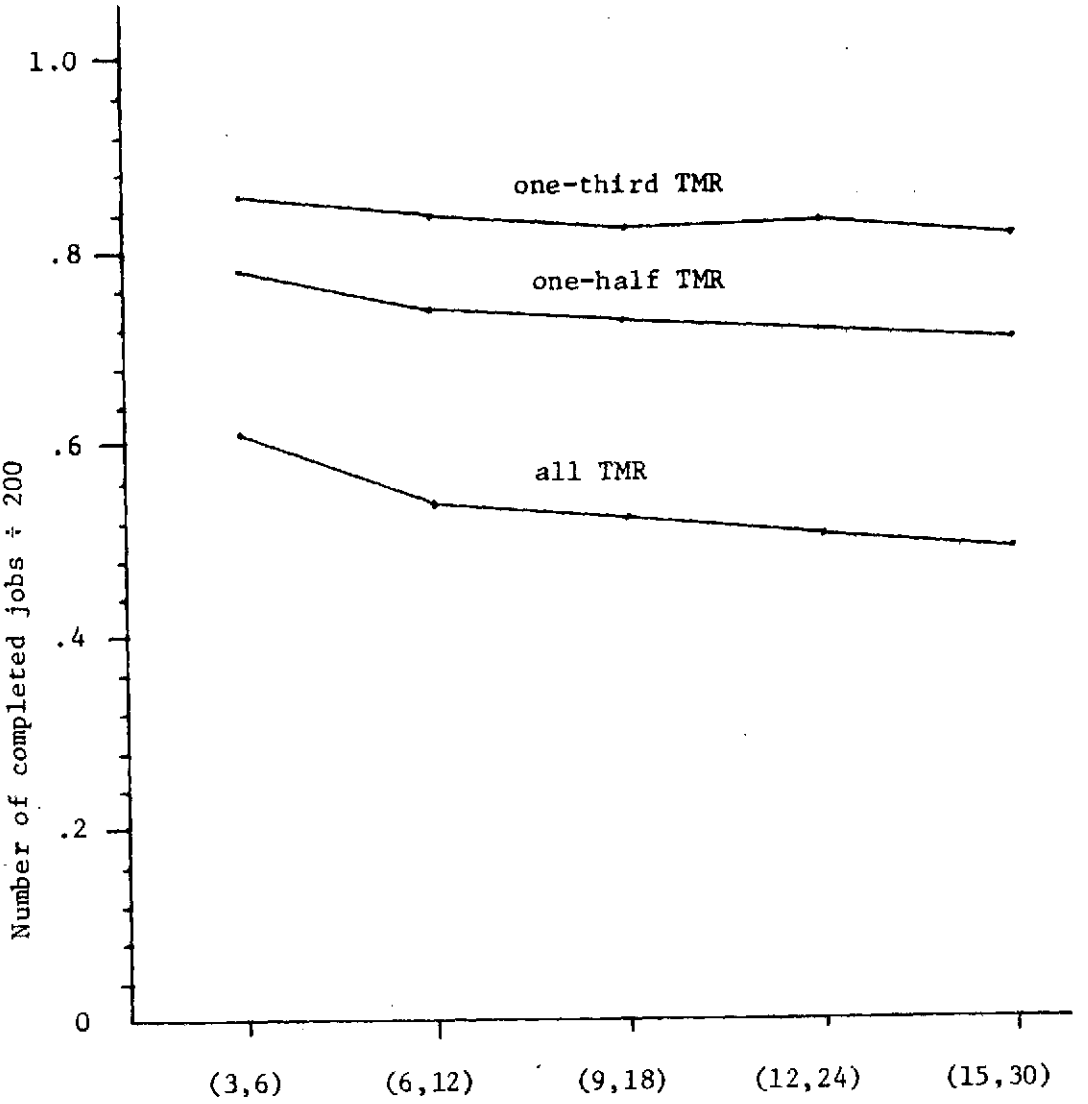


Fig. 6-39 The impact of TMR jobs on the memory allocation problem - Number of simulation steps between schedules.



(Number of Modules, Module Speed), Module speed is in basic clock periods per module cycle period.

Fig. 6-40 The impact of TMR jobs on the memory allocation problem - Number of jobs completed.

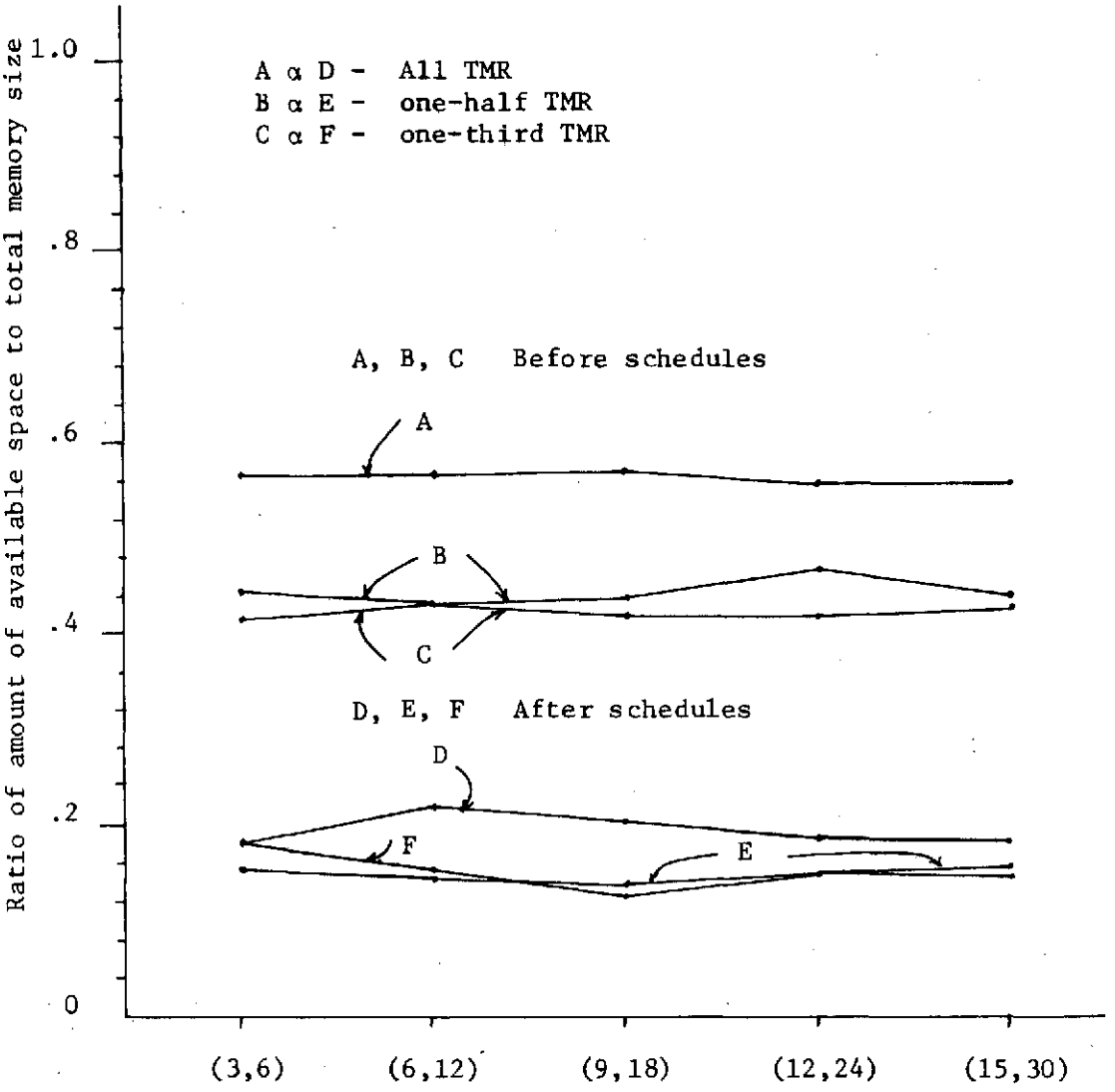


Fig. 6-41 The impact of TMR jobs on the memory allocation problem - Available space immediately before and after schedules.

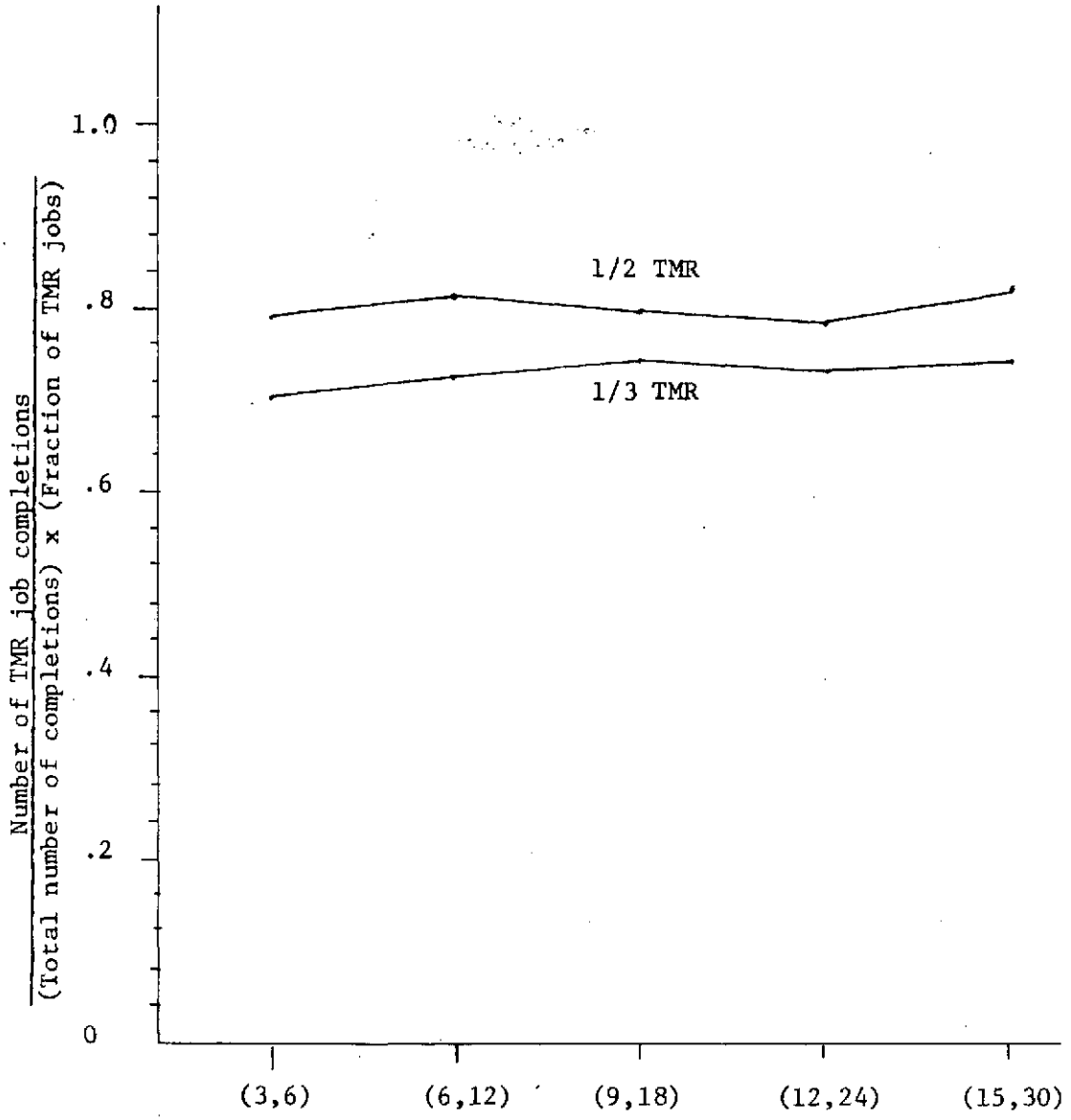


Fig. 6-42 The impact of TMR jobs on the memory allocation problem - Relative numbers of completions for TMR jobs.

VII. DISCUSSION AND CONCLUSIONS

A simulation model for a multiprocessor computer has been developed in order to study a few of the myriad of interacting problems that are associated with such systems. Of necessity, the model is a highly simplified embodiment of one view of what are the important constituents of such a system in regards to the types of problems to be investigated. These problems were mainly concerned with the effects of the aggregate characteristics of input job sets to the system on the output of the system as well as the effects of interactions of hardware processors in simultaneously using a common memory having limited space and bandwidth.

Although the model performs many of the system executive functions of a real multiprocessor, the impact of executive or monitor type processing does not show up explicitely in the simulation results since this processing is done essentially between simulation steps. Instead, the main thrust of this paper was directed at a stratum of problems lying at least one level below some nebulous but ever present executive system. Nevertheless, the problems actually investigated form a large part of the basic enviroment within which an executive processing system must operate.

The basic underlying assumptions for the simulation model were presented in Chapter II and some typical problems suitable for investigation by means of the model were presented in Chapter VI. Due to the granular manner, in reference to time, in which the model simulates the execution of programs the effects of access conflict, which occur when two or more

processors attempt to simultaneously use a common memory module, do not show up in the simulation results. Thus in Chapter III this problem was studied by analytical means. This analysis employed a modified binomial approach in which the sequential instruction by instruction execution of computer programs, reflecting remembrance by processors of the last memory access request not yet granted, and mean bandwidth requirements were taken into account. The results of this analysis showed that when the sum of the mean bandwidth requirements of all jobs being simultaneously executed from a common memory module is equal to the bandwidth of the module, access conflict can cause an overall slow down ranging from approximately seventeen percent with three processors to a slowly decreasing value of approximately eleven percent for sixteen processors.

A multiprocessor is quite often viewed as a large and powerful computing system in which several processors simultaneously share a common memory. Thus this common memory and the interactions between hardware processors in simultaneous use of the memory is of central importance in such a system. Increases in efficiency of use of the memory of only a small fraction can represent a large amount of computing power. The model attempts to increase the use of the memory's bandwidth by matching individual job requirements for space and bandwidth with that available from the memory at the time of allocation. Therefore in Chapter IV this problem was studied in order to determine expected maximum gains that a simple bandwidth matching shceme of the form assumed by the model could obtain. This analysis indicated that the best that can be obtained over a random placement scheme ranges from approximately twenty percent for two jobs being simultaneously executed from a common memory module to

approximately six percent when twelve to sixteen jobs are simultaneously executed from a common memory module under the assumptions that were listed for space and bandwidth requirements in that section.

In Chapter V a processor to memory interconnection and conflict resolution scheme was developed. Though not a perfect realization of the stated objectives for such a scheme, the one developed in that section is believed to be viable in a real priority driven multiprocessor. It is also believed that this scheme may easily be modularized for reliability enhancement purposes or expansion of an existing system.

In Chapter VI the simulations indicate that the memory fragmentation problem is not as severe as might at first be expected. Thus in a real system relocation of programs so as to collect the small unusable blocks into larger blocks of storage may not be necessary if the small overhead associated with the fragmentation that was observed concerning this problem is tolerable.

The simulations indicate that the feedback scheduler/allocator scheme can increase memory bandwidth utilization by as much as ten to fifteen percent over a nonfeedback scheme under conditions for which real choices actually exist for bandwidth matching. Thus the values determined in Chapter IV appear to be supported by the simulation results. The increased bandwidth utilization is quite often accompanied by a distortion in relative service to individual job demands from that specified by external priority. In situations in which no real choices actually exist for bandwidth matching there appears to be no substantial differences between the feedback and nonfeedback scheduler/allocators.

Memory interleaving would obviate the need for the feedback method of bandwidth matching but interleaving of memory tends to destroy reliability potential. Therefore interleaving was not considered in this paper. It should be pointed out that the method of bandwidth matching described in the model only assumed knowledge of an instruction mix parameter associated with each job. In this regard it should be contrasted with other methods for bandwidth matching that have recently been described by Covo [31] and by Kruzberg [32].

The simulations point out quite clearly that, aside from differences in cost and reliability, bigger and faster is better in regard to memory modules. This is due to the fact that an individual job's peak demands for bandwidth have a better chance for being fulfilled under conditions of packing of jobs into one end of the memory. This packing is desirable under generalized conditions in which job space requirements are widely dispersed and it is thereby essential to keep as much of the total available space as possible in one contiguous block so that the demands for large contiguous blocks of space have a high chance for being met.

The memory fragmentation problem is worse for all TMR or a mixture of TMR and SIMPLEX jobs than for the all SIMPLEX case but it does not appear that it is three times as bad as for the all SIMPLEX case. Likewise, service to the TMR jobs in a mixed enviroment is considerably less than to SIMPLEX jobs but the difference is not as great as the difference between resource requirements. For example, in the simulation case involving approximately equal numbers of TMR and SIMPLEX jobs the SIMPLEX

jobs received approximately 1.2/.8 = 1.5 times the service that the TMR jobs obtained.

There seems to be few universal truths associated with a multiprocessor system but one observation that may come close to being one is that a key factor in determining a successful multiprocessor system is to provide a large enough memory so that enough jobs may be loaded into the memory at one time such that system resources may be utilized at high levels for relatively long periods of time between executive processing. This observation is supported by the relationships between frequency of schedules and total memory size in the simulations that were performed in this paper.

LIST OF REFERENCES

[1] G. J. Burnett, 'Performance Analysis of Interleaved Memory Systems,' Ph. D. Dissertation, Princeton University, 127 pp., University Microfilms, High Wycomb, England, 1969.

[2] G. J. Burnett and E. G. Coffman, 'A Study of Interleaved Memory Systems,' AFIPS, Spring Joint Computer Conference, 1970, pp. 467-474.

[3] C. J. Conti, 'Concepts for Buffer Storage,' Computer Group News, March, 1969, pp. 9-13.

[4] C. J. Conti, D. H. Gibson, and S. H. Pitkowsky, 'Structural Aspects of the System 360 Model 85,' IBM Systems Journal, Vol. 7, No. 1, 1968.

[5] J. S. Liptay, 'Structural Aspects of the System 360 Model 85, II The Cache,' IBM Systems Journal, Vol. 7, No. 1, pp. 15-21, 1968.

[6] J. S. Liptay, 'The Model 85 Buffer Storage,' IBM Systems Journal, Vol. 7, No. 1, 1968.

[7] R. K. Richards, Electronic Digital Systems, John Wiley & Sons, New York, 1966.

[8] A. L. Leiner, A. W. Notz, J. L. Smith, and A. Weinberger, 'Pilot, A New Multiple Computer System,' Journal of the ACM, Vol. 6, No. 3, pp. 313-335, 1959.

[9] H. S. Bright, 'A Philco Multiprocessing System,' AFIPS Proc. SJCC, Vol. 26, pp. 97-141, 1964.

[10] A. J. Critchlow, 'Generalized Multiprocessing adn Multiprogramming Systems,' AFIPS Proc. FJCC 1963, pp. 107-126.

[11] J. A. Campbell, 'A Note on an Optimal-Fit Method for Dynamic Allocation of Storage,' The Computer Journal, Vol. 14, No. 1, pp. 7-9.

[12] D. E. Knuth, The Art of Computer Programming Volume 2 / Semi-numerical Algorithms, Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.

[13] N. Wiener, Extrapolation, Interpolation, and Smoothing of Stationary Time Series, M. I. T. Press, Cambridge, Massachusetts, 1949.

[14] W. Feller, An Introduction to Probability Theory and Its Applications, Vol. II, John Wiley & Sons, New York, 1971.

[15] H. Lorin, Parallelism in Hardware and Software: Real and Apparent Concurrency, Prentice-Hall Inc., Englewood, New Jersey, 1972.

[16] T. C. Hu, 'Parallel Sequencing and Assembly Line Problems,' Operations Research, Vol. 9, November, 1961, pp. 841-848.

[17] C. V. Ramamoorthy, K. M. Chandy and M. J. Gonzales, Jr., 'Optimal Scheduling Strategies in a Multiprocessor System,' IEEE Transactions on Computers, Vol., C21, No. 2, February, 1972, pp. 137-146.

[18] C. V. Ramamoorthy and M. J. Gonzales, Jr., 'A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs,' AFIPS Conference Proceedings FICC, 1969, pp. 1-15.

[19] R. R. Muntz and E. G. Coffman, Jr., 'Optimal Preemptive Scheduling on Two-Processor Systems,' IEEE Transactions on Computers, Vol. C18, No. 11, November, 1969, pp. 1014-1020.

[20] R. L. Graham, 'Bounds for Certain Multiprocessor Anomalies,' The Bell System Technical Journal, November, 1966, pp. 1563-1581.

[21] G. K. Manacher, 'Production and Stabilization of Real-time Task Schedules,' Journal of the Association for Computing Machinery, Vol. 14, No. 3, July, 1967, pp. 439-465.

[22] M. Lehman, 'A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processors,' Proceedings of the IEEE, Vol. 54, No. 12, December, 1966, pp. 1889-1901.

[23] A. J. Bernstein, 'Analysis of Programs for Parallel Processing,' IEEE Transactions on Electronic Computers, Vol. EC-15, No. 5, October, 1966, pp. 757-763.

[24] B. Randall and C. J. Kuehner, 'Dynamic Storage Allocation Systems,' Communications of the ACM, Vol. 11, No. 5, May, 1966, pp. 297-305.

[25] P. J. Denning, 'Resource Allocation in Multiprocess Computer Systems,' Rept. MAC-TR-50, May, 1968, U. S. Gov't R & D Reports, Vol. 68.

[26] L. A. Belady, 'A Study of Replacement Algorithms for a Virtual Storage Computer,' IBM System Journal, Vol. 5, No. 2, 1966, pp. 78-101.

[27] E. W. Reigel, 'A Study of Parallelism in Computing Systems,' Ph.D. Dissertation, University of Pennsylvania, 1969.

[28] B. Wald, 'Utilization of a Multiprocessor in Command and Control,' Proceedings of the IEEE, Vol. 54, No. 12, December, 1966.

[29] R. E. Lyons and W. Vanderkulk, 'The Use of Triple-Modular Redundancy to Improve Computer Reliability,' IBM Journal, Vol. 6, No. 2, pp. 200-209, 1962.

[30] F. P. Mathur and A. Avizienis, 'Reliability Analysis and Architecture of a Hybrid-Redundant Digital System: Generalized Triple Modular Redundancy with Self-Repair,' AFIPS Spring Joint Computer Conference Proceedings, pp. 375-383, 1970.

[31] A. A. Covo, 'Analysis of Multiprocessor Control Organization with Partial Program Memory Replication,' IEEE Transactions on Computers, Vol. C-23, No. 2, February, 1974, pp. 113-120.

[32] J. M. Kruzberg, 'On the Memory Conflict Problem in Multiprocessor Systems,' IEEE Transactions on Computers, Vol. C-23, No. 3, March, 1974, pp. 286-293.

APPENDIX A

Command and Data Structures, Program Descriptions and Bandwidth Adjustment

Command and Data Format

$␣INIT␣SYS

causes SETUP routine to be called. System parameters are entered immediately after this card as follows:

XXX1 = y1
XXX2 = y2
.
.
.
XXXN = yN

Where XXXN is a 1 to 4 letter mnemonic symbol for the parameter and yN is the value to be assigned to the parameter.

*END

signals the end of parameter initialization.

$␣ENTR␣DIST

causes STAT routine to be called; data must be entered immediately after this card as follows:

NAME = XXXX

where XXXX is some 1 to 4 letter mnemonic to be assigned to the first distribution curve.

DATA = $D_1, D_2, \ldots, D_{41}$.

up to 41 data points may be entered. Data points must be separated by either a blank or a comma and may be real or integer.

The above name and data pairs may be repeated for as many as a total of 25 distributions.

$␣EXIT␣DIST

signals the end of distribution curve entry processing.

$␣ENTR␣BST

causes BULK routine to be called. Data must be entered immediately behind this card as follows:

NAME = *XXX1, XXX2, . . . , XXX7
The asterisk denotes a periodic program, XXXN is a 1 to 4 letter mnemonic
N = 1 indicates job name
N = 2 indicates processing shape curve name
N = 3 through 7 indicates predecessors.
All entries must be separated by commas.

$␣END␣BLOC
signals the end of common statistics job block.

SEED = N
seed value for a random number sequence.

FCTN = XXX1, XXX2, . . . , XXX9
Distribution curve names to be used in generating the job characteristics data.

FMAX = X1, X2, . . . , X9
Real or integer values which designate the maximum allowable values for each parameter.

FMIN = X1, X2, . . . , X9
Real or integer values which designate the minimum allowable values for each parameter.

This card may be followed by more cards of the type between the NAME card following the $ ENTR BST card and FMIN for a different set of job characteristics.

$␣EXIT␣BST
signals the end of bulk storage table processing.

$␣ERAS␣MJDT
causes the arrays JOVF, JM2, MNAME and FLAG to be set to zero by the CLEAR routine.

$␣PRNTb=␣MJDT
causes a print out of time and the information in the J-TABLE by PRNT.

$␣ENTR␣MJDT
causes the ENTR routine to be called. Names of jobs must be entered immediately after this card as follows:

NAME = XXX1, XXX2, . . . , XXXN
where XXXM is the job name of each job to be entered into the J-TABLE.

*END

signals the end of job entry processing.

$bXEQbSIMNbL*N,

where $1 \leq N \leq 4$, causes execution of simulation run. N is the number of time options which the user wishes to specify. These options are as follows:

STIM = Simulation start time. Default value = 0.0
FTIM = Simulation finish time. Default value = 10.0
DELT = Simulation time increment. Default value = 0.01
PDEL = Simulation print time increment. Default value = 0.5

If no values are to be specified then the L*N is omitted and the default values will be used. The optimal values are entered as follows:

PDEL=A
DELT=B
STIM=C
FTIM=D

The order of entry does not matter but there must be the same number of options specified as the value of N.

$bSTOP

This signals the end of the simulation.

Any data card whose first two punched characters are 'C*' is considered to be a comments card and is not processed by the simulator. These cards may be placed anywhere in the data deck before the $bSTOP card.

Data cards may be continued by placing an asterisk in card column 80.

If other runs are to be made with some of the jobs already generated then before the $ STOP card place the following command and data cards:

$bINITbSYS

New Parameter Values

$bERASbMJDT

$bENTRbMJDT

```
  jobs to be entered
*END
$¢XEQ¢SIMN¢L*N
  time parameters
     .
     .
     .
$¢STOP
```

Common Blocks

BLK 1

ELEMENTS:
IASK, IDSH, LIND(33), LINA(33), IBLK, IDLR, IEQL, IDIST, INAME, IDATA, MJDT, IBST, IEXIT, IFLD

HOW USED:
Used in conjunction with inputting and outputting of error messages or for control purposes. These elements do not enter into any calculations.

WHERE USED:
Initialized in BLOCK DATA subroutine and used during job generation and by the group of programs referred to as the job generator. These include MAIN, SETUP, RECHEK, ICARD, BULK, STAT, PRNT, CLEAR, and ENTR.

BLK2

ELEMENTS:
IWRD(100), FLD(50)

HOW USED:
Used to input all data to MAIN and any other routines that call ICARD. IWRD holds alphabetical data and FLD holds numerical data.

WHERE USED:
Initialized by ICARD and used by MAIN, SETUP, ICARD, BULK, STAT, AND ENTR.

BLK3

ELEMENTS:
IDIST(25), F(25,41), NDIST

HOW USED:
IDIST holds names of distributions whose data is contained in F. F holds the data for the distribution curves. NDIST is the number of distributions in F and may not exceed 25.

WHERE USED:
Used by any routine needing access to any of the distributions. These include the following:
MAIN, BULK, PRNT, RECHEK, SHAPE, HLS, NORPRB, and PEX.

BLK4

ELEMENTS:
(NAME(75,8), DATA(75,9),NJOB)
NAME array is sometimes referred to as the bulk storage table. It holds the following information about attributes of each job
NAME(*,1)--periodic/aperiodic
NAME(*,2)--job name
NAME(*,3)--processing curve name
NAME(*,4-8)--predecessors

DATA array holds basic numerical descriptive data for each job in the following order:

EP--external priority

TMR/SIMPLEX--type job

CR--main memory space requirement

I-O--number of main memory accesses for I/O processing

NI--number of main memory accesses for CPU processing

MIX--proportion of short instructions

RR--repetition period(periodic jobs)

IAT--mean inter-arrival period(aperiodic jobs)

DVT--not used

NJOB contains the number of jobs in NAME array.

HOW USED:
Method of use is explained under each entry above

WHERE USED:
used by the following programs in the job generator group:
MAIN, BULK, RECHEK, CLEAR ENTER

BLK5

ELEMENT:
SET

HOW USED:
used to signal condition return codes from subroutine calls in the job generator group.

WHERE USED:
MAIN, SETUP, BULK, STAT, PRNT, RECHEK, CLEAR ENTR

BLK6

ELEMENT:
MNAME(8,64)

HOW USED:
used to hold names of jobs that are in the J-Table (J(27,64) in BLK7)

WHERE USED:
PRNT, RECHEK, CLEAR, and ENTR

BLK7

ELEMENT:
J(27,64) Aliases JM2(27,64), JM(27,64)

HOW USED:
Holds the description and current status of all jobs that are to enter into the current run. The elements of this array are used as follows:

J(1,N)	1-7	Job name (index in MNAME)
	8-15	Index of 1st memory block in MALC for this job.
J(2,N)	1st	Predecessor
J(3,N)	2nd	Predecessor
J(4,N)	3rd	Predecessor

J(5,N)	4^{th}	Predecessor
J(6,N)	1-7	1^{st} successor
	8-14	2^{nd} successor
J(7,N)	1-7	3^{rd} successor
	8-14	4^{th} successor
J(8,N)	Time of last execution	
J(9,N)	Number of memory accesses required in CPU mode.	
J(10,N)	1-7	Instruction mix
	8-15	Internal priority
J(11,N)	1-10	Mean number of memory access requests per simulation step while in CPU mode.
	11-15	Processing shape curve
J(12,N)	1-10	Standard deviation for the number of access requests at each simulation step while in the CPU mode
	11-15	External priority
J(13,N)	1-10	Mean number of memory access requests per simulation step while in I/O mode.
	11-15	not used
J(14,N)	Repetition period if periodic; mean interarrival time if aperiodic	
J(15,N)	Number of memory accesses for I/O operations not including program loading.	
J(16,N)	Number of active simulation steps	
J(17,N)	1-11	main memory space
	12-15	most significant four bits of 3^{rd} memory block index in MALC.
J(18,N)	completion count	

J(19,N)	1-8	Index of 2nd memory block in MALC
	9-12	Least significant four bits of index of 3rd memory block in MALC
J(20,N)	Estimated minimum processing time excluding load time	
J(21,N)	1-10	Standard deviation for number of memory access requests while in the I/O mode
	11-15	not used
J(22,N)	Field overflow indicators	
	Bit 1	Predecessors
	Bit 2	successors
	Bit 3	time of last execution completion
	Bit 4	number of memory accesses in CPU mode
	Bit 5	Repetition period
	Bit 6	memory modules
	Bit 7	main memory space
	Bit 8	memory block index
	Bit 9	completion count
	Bit 10	target time
	Bit 11	next I/O interrupt
	Bits 12-15	not used
J(23,N)	memory preference factor	
J(24,N)	address of first overflow block in overflow area	
J(25,N)	target time	
J(26,N)	next I/O interrupt point	

J(27,N) Job Status

Bit 1	ready
Bit 2	initiated
Bit 3	waiting CPU
Bit 4	waiting IOP
Bit 5	waiting storage space
Bit 6	Holding CPU
Bit 7	Preempted of CPU
Bit 8	mode, 0=CPU, 1=IOP
Bit 9	periodic/aperiodic
Bit 10	resident in main memory
Bit 11	TMR/SIMPLEX, 1=TMR
Bit 12	waiting I/O completion
Bit 13	In Future Events Chain
Bit 14	In Load State

BLK8

ELEMENT:
ICLK alias TIME

HOW USED:
Holds the current simulation step number

WHERE USED:
MAIN, RECHEK, FEC, CEC, HLS, LLS, PEX, NFMAL, MAPREF, and MASGN.

BLK9

ELEMENT:
JOVF(32,32) Alias JOVFL(32,32)

HOW USED:
Holds overflow from fields of the J-TABLE entries

WHERE USED:
CLEAR, and OVFLMG

BLK10

ELEMENT:
FLAG(64)

HOW USED:
Used by RECHEK to determine whether or not a job entry in the J-TABLE has been previously processed by ENTR Command

WHERE USED:
RECHEK and CLEAR

BLK11

ELEMENT:
RTIME

HOW USED:
A time variable used during print out of the J-TABLE.

WHERE USED:
MAIN, and PRINT

BLK12

ELEMENT:
PARM(35)(in MAIN, SETUP, and RECHEK) in other routines the elements of PARM are referred to by the following names: ICECSZ, ICON, IN1, IFECSZ, IPCT, IP1, IP2, IQ1, IQ2, IQ3, IR, IS, ITA, MINBLK, MMCT, MMST, MTOTAL, NI, NIOS, NMODS, NPCL, NPCS, NCPUS, IFEDBK, IRN1, IRN2, IRN3, IRN4, IRN5, IRN6, IRN7, IRN8, IRN9, MMAPF, MMSPF.

HOW USED:
The integer parameters of the system are contained in this block, thus they may be initialized from the input stream at the beginning of a set of runs. Any parameter changes for succeeding runs must be made from the input stream prior to that run. The parameter maintains the value of its last specification. The individual elements are used in the following manner.

ICECSZ--notifies all routines of the current size of the current events array. Must be less than or equal to the size specified (N) in a dimension statement for ICEC (*,N).

ICON--specifies the number of basic clock periods per simulation step.

IN1--a constant used by LLS in comparison with number of I/O accesses to be done at I/O interrupt to help determine whether job can retain CPU.

IFECSZ--notifies all routines of the current size of the future events array. Must be less than or equal to the size specified (N) in a dimension statement for IFEC(N,*).

IPCT--no. of basic clock periods per CPU cycle period.

IP1--a constant used by CEC in comparison with external priority of a newly arrived job to determine whether or not a new schedule should be called for.

IP2--a constant used by LLS in comparison with job internal priority at I/O interrupt points to help determine whether or not job should retain its CPU(s).

IQ1--a constant used by CEC in comparison with number of scheduled entries in the current events chain to determine whether or not to call for a new schedule.

IQ2--a constant used by HLS and LLS in comparison with the number of IOPs already assigned to load operations to determine whether or not more jobs in the load state may be started.

IQ3--a constant used by LLS in comparison with the number of scheduled jobs in the current events chain to determine whether or not a job may retain its CPU upon I/O interrupt initiation.

IR--a constant used by CEC in comparison with the mean bandwidth requirement of a completing job to determine whether a new schedule should be called for.

IS--a command for writing out various arrays for debugging purposes.
BIT#1=1 causes the non zero entries of the allocated and available lists (MALC and MAVL) of main memory storage blocks to be written out upon departing from the memory allocation routines MAPREE or NFMAL.

BIT#2=1 causes PEX to write out the array NAA which contains the current estimate of the available bandwidth from each memory module and MAS which contains the amount of available space in each memory module at each simulation step.

BIT#3=1 causes MAIN to print the information contained in the J-TABLE following completion of each run.

Other bits are presently unused.

ITA--holds memory access time x 100 in basic clock periods.

MINBLK--holds the smallest memory increment that may be allocated to a job. Also used by CEC and HLS to determine amount of time required to load a job.

MMCT--the number of basic clock periods required for one memory module cycle.

MMST--a constant used by HLS in comparison with the amount of space occupied by a nonresident job at completion time to help determine whether or not a new schedule should be called for.

MTOTAL--total memory size.

NI--a constant used by LLS to control the maximum number of jobs that may be initiated.

NIOS--the number of IOPs in the system, must be less than or equal to the maximum number specified (N) in a dimension statement for the IOP status array IPROS(N).

NMOD--the number of memory modules in the system.

NPCL--the number of CPU cycle periods required for execution of a long instruction.

NPCS--the number of CPU cycle periods required for execution of a short instruction.

NCPUS--the number of CPUs in the system, must be less than or equal to the maximum number specified (N) in a dimension statement for the CPU status array ICPU(N).

IFEDBK--indicates which memory allocator is to be used for current run. IFEDBK=1 indicates MAPREF is to be used. IFEDBK=0 indicates NFMAL is to be used.

IRN1--used by PEX as a seed for calculating a random number sequence for determining the number of memory accesses to request for each active job at each simulation step.

IRN2--used by HLS as a seed for calculating a random number sequence to determine the mean of the number of memory accesses to be made for each job during its processing period.

IRN3--used by HLS in calculation of target times for aperiodic jobs during the initial preprocessing phase and following job completion.

IRN4-- not used

IRN5--used by PEX as the number of simulation periods over which the available bandwidth of each memory module is estimated.

IRN6--used by HLS in determining average memory access rate while in I/O mode and in determining minimum processing time.

IRN7--represents speed of IOP's relative to that of CPUs.

IRN8--used by HLS to determine residency of jobs. Represents cost of memory relative to cost of processor time.

IRN9--used by RECHEK in generating some of the characteristics of each job in the J-TABLE.

MMAPF--used by MAPREF in calculating memory preference factor based on space considerations.

MMSPF--used by MAPREF in calculating memory preference factor based on space considerations.

WHERE USED:
MAIN, SETUP, RECHEK, CKCPU, CKIOS, FEC, CEC, HLS, LLS, PEX, NFMAL, MAPREF, MASGN, and MEMRLS.

BLK13

ELEMENTS:
RPAR(11) (in MAIN, and SETUP) in other routines the elements of RPAR are referred to by the following names:
RP01, RP02, RP03, RP04, RP05, RP06, RP07, RP08, RP09, RP10, and RP11.

HOW USED:
The real parameters of the system are contained in this block so they like the integer parameters may be initialized from

the input stream at the beginning of a set of runs. Any parameter changes for succeeding runs must be made from the input stream prior to that run. The parameter maintains the value of its least specification. The individual elements are used in the following manner:

RP01--used by MAIN in testing level of short term utilizations for triggering new schedules.

RP02--used by MAPREF to help prevent memory space lockout.

RP03--used by CEC as a weighting factor for external priority in the internal priority equation.

RP04--used by CEC as a weighting factor on amount of time to target time in internal priority equation.

RP05--used by CEC as a weighting factor on time in queue in calculation of internal priority equation.

RP06--used by HLS to determine the number of minimum processing periods before target time that each should be transferred from the Future Events Chain to the Current Events Array. Also used by CEC in calculating internal priority.

RP07--used by HLS in determining residency for each job.

RP08--used by CEC and by HLS to determine what proportion of each job's memory space is initialized at load time.

RP09--used by HLS to determine the deviation in memory access requests while in CPU mode.

RP10--Access request distribution, 1.0 = random, 2.0 = sequential.

RP11--not used.

WHERE USED:
MAIN, SETUP, CEC, HLS, LLS, PEX, and MAPREF.

BLK14

ELEMENTS:
IFLAG(47), ITAG(46)

HOW USED:
IFLAG is used in determining whether or not the system parameters have all been initialized. ITAG is used during pre-execution printout to determine whether or not the variables were

initialized immediately before execution of a run or remain at a previously defined value.

WHERE USED:
MAIN and SETUP

BLK15

ELEMENT:
KNAME(46)

HOW USED:
Used in initializing the system parameters. Each element is a 1 to 4 letter mnemonic for each variable in BLK12 and BLK13. The order in which the mnemonics are listed in BLK15 should correspond to the order in which the full mnemonics are listed in BLK12 and BLK13.

WHERE USED:
MAIN and SETUP.

BLK16

ELEMENTS:
IBWCTR(24), ICEC(5,40), ICECS, ICPU(10), IPASS(20), IFECS, IFEC(40,4), IPROS(16), ISAVE(3,3), IVALOV(10), MALC(265,5), MALCS, MAS(24), MAVL(128,5), MAVLS, MODNM(24), MTP1(24), NA(24), NAA(24), NAB, NAG, NAML(40), NBLK(128), NFB(3), NJWM, NSCHED, NREQ(40,24), NTP(24), NTR(24), NUCA(24).

HOW USED:
IBWCTR--counters for the number of simulation steps since an allocation or release of memory from each memory module.

ICEC--the current events array. Serves as the job queue for the system. The information in each of the five entries of each position of the ICEC array is as follows:

ICEC(1,N) = Internal priority of job occupying position N.

ICEC(2,N) = J-Table index of job occupying position N.

ICEC(3,N) = time of entry into the array.

ICEC(4,N) = An indicator of whether the job occupying position N is currently active.

ICEC(5,N) = A pointer to the next entry of the Current Events Chain.

ICECS--a pointer to the first entry of the Current Events Chain.

ICPU--the status table for the system's CPUs.

IPASS--an array which holds the current status of the system.

IPASS(1)--the number of entries in the ICEC array.

IPASS(2)--the number of entries in the Current Events Chain.

IPASS(3)--the number active jobs in the system.

IPASS(4)--the number of IOPs that have been initiated.

IPASS(5)--the number of IOPs that are assigned to jobs in the Load state.

IPASS(6)--the time at which the last schedule occurred.

IPASS(7)--the number of idle CPUs.

IPASS(8)--used by MAIN and FEC to pass the time of the next event in the Future Events Chain to MAIN.

IPASS(9)--the number of free IOPs.

IPASS(10)--the amount of main memory space currently free.

IPASS(11)--the number of jobs scheduled and waiting for storage space.

IPASS(12-20)--not used.

IFECS--a pointer to the first entry of the Future Events Chain, -1 indicates an empty chain.

IFEC--the Future Events array. The information contained in each of the four positions of an entry of this array are as follows:

IFEC(1,N) = Event time
IFEC(2,N) = type event
IFEC(3,N) = operand of event
IFEC(4,N) = pointer to next entry of chain.

IPROS--the current status table for the IOPs of the system.

ISAVE--used by MAPREF and NFMAL to hold information used in making allocations of storage to TMR jobs.

IVALOV--used in passing information to and from the overflow area for the J-TABLE.

MALC--the array used for the allocated storage list. The information contained in each of the five entries of each position of this array are as follows:

MALC(N,1) = pointer to next entry in the chain

MALC(N,2) = pointer to preceeding entry in the chain

MALC(N,3) = starting address for block described by this position.

MALC(N,4) = pointer to next entry in the available list.

MALC(N,5) = length of block described by this position.

MALCS--pointer to first entry of the allocated chain.

MAS--holds the amount of available storage space in each memory module.

MAVL--the array used for the available storage list. The information contained in each of the five entries of each position of this array are as follows:

MAVL(N,1) = pointer to next entry in the chain.

MAVL(N,2) = pointer to preceeding entry in the chain.

MAVL(N,3) = starting address for block described by this position.

MAVL(N,4) = pointer to next entry in the allocated list.

MAVL(N,5) = length of block described by this position.

MAVLS--pointer to the first entry of the available chain.

MODNM--holds information concerning modules in which blocks of storage are contained.

MTP1--holds total priority of all active jobs having storage space in each of the memory modules that request more memory accesses than their relative priority dictates that they may obtain.

NA--holds the number of jobs having space in each memory module.

NAA--holds the current estimate of the available bandwidth of the available space in each memory module.

NAB--the number of allocated blocks of storage.

NAG--used by PEX in granting accesses to memory to each active job.

NAML--used by MAPREF for holding the list of jobs that are waiting for memory. Also used by CEC in reordering the Current Events Chain.

NBLK--used by MAPREF for holding the list of available blocks of memory.

NFB--the number of free blocks of storage. Used by NFMAL.

NJWM--the number of jobs that are waiting for storage space.

NSCHED--used for calling for a new schedule.

NREQ--holds the number of memory access requests from each active job to each memory module at each step of execution processing by PEX.

NTP--used by PEX to hold the total priority of all the active jobs having space in each memory module.

NTR--used by PEX to hold the total number of memory access request to each memory module at each step of execution processing.

NUCA--used by PEX to hold the sum of the differences in the number of memory access dictated by relative priorities and the number actually requested. Thus, the number of unclaimed accesses.

WHERE USED:
MAIN, SETUP, CKCPU, CKIOS, LAST3, FEC CEC, HLS, LLS, PEX, NFMAL, MAPREF, MASGN, and MEMRLS

BLK17

ELEMENTS:
ISTCNT, ISTAT(6,400), IUTL(6,200)

HOW USED:
ISTCNT holds the count of the number of entries made in the ISTAT array during each run.

ISTAT holds the statistics for each completed job. Its entries have the following meaning:

ISTAT(1,N) = job's index in J-TABLE

ISTAT(2,N) = time of entry into the Current Events Array

ISTAT(3,N) = time completed

ISTAT(4,N) = estimated minimum processing time

ISTAT(5,N) = target time

ISTAT(6,N) = actual number of active simulation steps for the job.

IUTL holds the statistics for utilization of the system's resources at selected step intervals. The entries hold the following information:

IUTL(1,N)--time

IUTL(2,N)--CPU utilization

IUTL(3,N)--IOP utilization

IUTL(4,N)--memory bandwidth utilization

IUTL(5,N)--memory space utilization

IUTL(6,N)--mean job queue size.

WHERE USED:
MAIN, SETUP and PEX

BLK18

ELEMENTS:
RUTL(15) in setup, other routines refer to the individual elements of this array by the following names:
AN, CUU, PUU, BWU, SPU, BCUU, BPUU, BBWU, BSPU, ACUU, APUU, ABWU, ASPU, BQSIZ, AQSIZ

HOW USED:
AN is a time counter
CUU holds the CPU utilization for the past step
PUU holds the IOP utilization for the past step
BWU holds the memory bandwidth utilization for the past step
SPU holds the memory space utilization for the past step
BCUU holds the short term CPU utilization
BPUU holds the short term memory bandwidth utilization
BSPU holds the short term memory space utilization
ACUU holds the long term CPU utilization
APUU holds the long term IOP utilization
ABWU holds the long term memory bandwidth utilization
ASPU holds the long term memory space utilization
BQSIZ holds the short term average job queue size
AQSIZ holds the long term average job queue size

WHERE USED:
MAIN, SETUP, and PEX.

A Brief Description of Each Routine

MAIN

This program interprets the commands and data arriving from the input stream and in general directs the generation of the set of jobs that the system is to execute during a set of simulation runs. It also determines when to initiate new schedules during the course of a simulation run, advances the simulation clock and prints out statistics concerning utilization of system resources and service to individual jobs.

SETUP

The routine SETUP initiates the system parameters in common blocks, BLK12 and BLK13 as well as certain variables in common block BLK16.

ICARD

The ICARD routine reads command and data cards one at a time and checks for syntax errors on these cards. It has provisions for handling continuation cards. It places each character that it reads into the arrays IWRD and FLD.

BULK

This routine causes the job names and predecessors to be placed in the NAME array. It then generates characteristics for the jobs in the NAME array using random distributions residing in the F array and the limit variables specified on the FMAX and FMIN cards for this set of jobs.

STAT

The STAT routine causes information concerning probability distributions and processing shape curves to be read into the F array.

ENTR

The ENTR routine causes a specified selection of job names to be entered into the MNAME array provided these job descriptions are in the NAME array.

PRNT

The PRNT routine prints out the information from the J-TABLE at selected intervals of time or upon command from some other routine.

CLEAR

The CLEAR routine sets all entries in the J-TABLE and overflow area to zero upon command from the input stream.

RECHEK

The RECHEK routine transfers the data residing in the DATA and NAME arrays that is associated with elements of the MNAME array into the J-TABLE upon commands from the input stream.

UNMIX

The UNMIX routine separates any selection of consecutive bits from a word. It is used in unpacking data.

PACK

The PACK routine is used to pack data into a single word.

RANDN

The RANDN routine is used to generate normally distributed random number sequences.

RANDU

The RANDU routine is used to generate uniformly distributed random number sequences.

SHAPE

The SHAPE routine is used to update the next interrupt point of for each job by using the processing shape curves.

CKCPU

The CKCPU routine is used to update the status of CPUs and to search the CPU status table for idle CPUs.

CKIOS

The CKIOS routine is used to update the status of IOP and to search the IOP status table for idle IOPs.

LAST3

LAST3 finds the lowest priority jobs with CPU(s) for preempting.

BITWRT

BITWRT is used by the PRINT routine for printing job status words out in binary form.

FEC

The FEC routine manages the Future Events Chain. It moves events into or out of this chain upon command from MAIN and HLS.

CEC

The CEC routine manages the Current Events Chain. It places entries into this chain, removes entries from the chain and orders the entries of the chain according to priority. It also calculates the priorities of the entries in the chain and sets the load status bits in the J-Table for new jobs entering the current events array that are not presently loaded into the memory.

HLS

The HLS routine has three major functions as follows:

(1) HLS performs preprocessing on the set of jobs of a simulation run at the beginning of the run. During this preprocessing it determines the minimum processing time for each job, determines which jobs should be resident in the main memory based on load times, mean interarrival times and relative costs of IOP time and memory space. It also calculates an initial target time for all jobs with no immediate predecessors.

(2) Following a new schedule HLS makes an initial assignment of processors to each active job.

(3) Following job completion the HLS routine calculates a new target time and determines whether the job goes into the Current Events array, Future Events Chain or neither depending on value of next target time, minimum processing time and precedence relations. It also checks all immediate successor jobs to see if all their other predecessors have been run recently enough that their output data is still valid.

LLS

The LLS routine switches jobs under the following three conditions:

(1) At I/O interrupt initiation

(2) At I/O interrupt completion

(3) At job completion

This routine will preempt jobs of lower priority of their CPUs and reassign these CPUs to higher priority jobs. A considerable portion of this routine is concerned with problems associated with the simultaneous execution of TMR and SIMPLEX jobs.

PEX

The PEX routine performs execution processing, keeps up with the amount of processing done on each job, calls LLS at I/O interrupt points, estimates the available bandwidth in each memory module and tallies system utilization and processor time to individual jobs.

NORPRB

This routine calculates the probability that a sample from a normal distribution with mean μ_1 and standard deviation σ_1 will exceed a sample from a normal distribution with mean μ_2 and standard deviation σ_2.

NFMAL

This routine is the nonfeedback memory allocator. It makes allocations to jobs without knowledge of bandwidth requirements. It uses the optimum-fit algorithm. Allocations to TMR jobs are made from three disjoint sets of memory modules.

MAPREF

This routine is the feedback memory allocator. It uses an adaptation of the best fit algorithm. Memory assignments are made on the basis of best fit in space and bandwidth, however a portion of other priority factors such as time in queue may enter into the selection in order to avoid memory lockout. This routine is influenced to a considerable extent by the requirements for allocations to TMR jobs.

MASGN

This routine is called by MAPREF and NFMAL for updating the lists of available and allocated storage at each memory allocation. It also helps to tally the amount of available space and bandwidth in each memory module.

MEMRLS

This routine is called by HLS for updating the lists of available and allocated storage each time it is desired to release a block of storage.

OVFLMG

This routine manages the overflow area from the J-TABLE. It will place, retrieve, or remove information from this area.

Adjusting the Mean Bandwidth Requirements

The model provides for adjustment of the mean bandwidth requirements from any number of subsets of jobs in the set of a run by means of the mix parameter. Specifically the mean number of access requests per simulation step for CPU processing is given by

MNARC = ICON/(ITA + IPCT * FMIX)

where FMIX = NPCS * INM + NPCL * (1 - INM)

ICON is the number of basic clock periods per simulation step. ITA is the number of basic clock periods for memory access, IPCT is the number of CPU cycle periods per basic clock period, NPCS = number of CPU cycle periods per short instruction, INM is the proportion of short instructions and NPCL is the number of CPU cycles per long instruction. Similarly the number of accesses available from each memory module per simulation step is ICON/MMCT. Thus let A represent the ratio of mean bandwidth requirement per job to that of the bandwidth available from a memory module. We then have

MNARC = A * ICON/MMCT

ICON/(ITA + IPCT * FMIX) = A * ICON/MMCT

Rearranging we obtain

$$INM = \frac{MMCT/A - ITA - IPCT * NPCL}{IPCT * (NPCS - NPCL)}$$

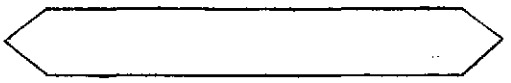
where $0 \leq INM \leq 1$

For a given set of values for ITA, IPCT, NPCS, NPCL and MMCT (chosen for the particular run) there are bounds on the value of A. Specifically as INM ranges from 0 to 1 A ranges from MMCT/(ITA + IPCT * NPCL) to MMCT/(ITA + IPCT * NPCS).

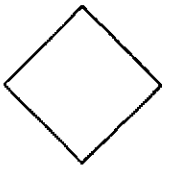
APPENDIX B

Program Flow Charts

NOTE: In these charts the symbol



has been used as the decision element
in place of the usual symbol



to reduce the amount of paper required.

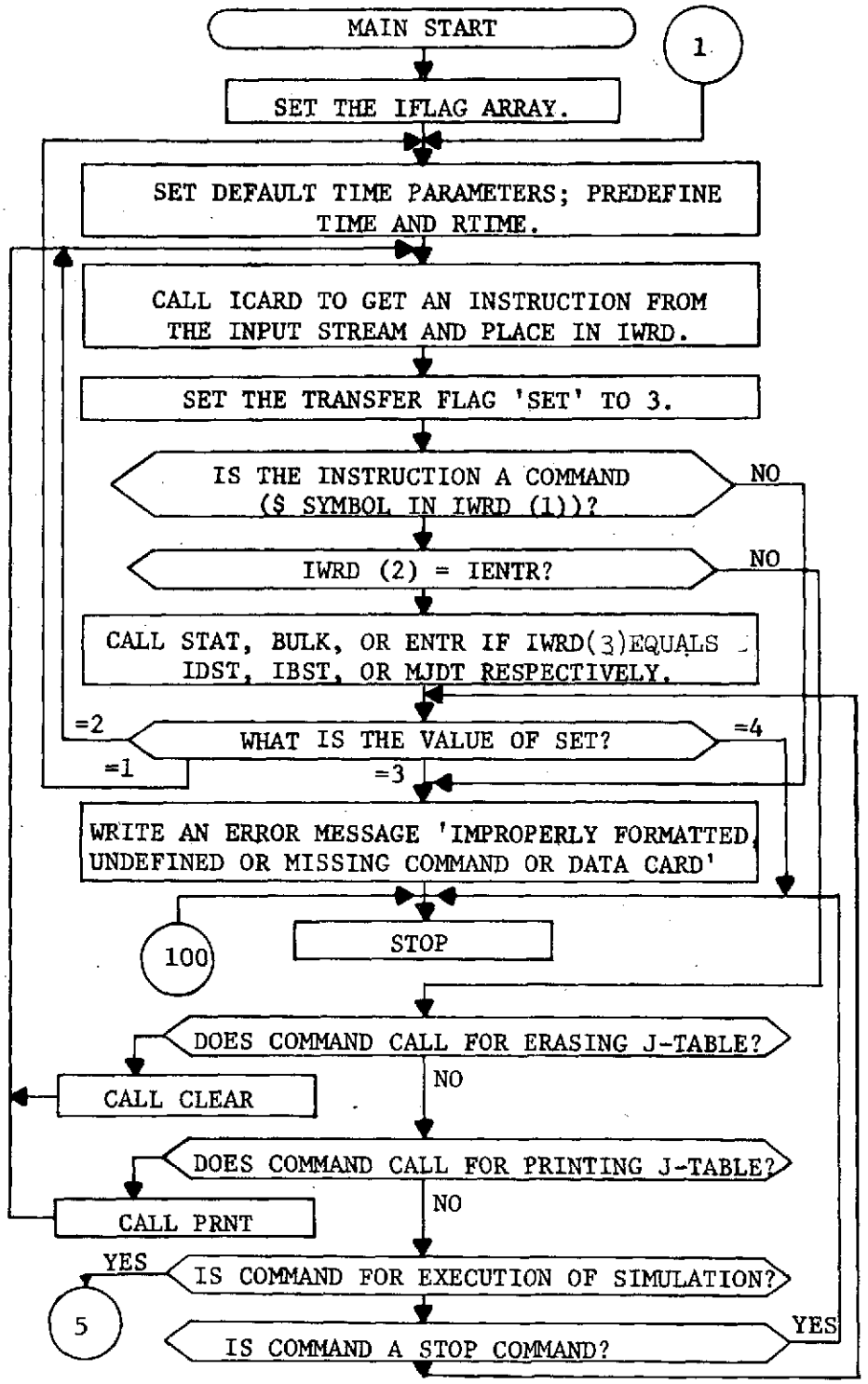


Fig. B-1 MAIN PROGRAM (sheet 1 of 3)

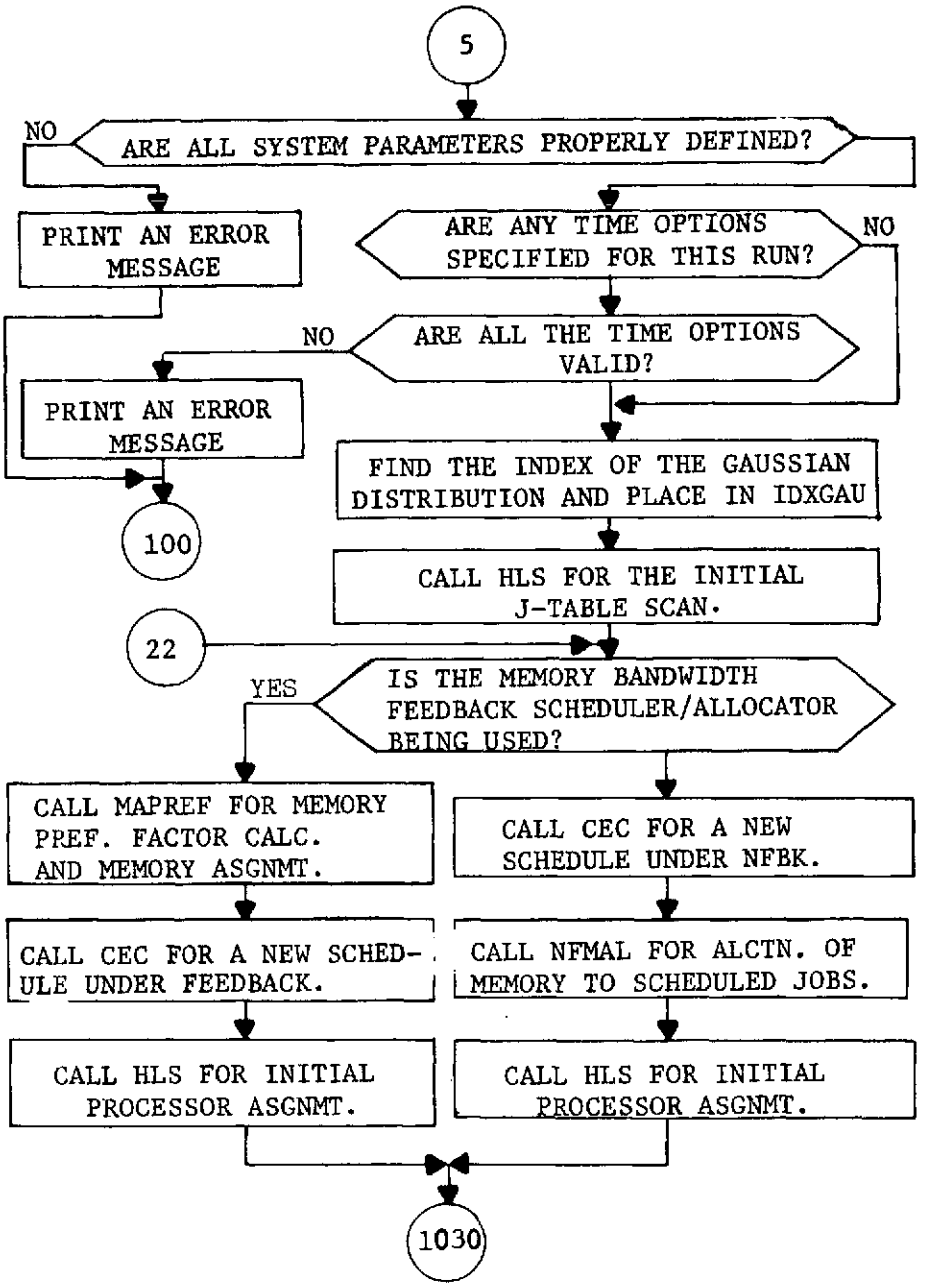


Fig. B-1 MAIN PROGRAM (sheet 2 of 3)

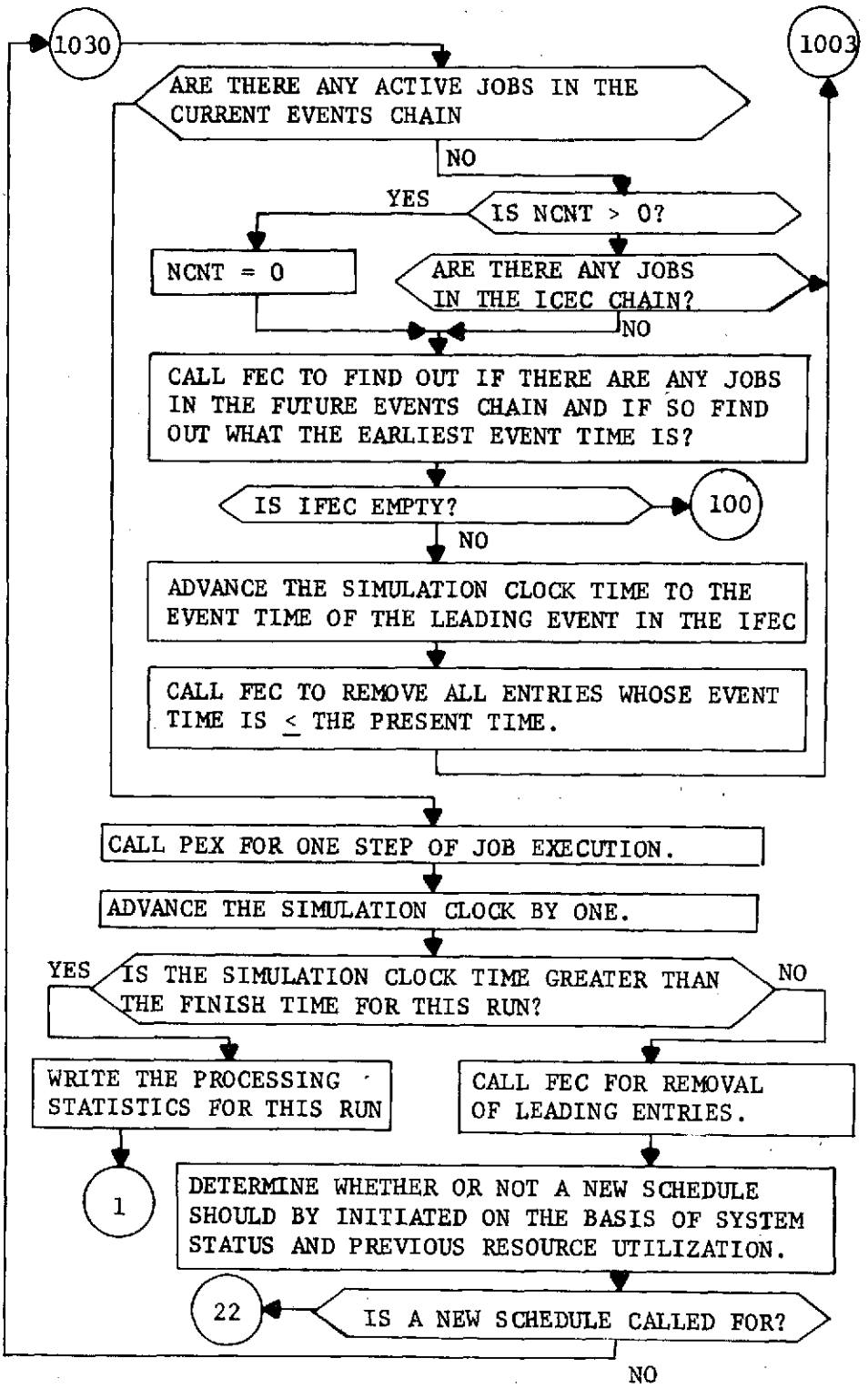


Fig. B-1 MAIN PROGRAM (sheet 3 of 3)

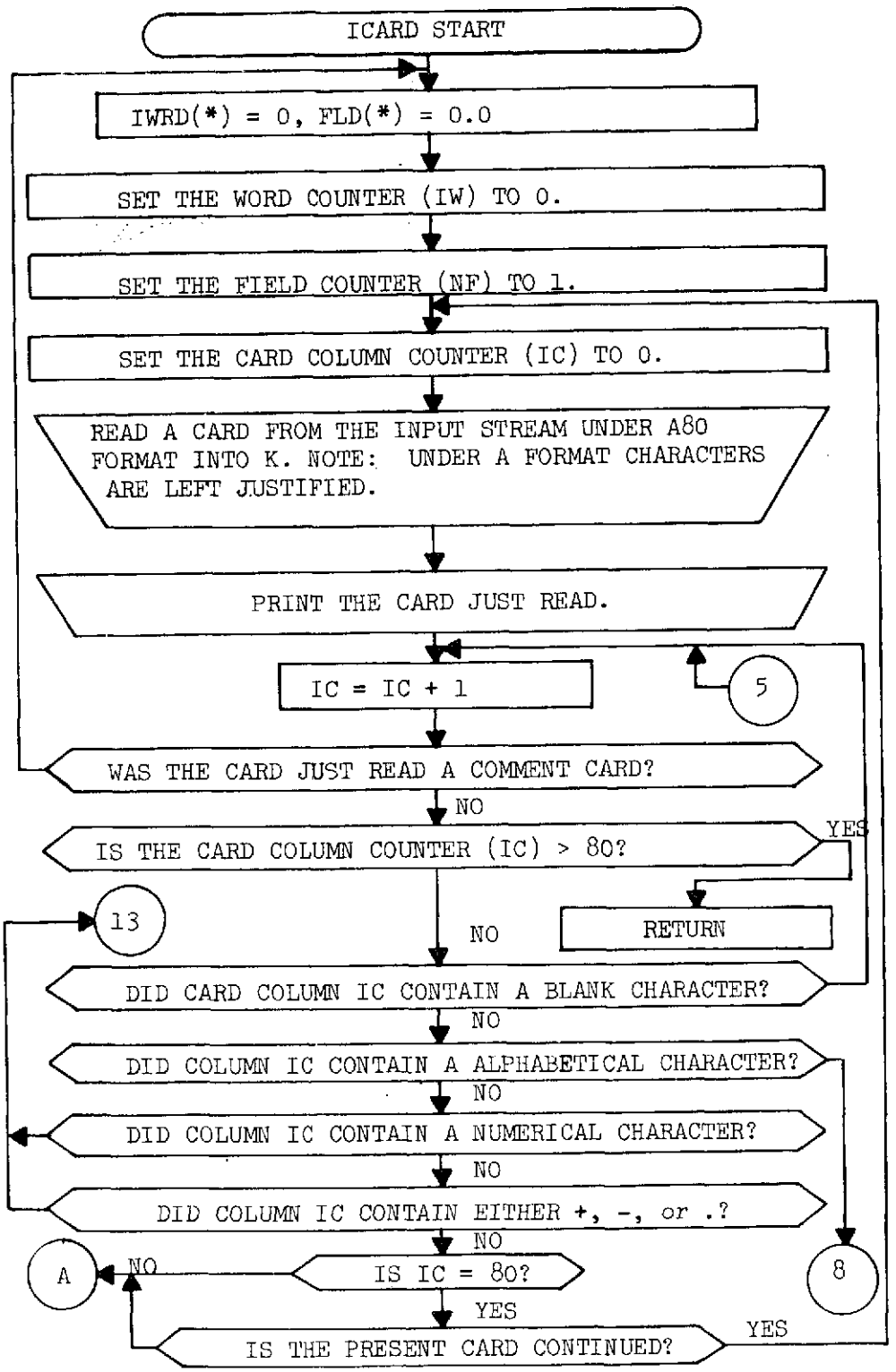


Fig. B-2 ICARD subroutine (sheet 1 of 4)

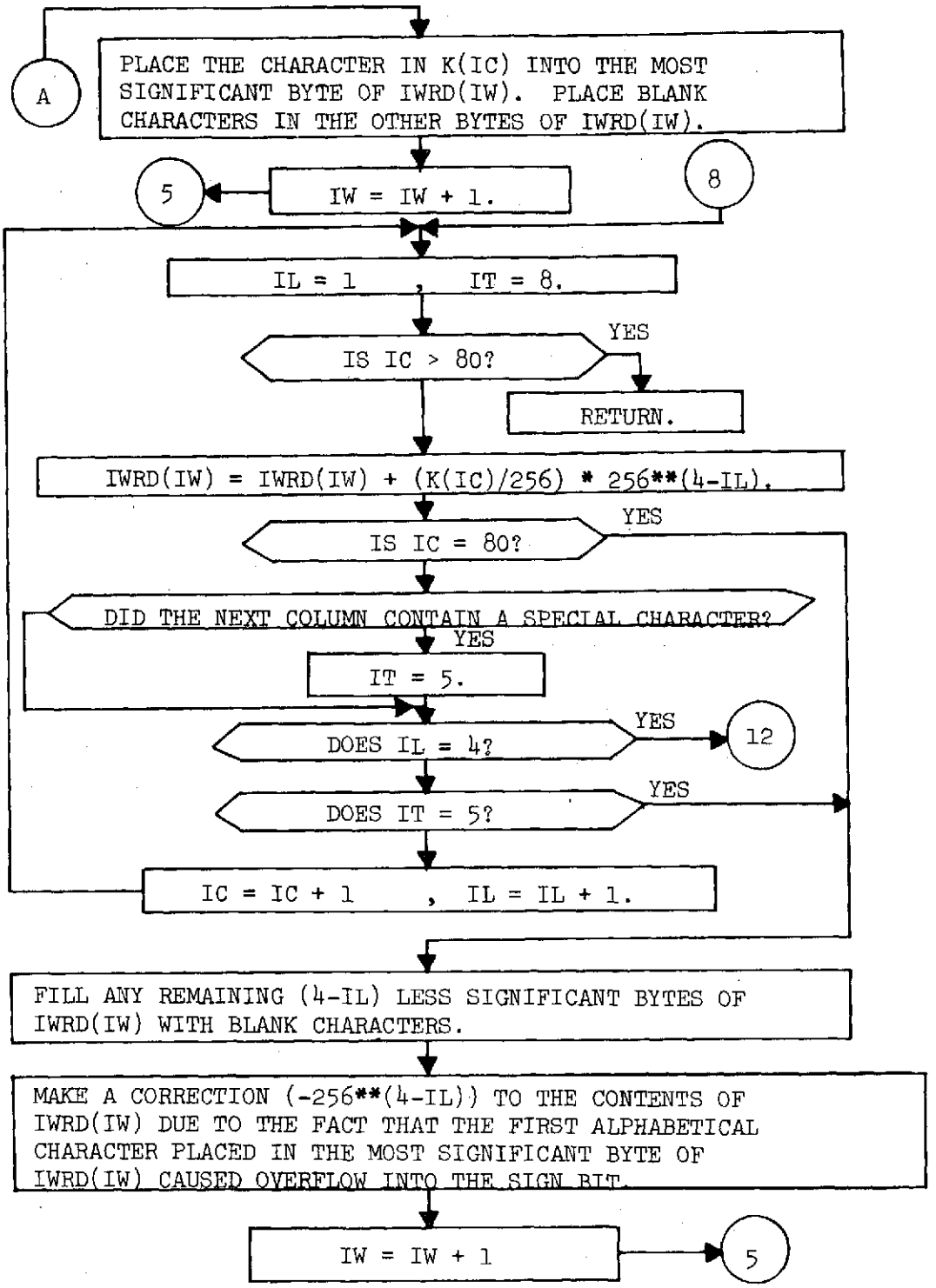


Fig. B-2 ICARD subroutine (sheet 2 of 4)

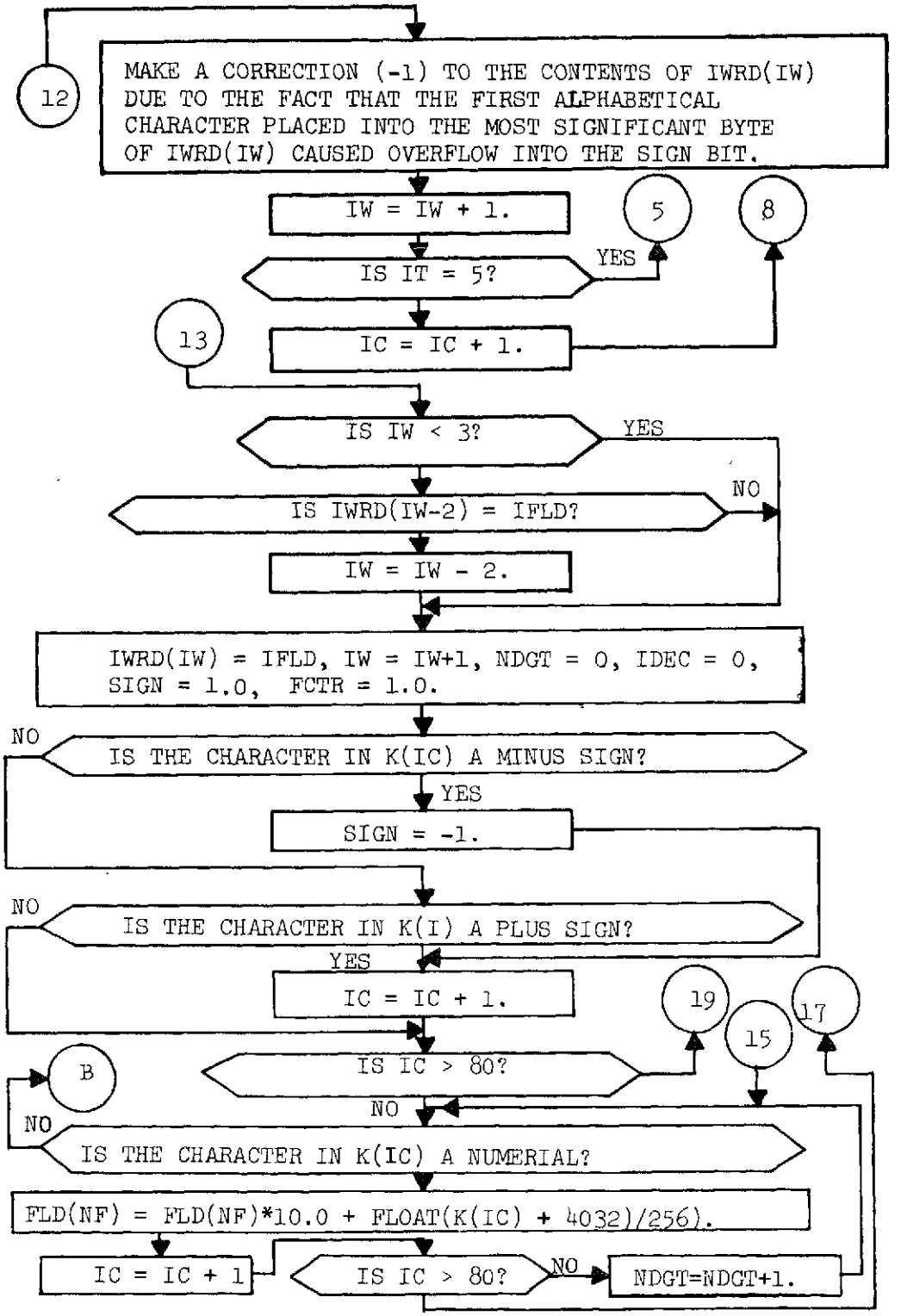


Fig. B-2 ICARD subroutine (sheet 3 of 4)

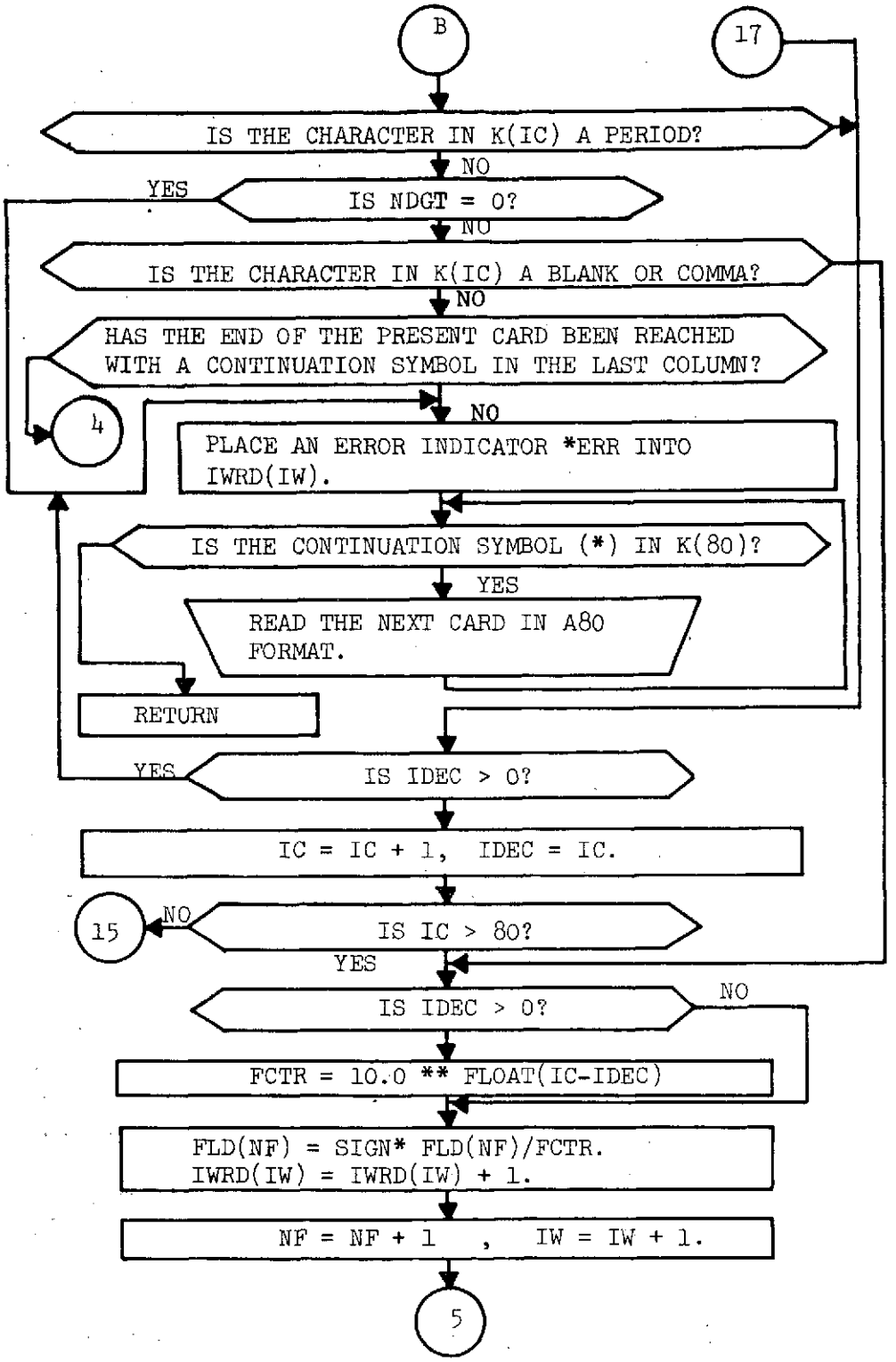


Fig. B-2 ICARD subroutine (sheet 4 of 4)

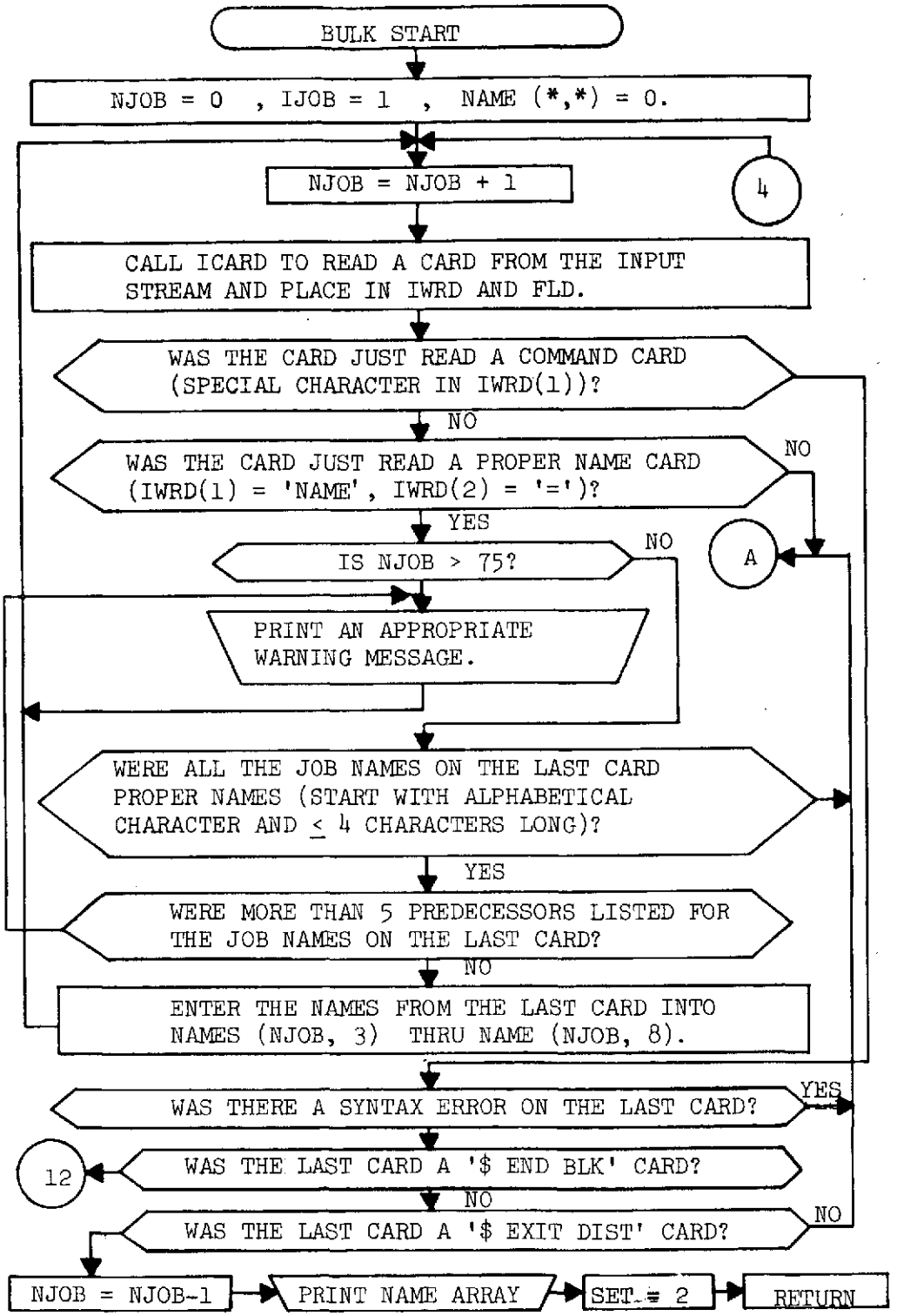


Fig. B-3 BULK subroutine (sheet 1 of 3)

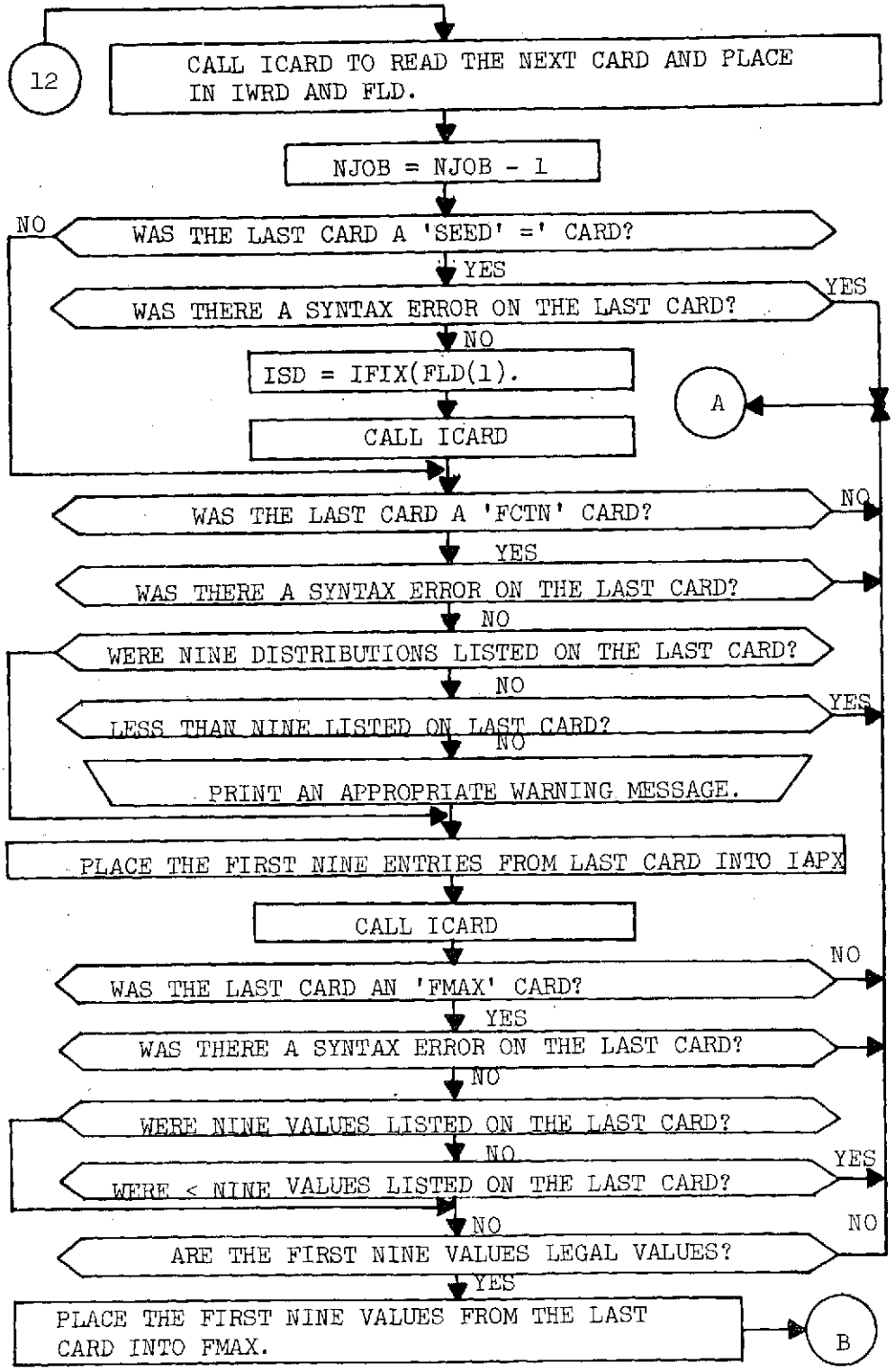


Fig. B-3 BULK subroutine (sheet 2 of 3)

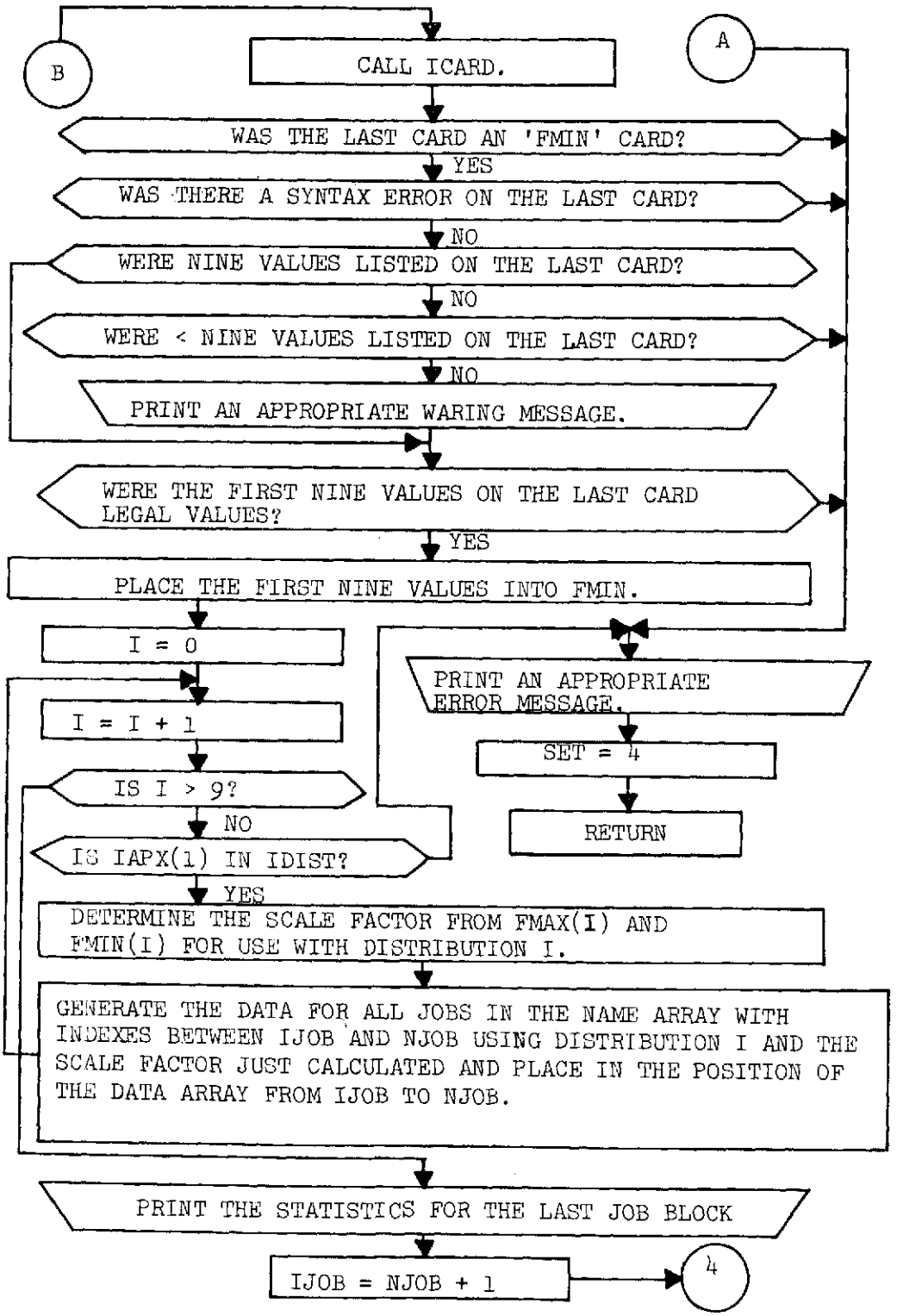


Fig. B-3 BULK subroutine (sheet 3 of 3)

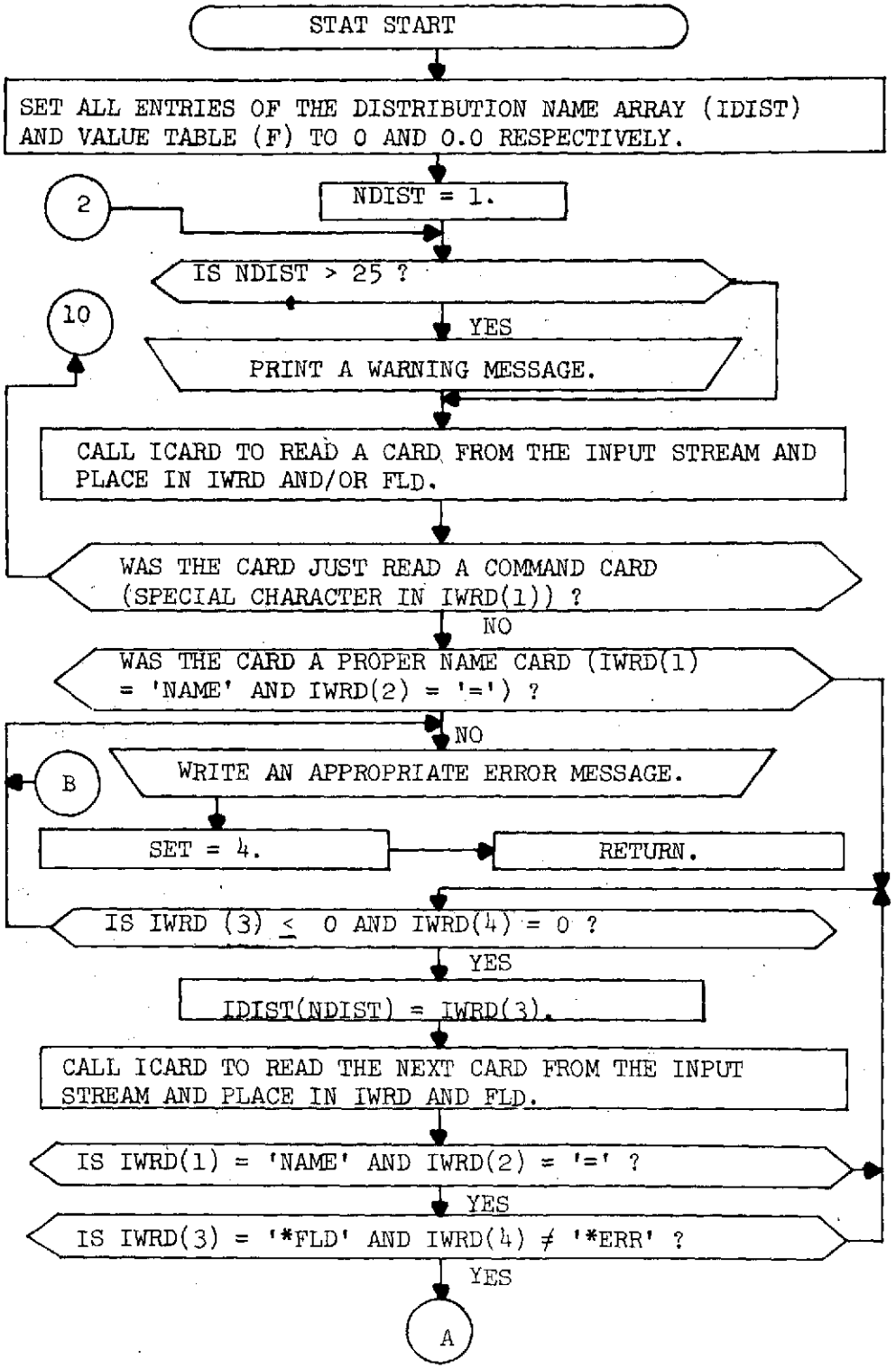


Fig. B-4 STAT subroutine (sheet 1 of 2)

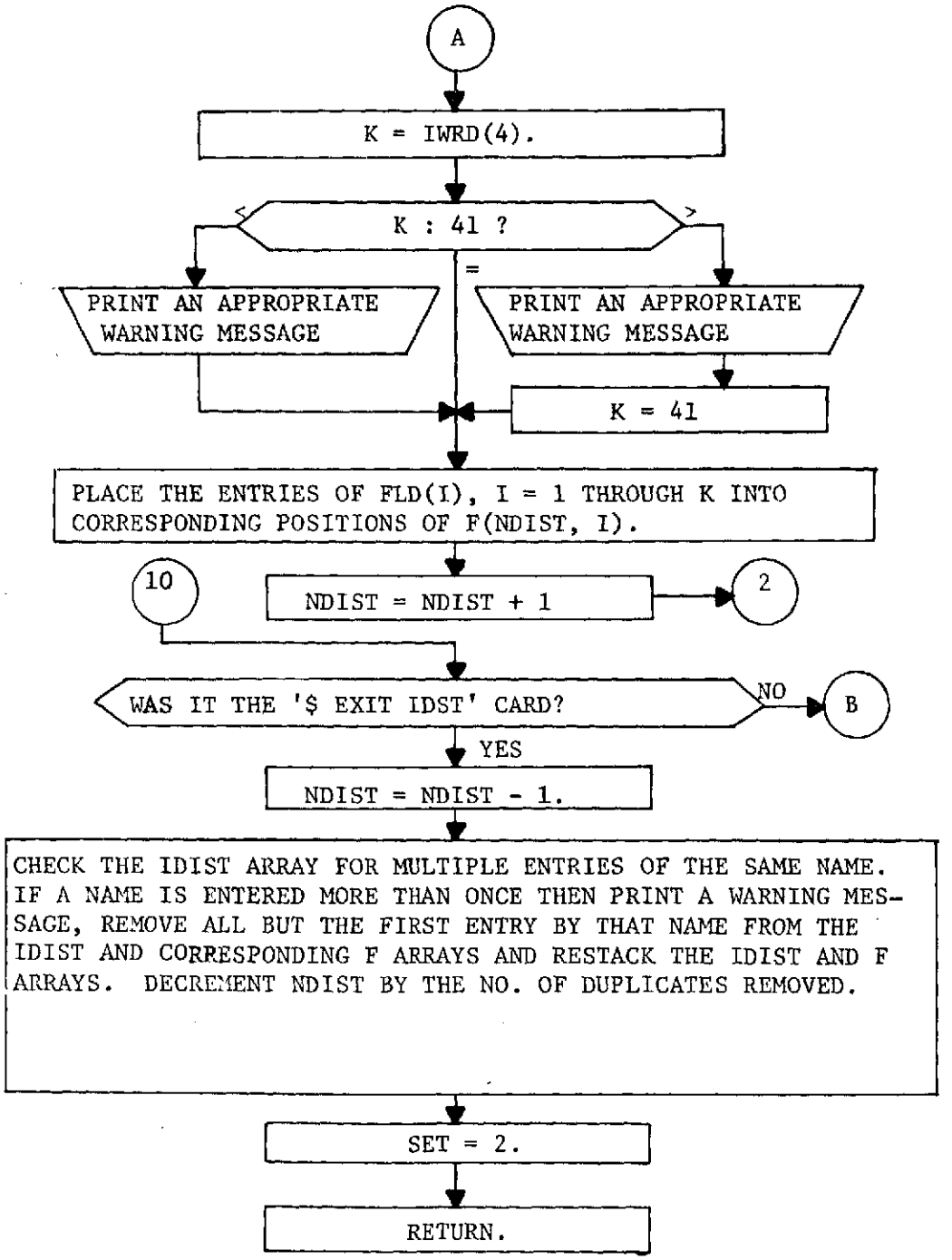


Fig. B-4 STAT Subroutine (Sheet 2 of 2).

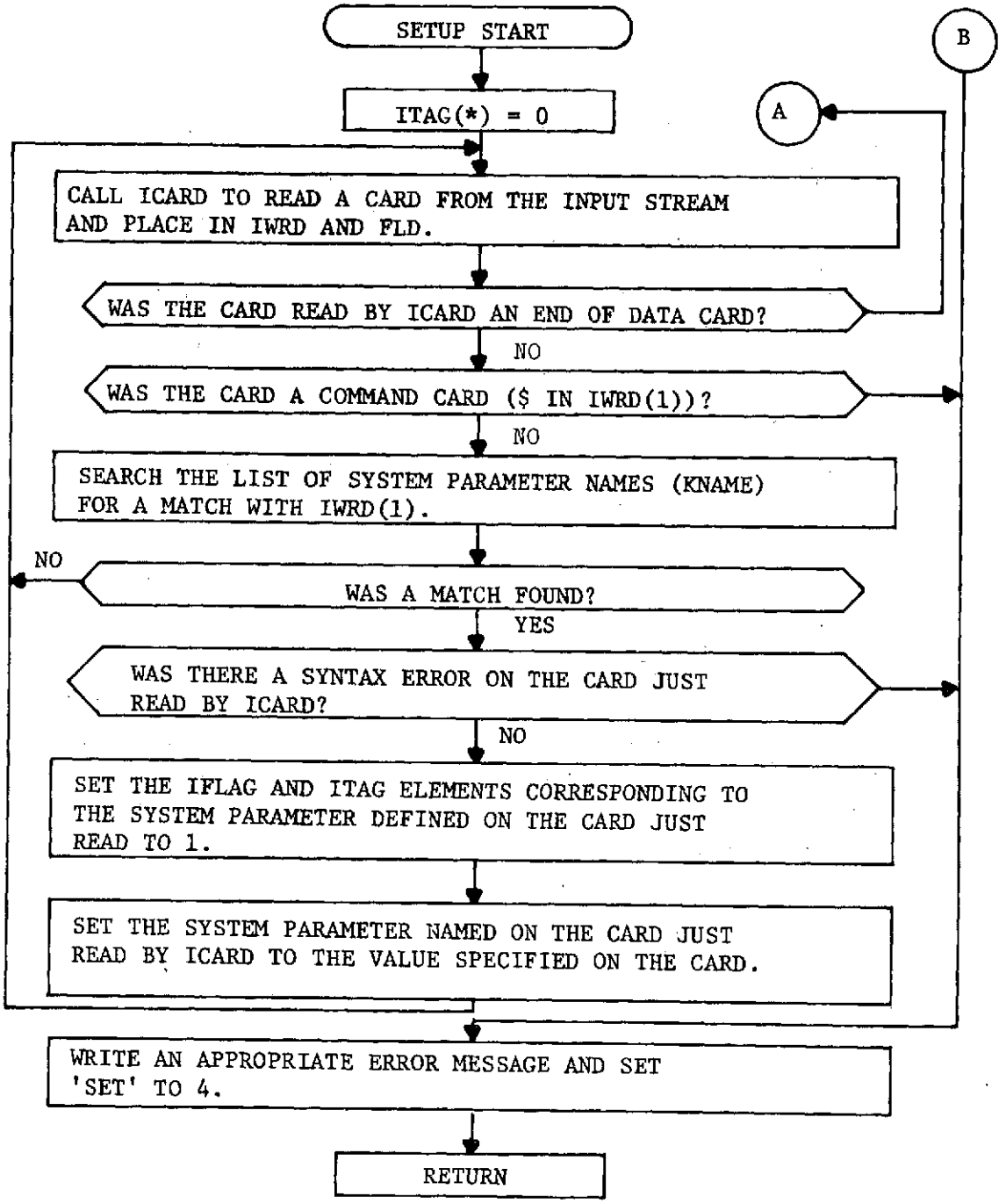


Fig. B-5 SETUP subroutine (sheet 1 of 2)

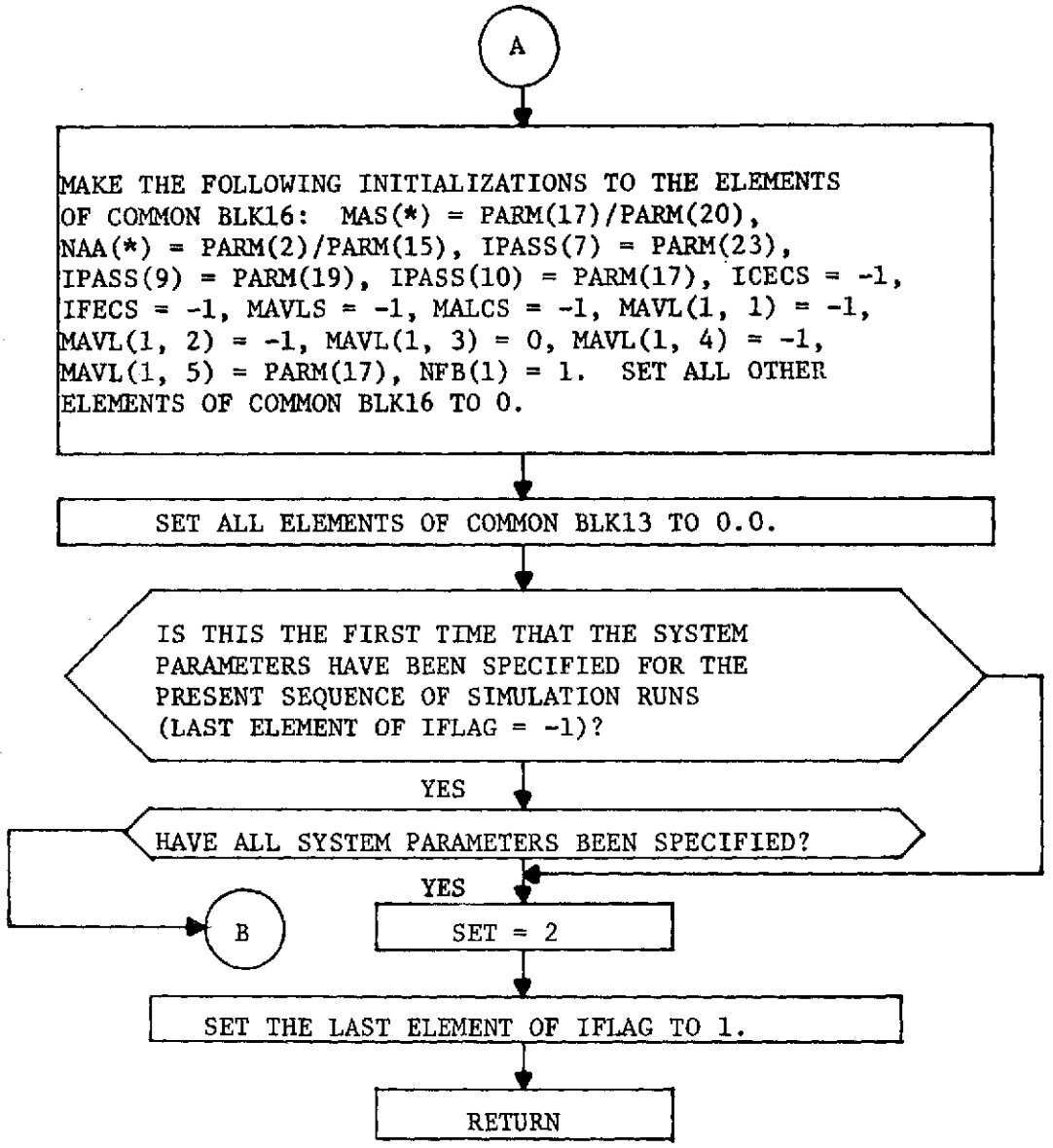


Fig. B-5 SETUP subroutine (sheet 2 of 2)

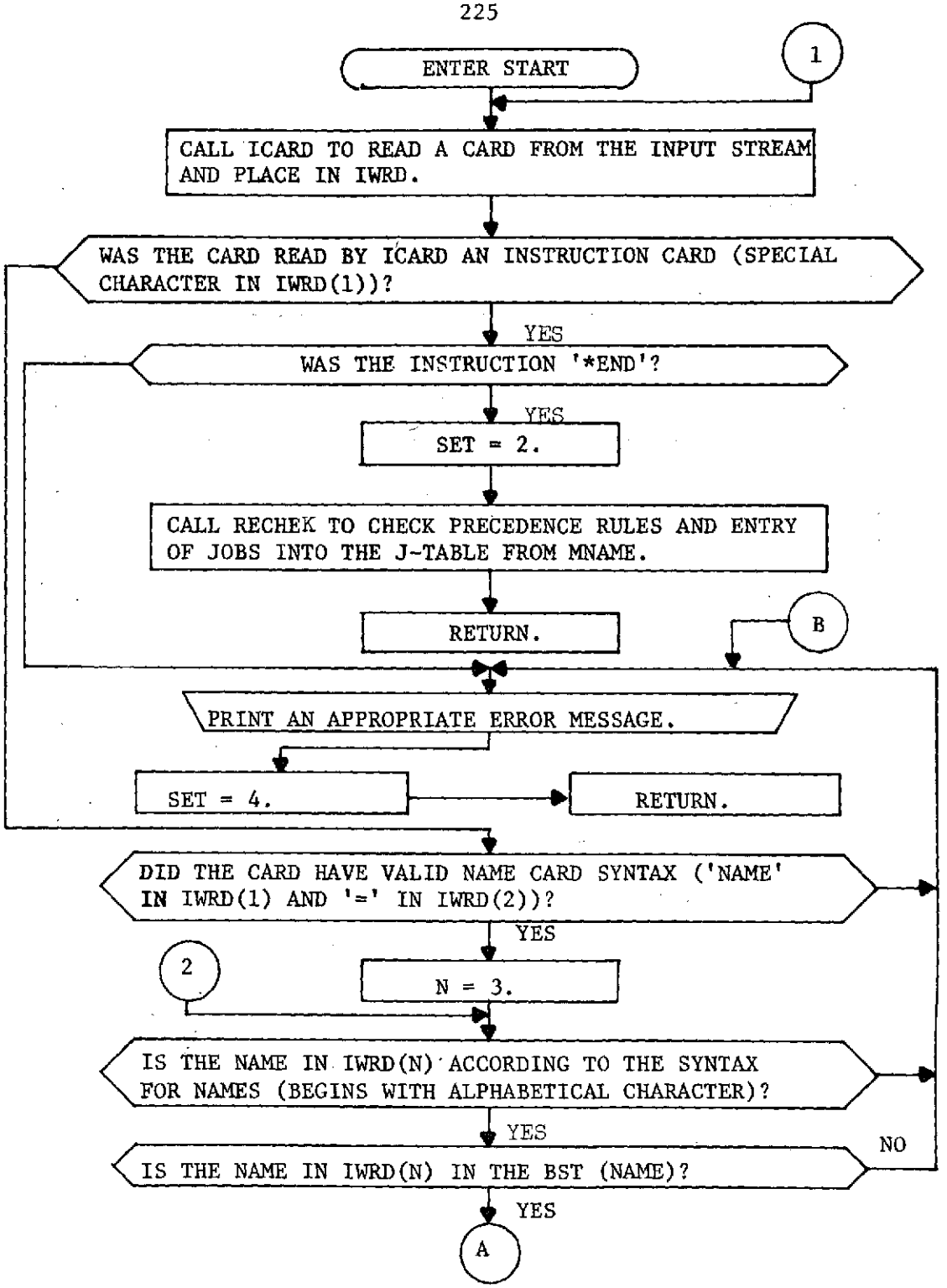


Fig. B-6 ENTER subroutine (sheet 1 of 2)

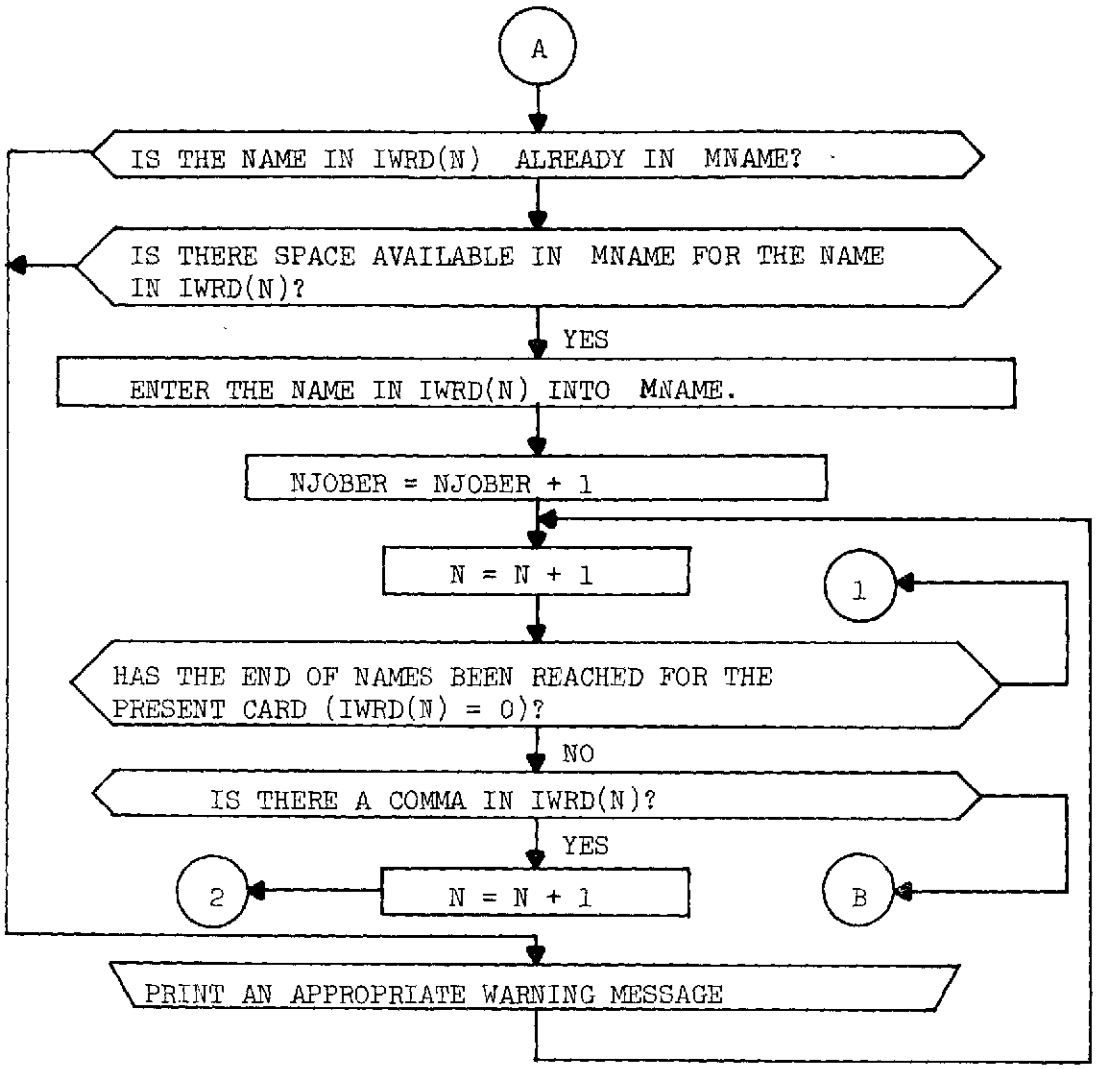


Fig. B-6 ENTER subroutine (sheet 2 of 2)

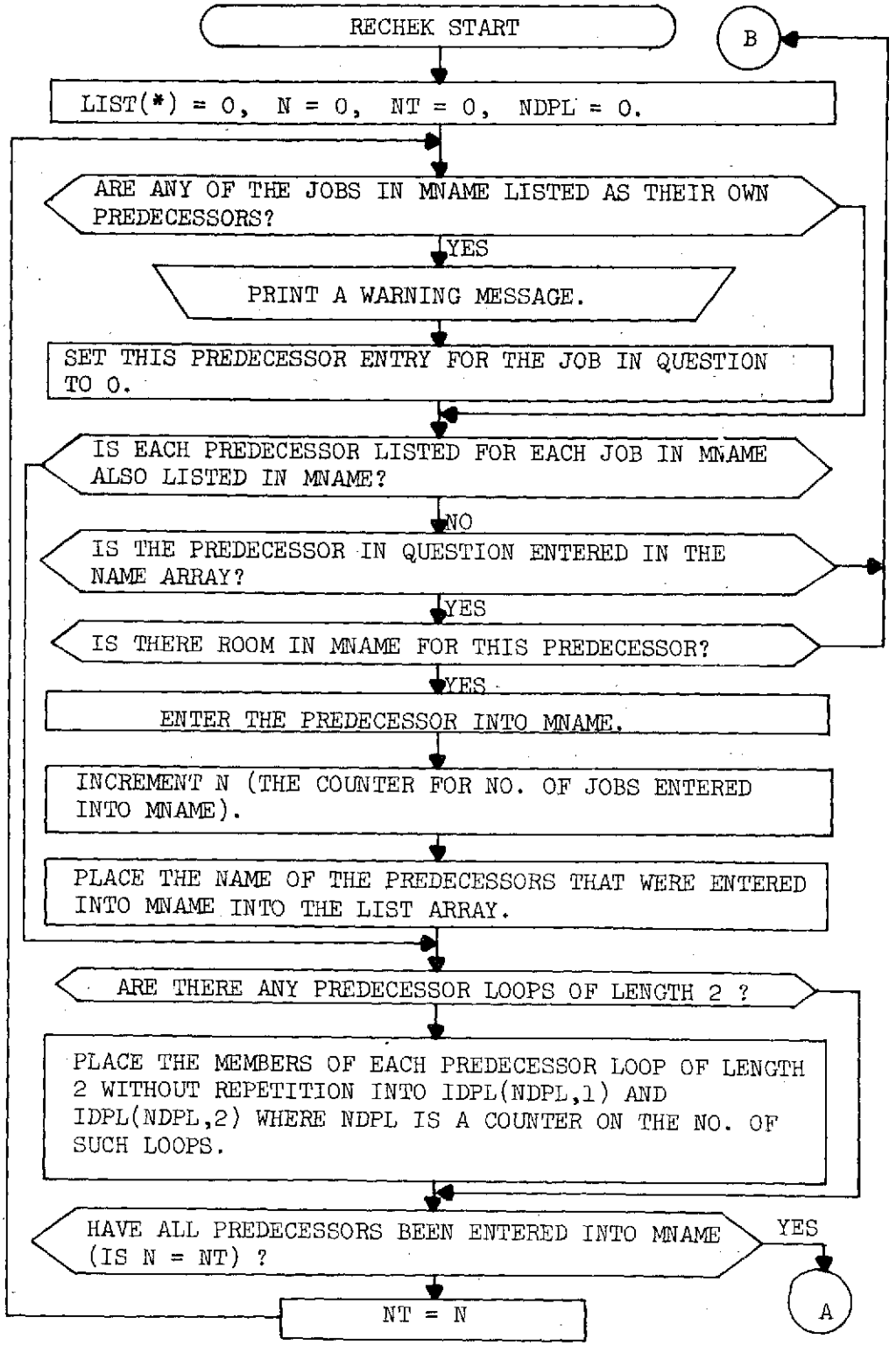


Fig. B-7 RECHEK subroutine (sheet 1 of 4)

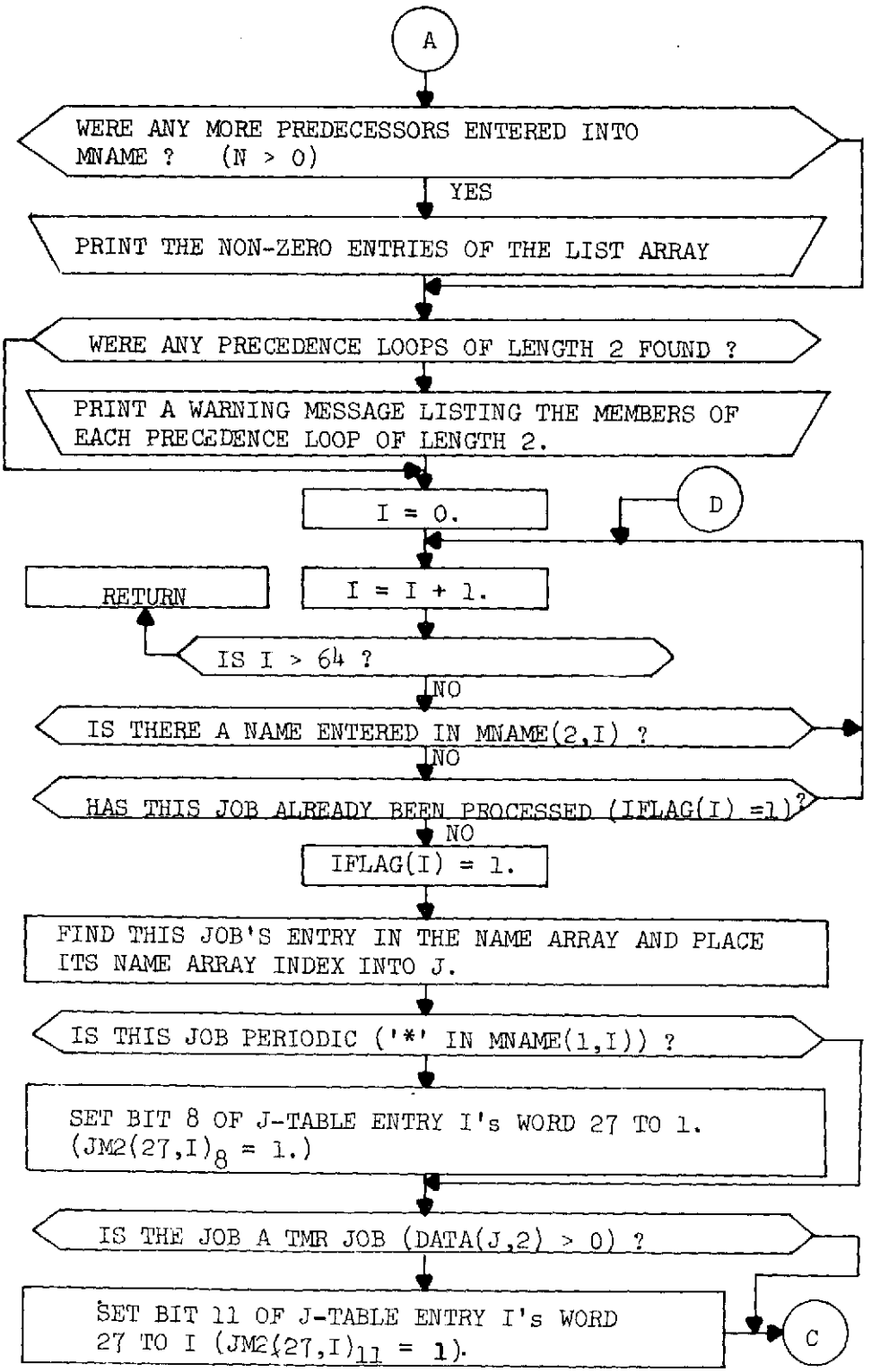


Fig. B-7 RECHEK subroutine (sheet 2 of 4)

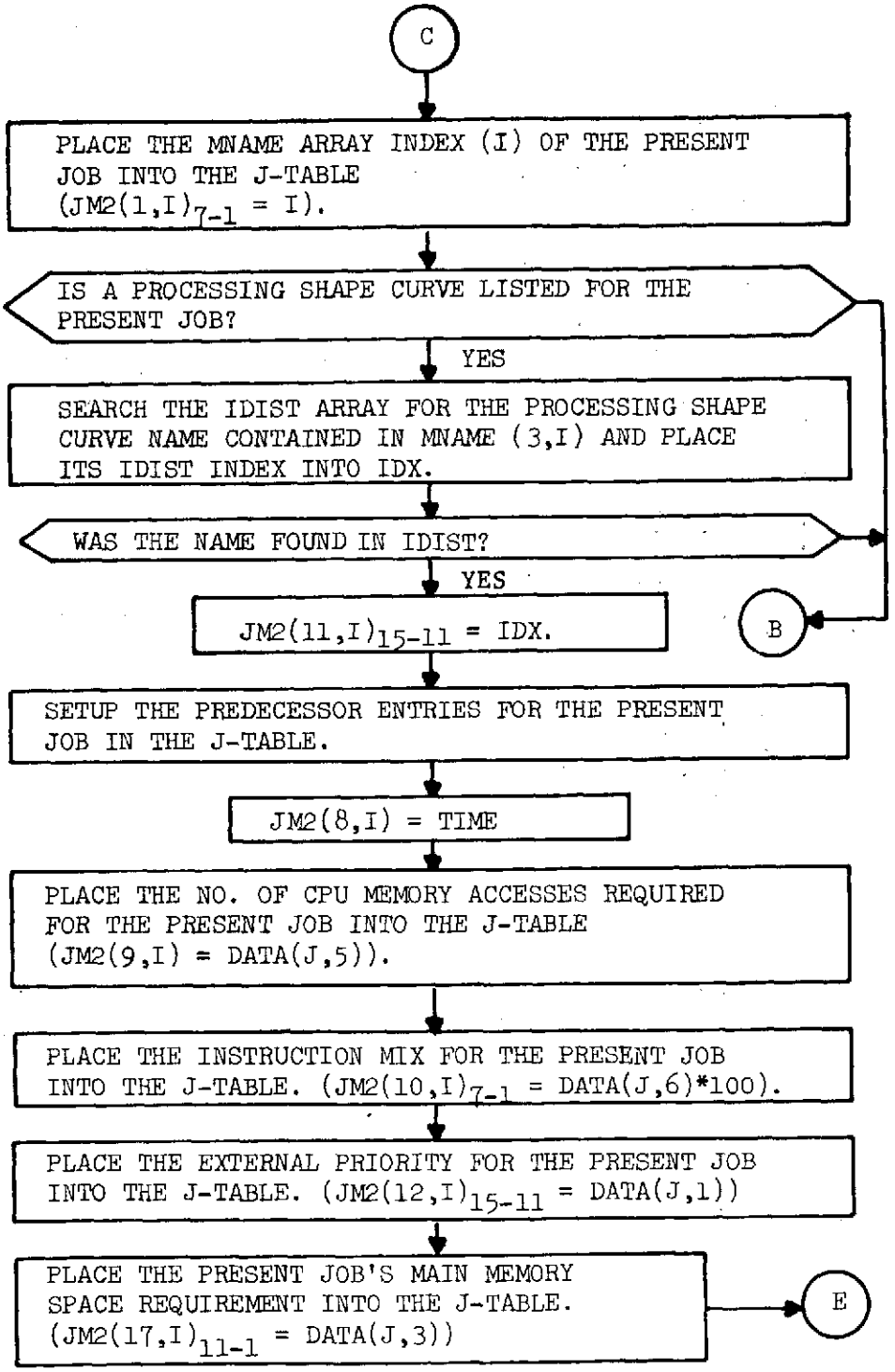


Fig. B-7 RECHEK subroutine (sheet 3 of 4)

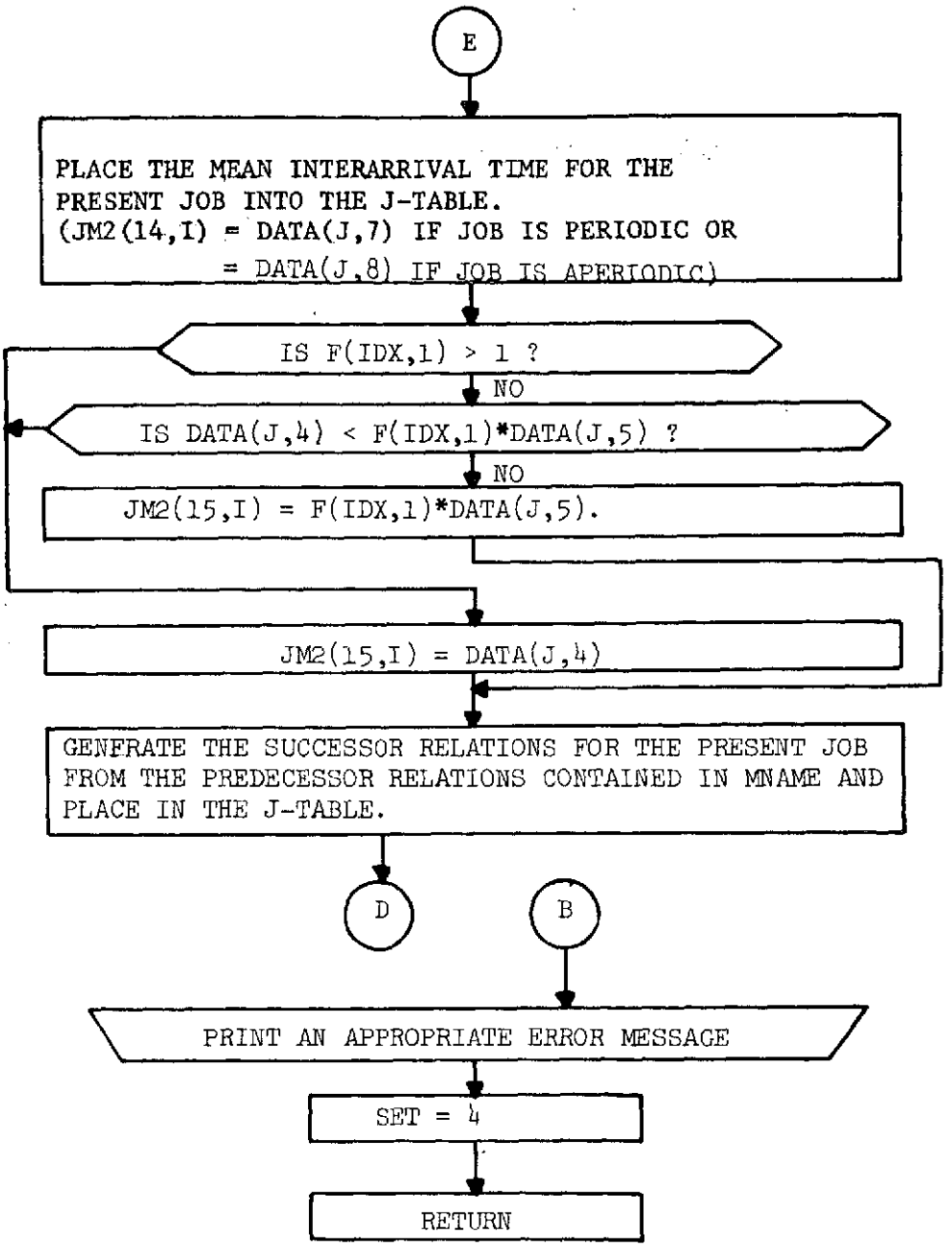


Fig. B-7 RECHEK subroutine (sheet 4 of 4)

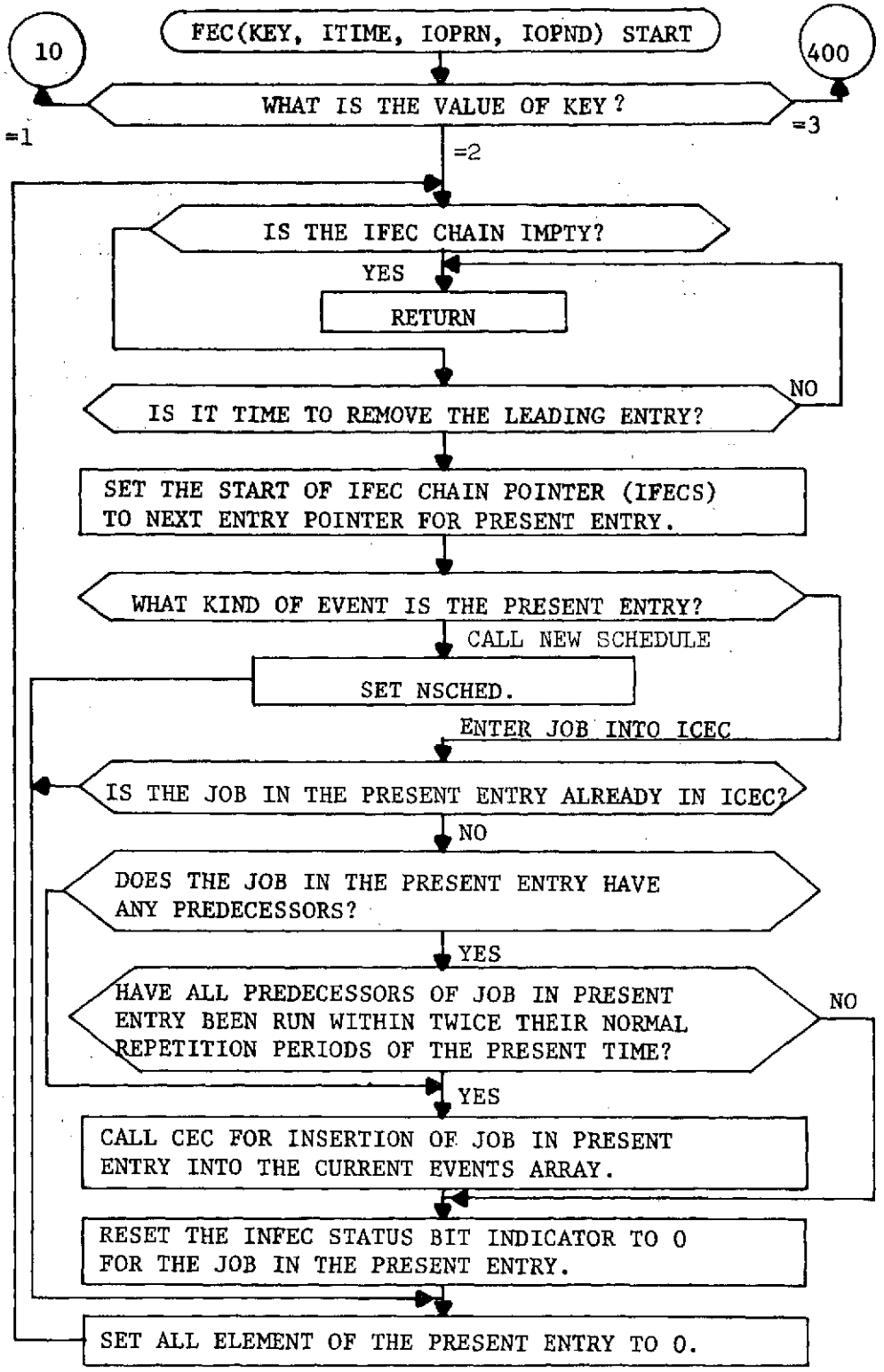


Fig. B-8 **FEC** subroutine (sheet 1 of 2)

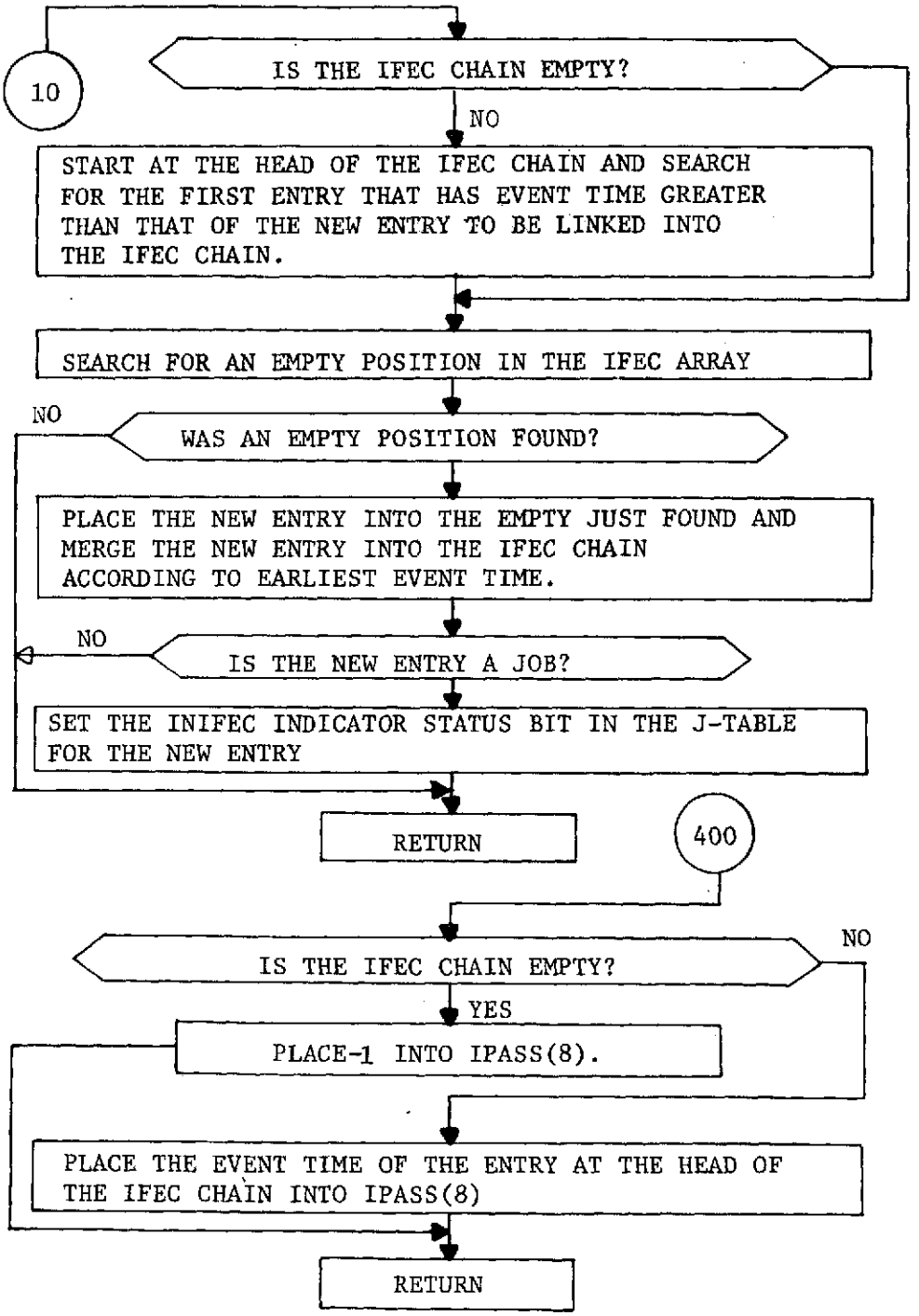


Fig. B-8 FEC subroutine (sheet 2 of 2)

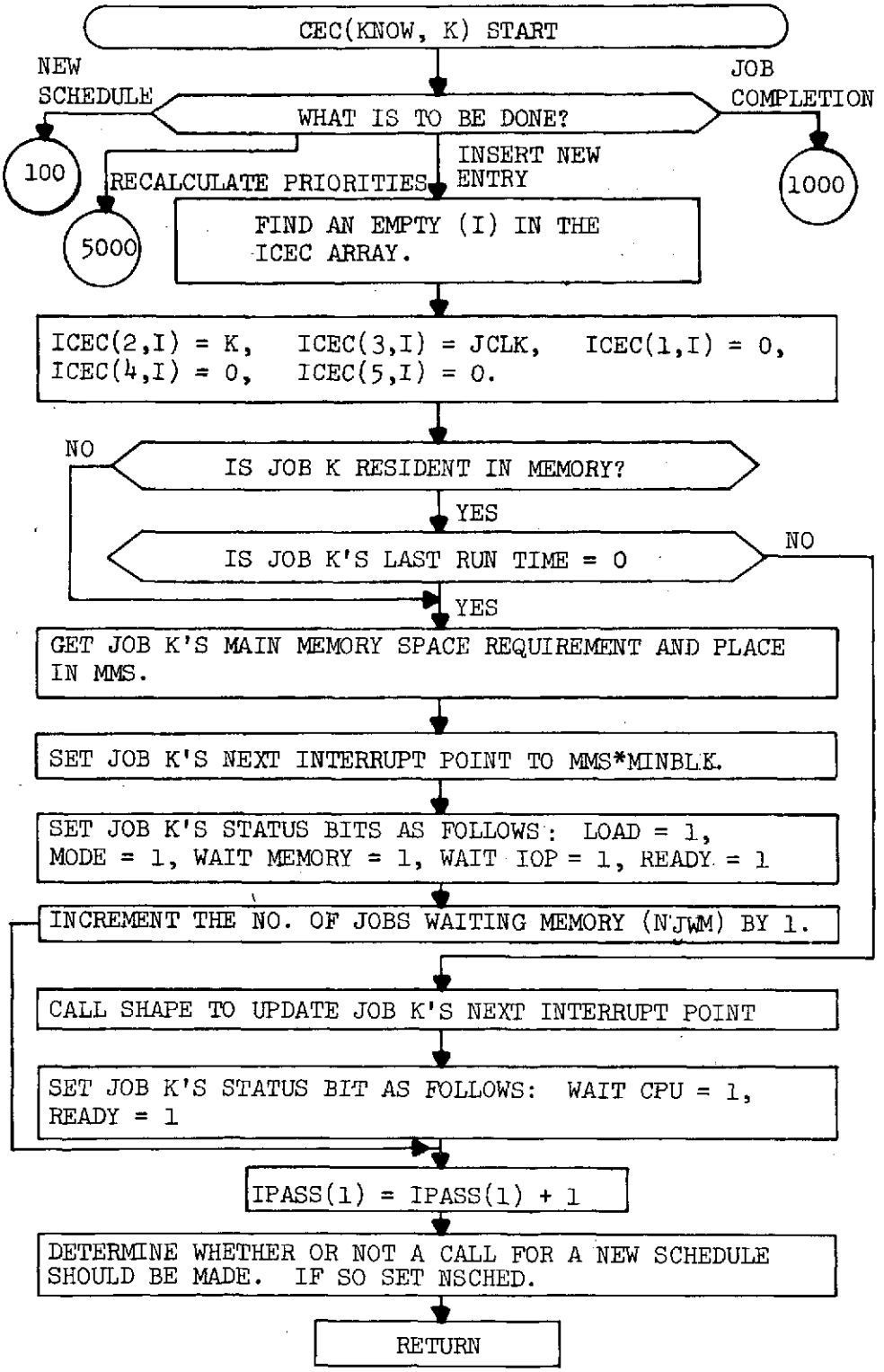


Fig. B-9 CEC subroutine (sheet 1 of 4)

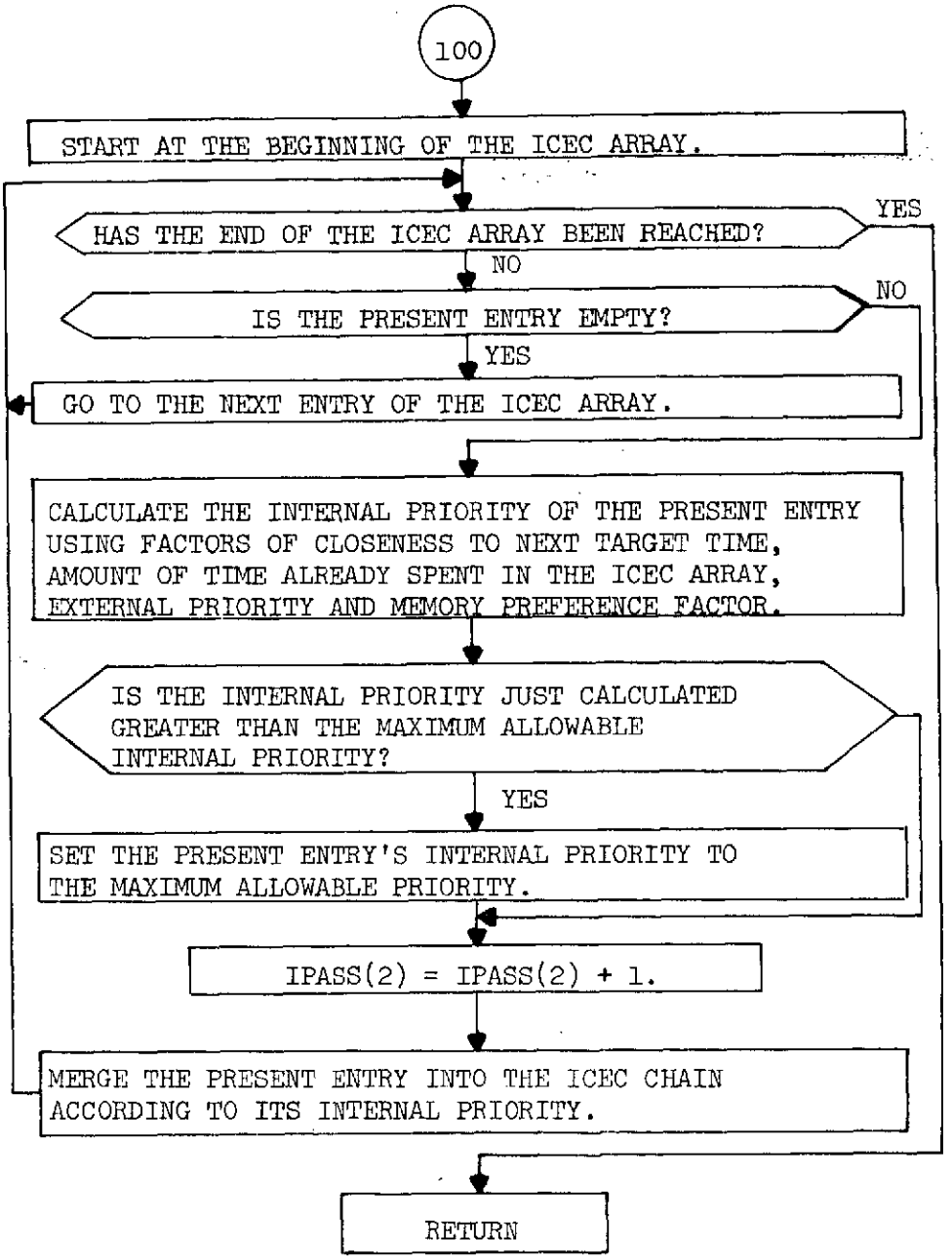


Fig. B-9 CEC subroutine sheet (2 of 4)

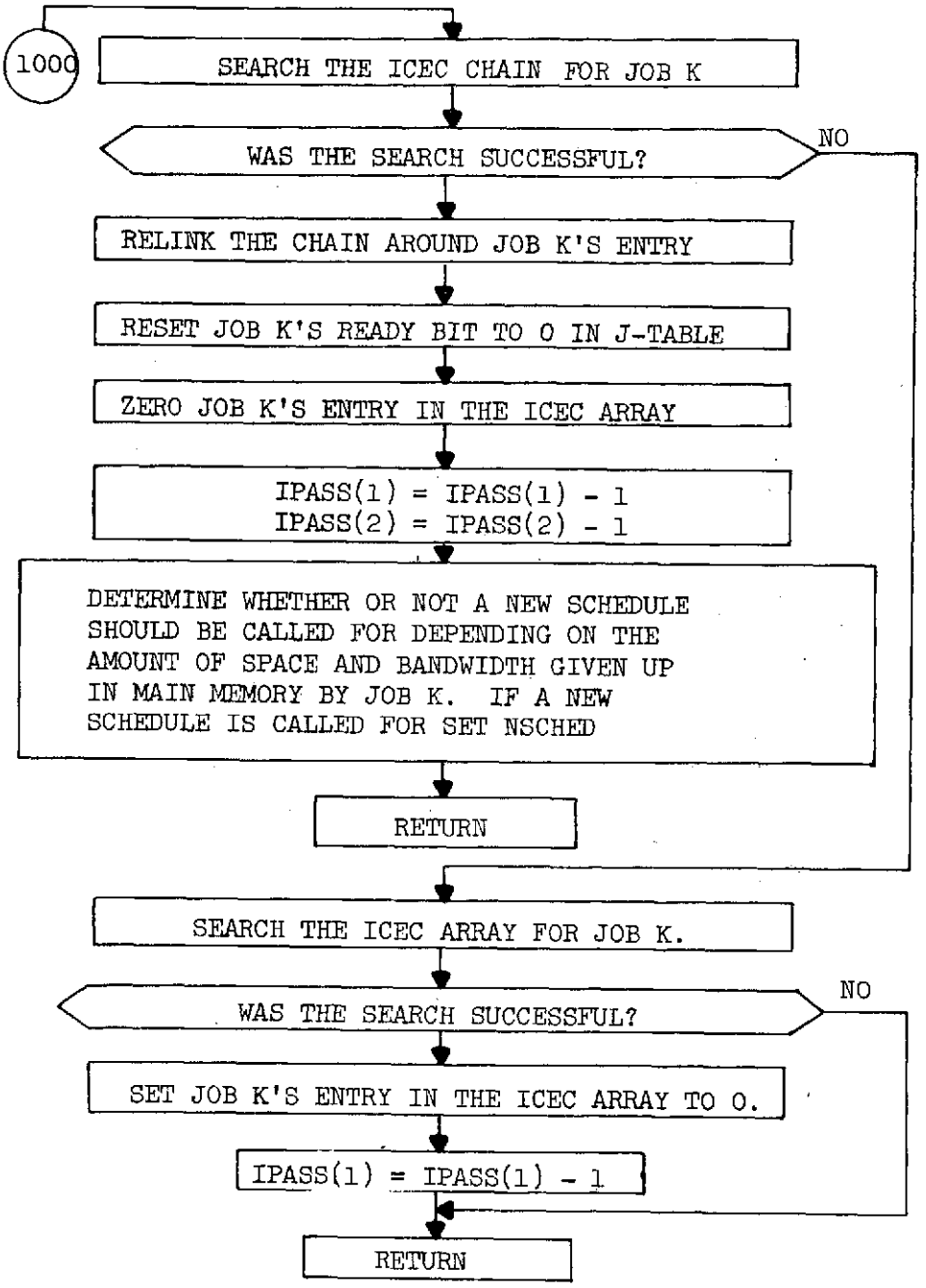


Fig. B-9 CEC subroutine (sheet 3 of 4)

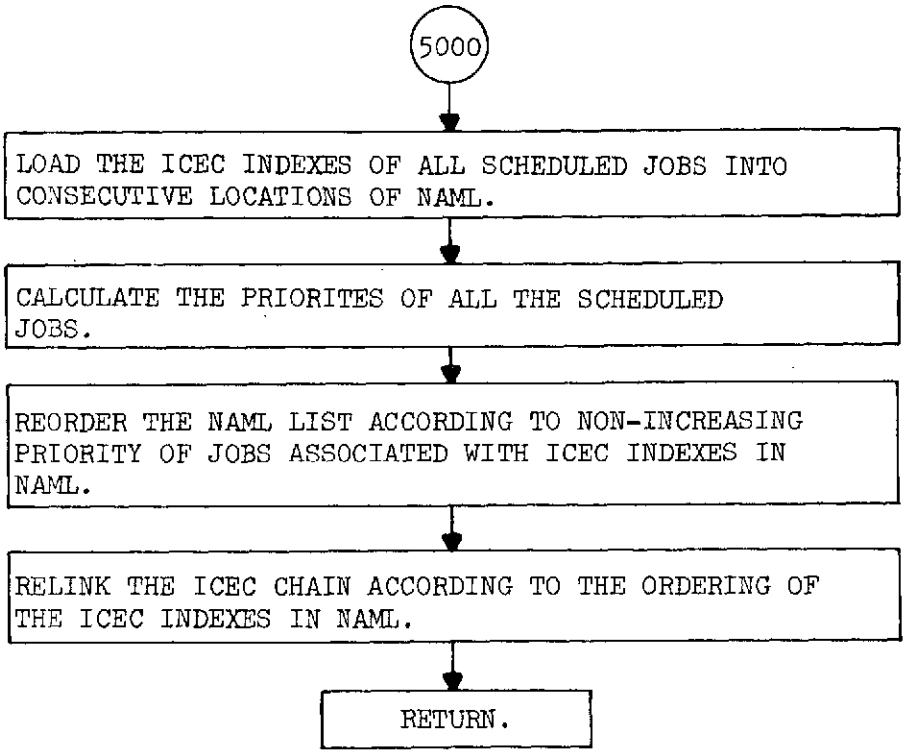


Fig. B-9 CEC subroutine (sheet 4 of 4)

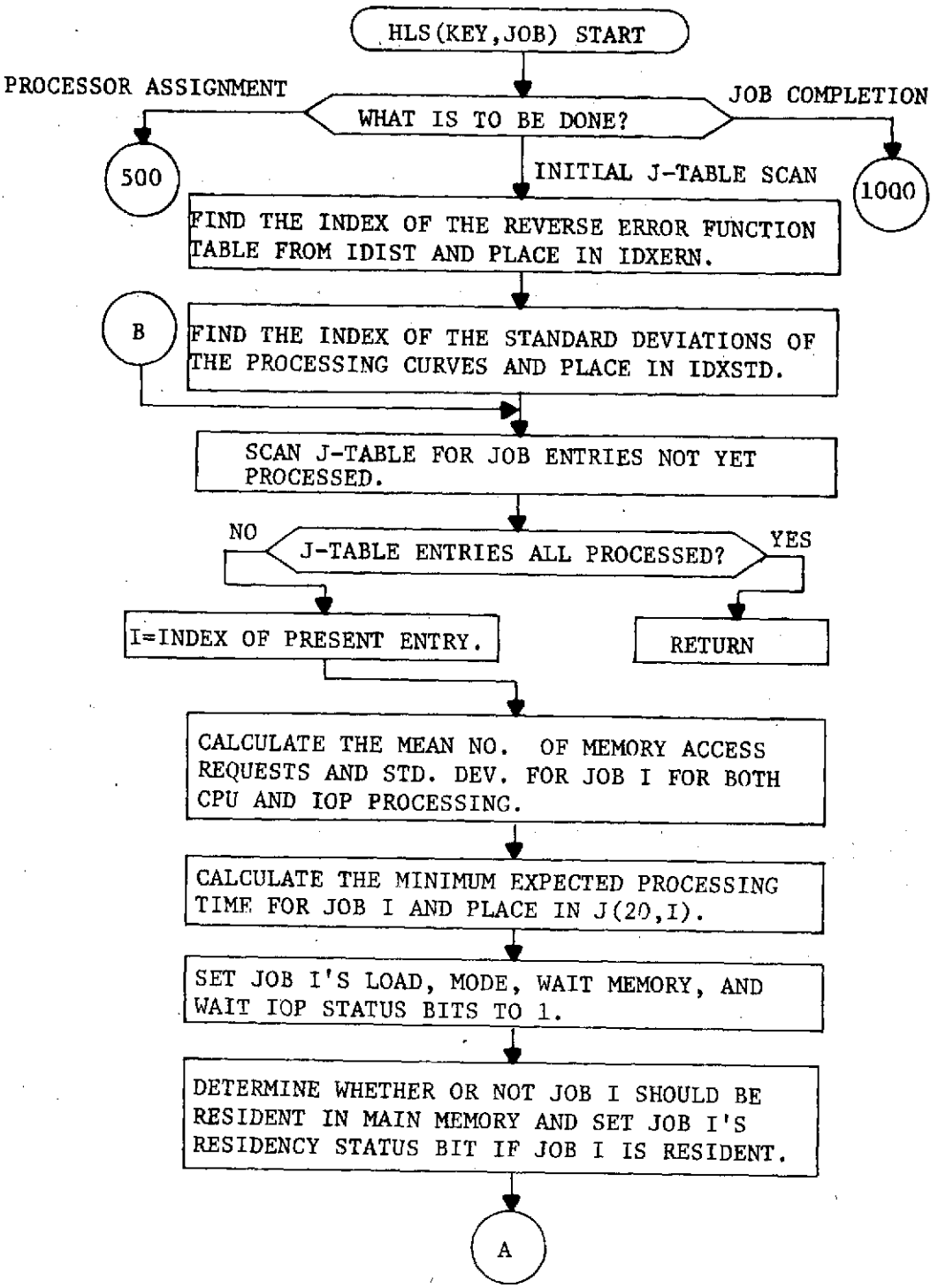


Fig. B-10 HLS subroutine (sheet 1 of 6)

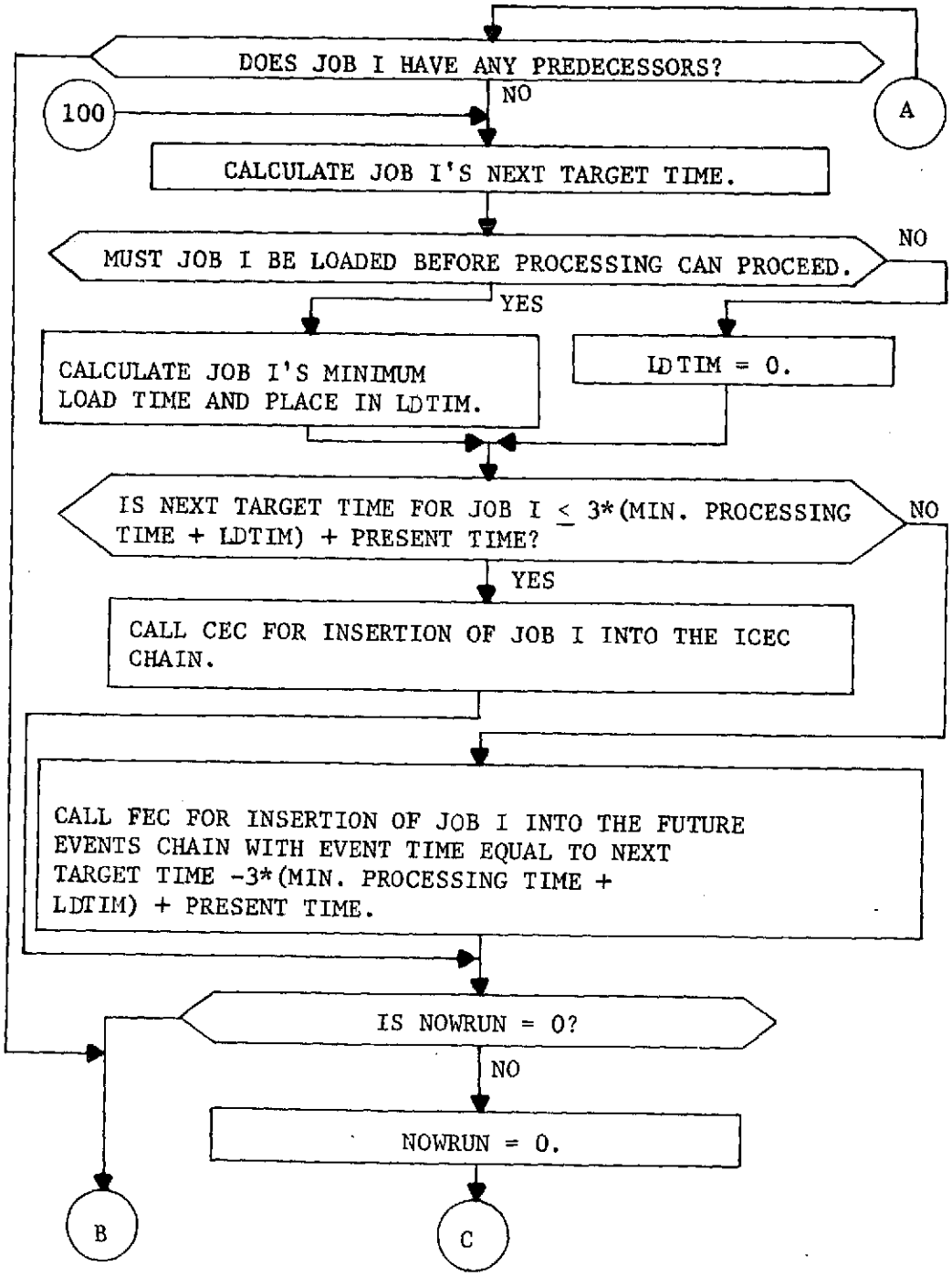


Fig. B-10 HLS subroutine (sheet 2 of 6)

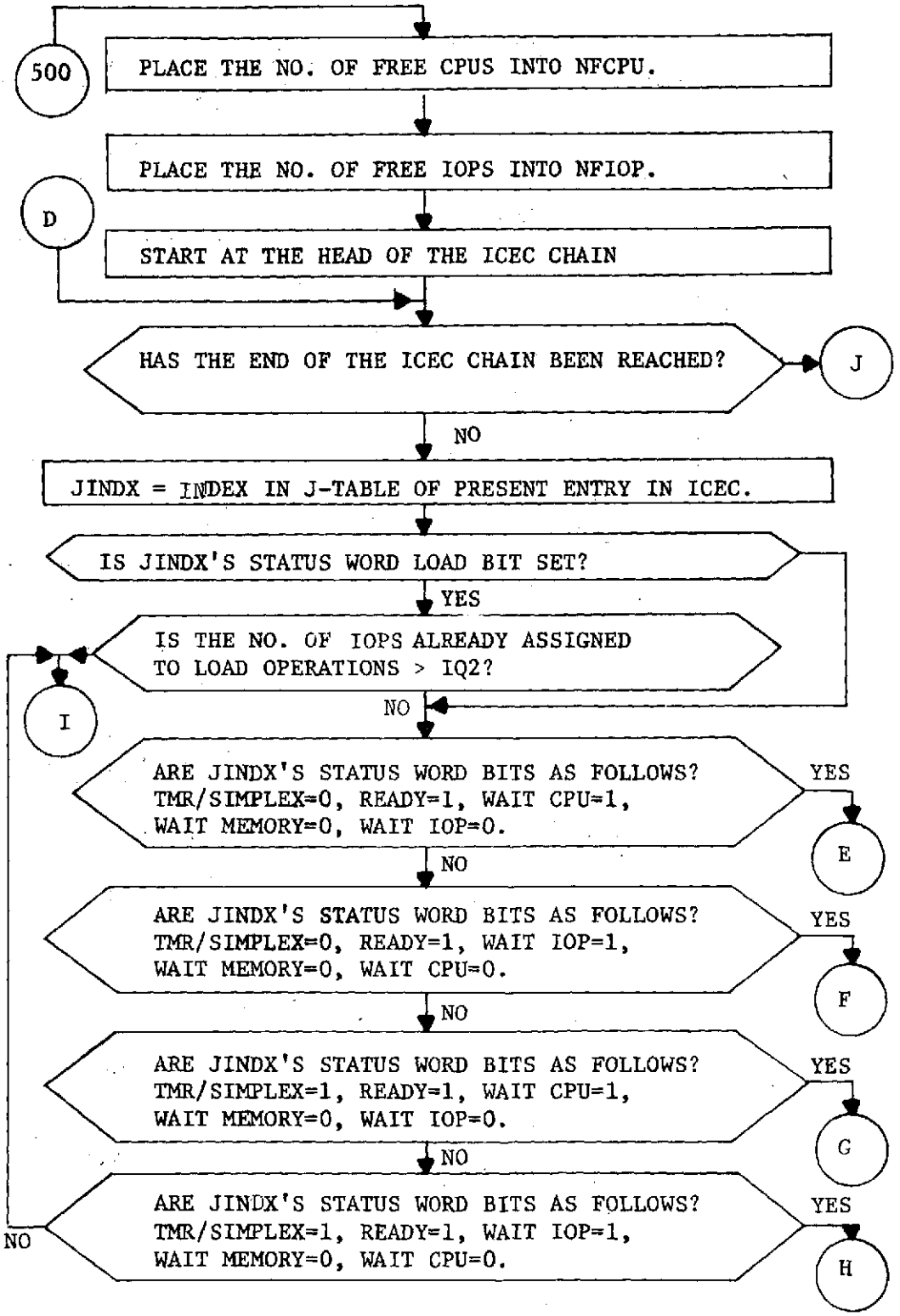


Fig. B-10 HLS subroutine (sheet 3 of 6)

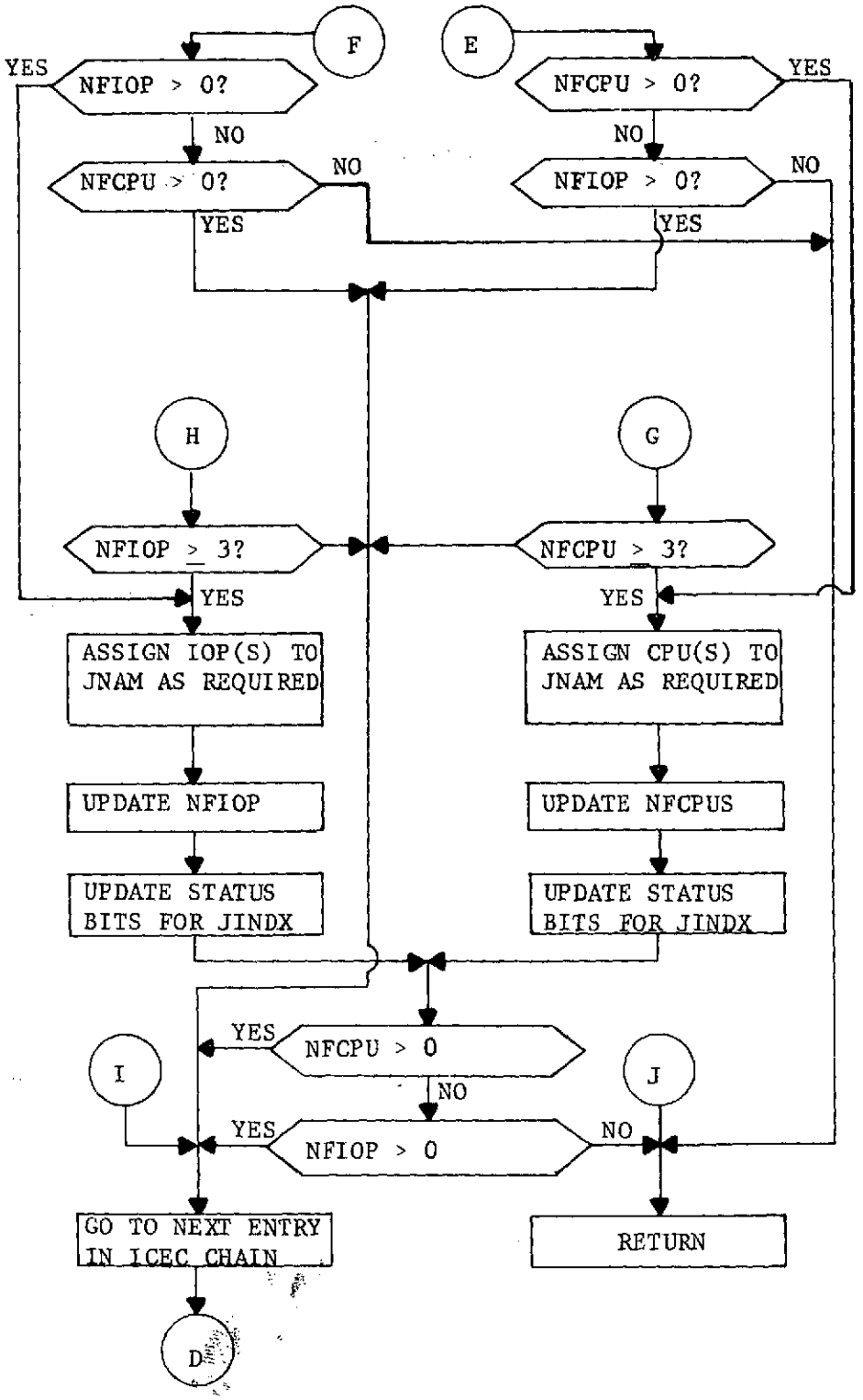


Fig. B-10 HLS subroutine (sheet 4 of 6)

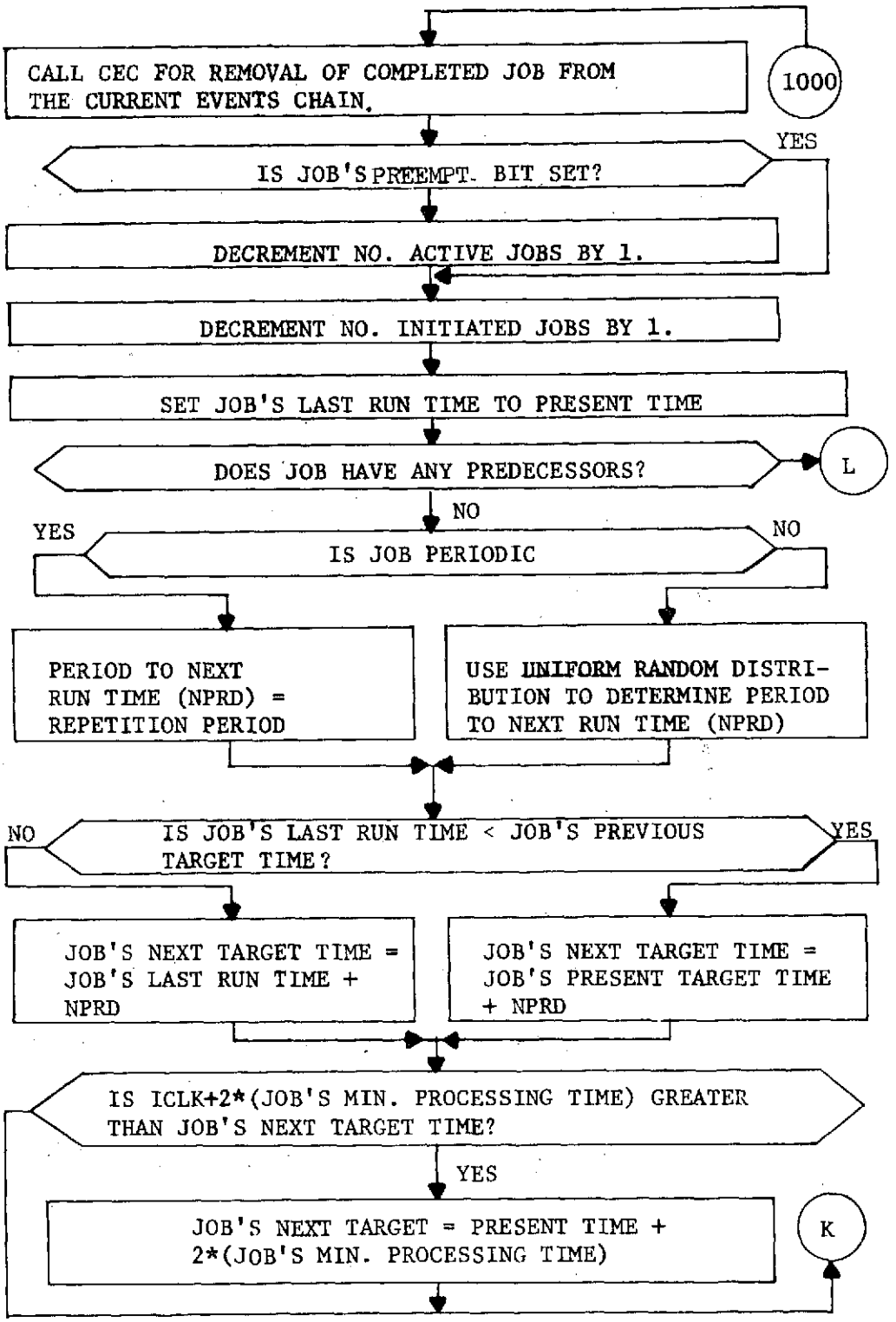


Fig. B-10 HLS subroutine (sheet 5 of 6)

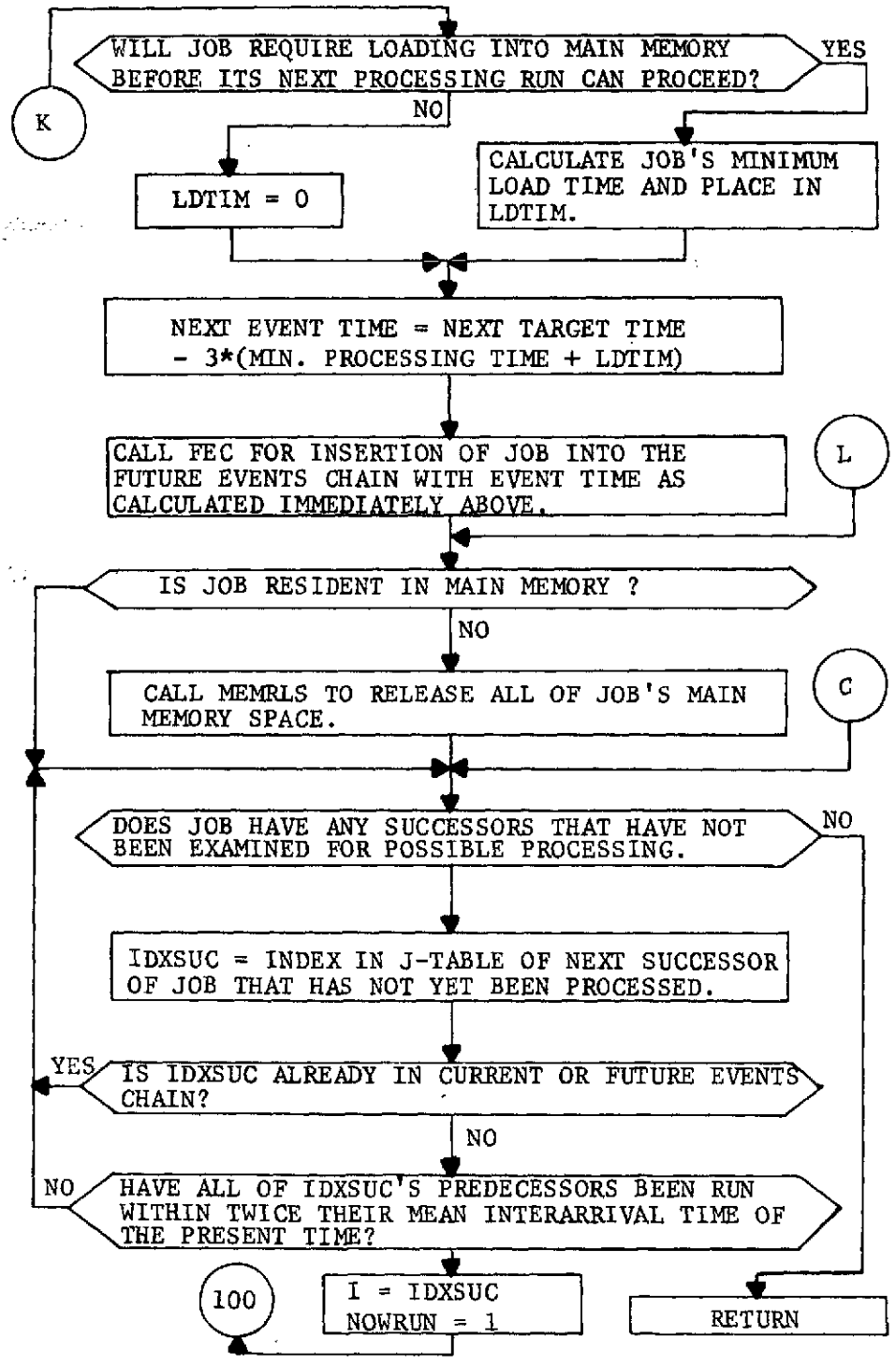


Fig. B-10 HLS subroutine (sheet 6 of 6)

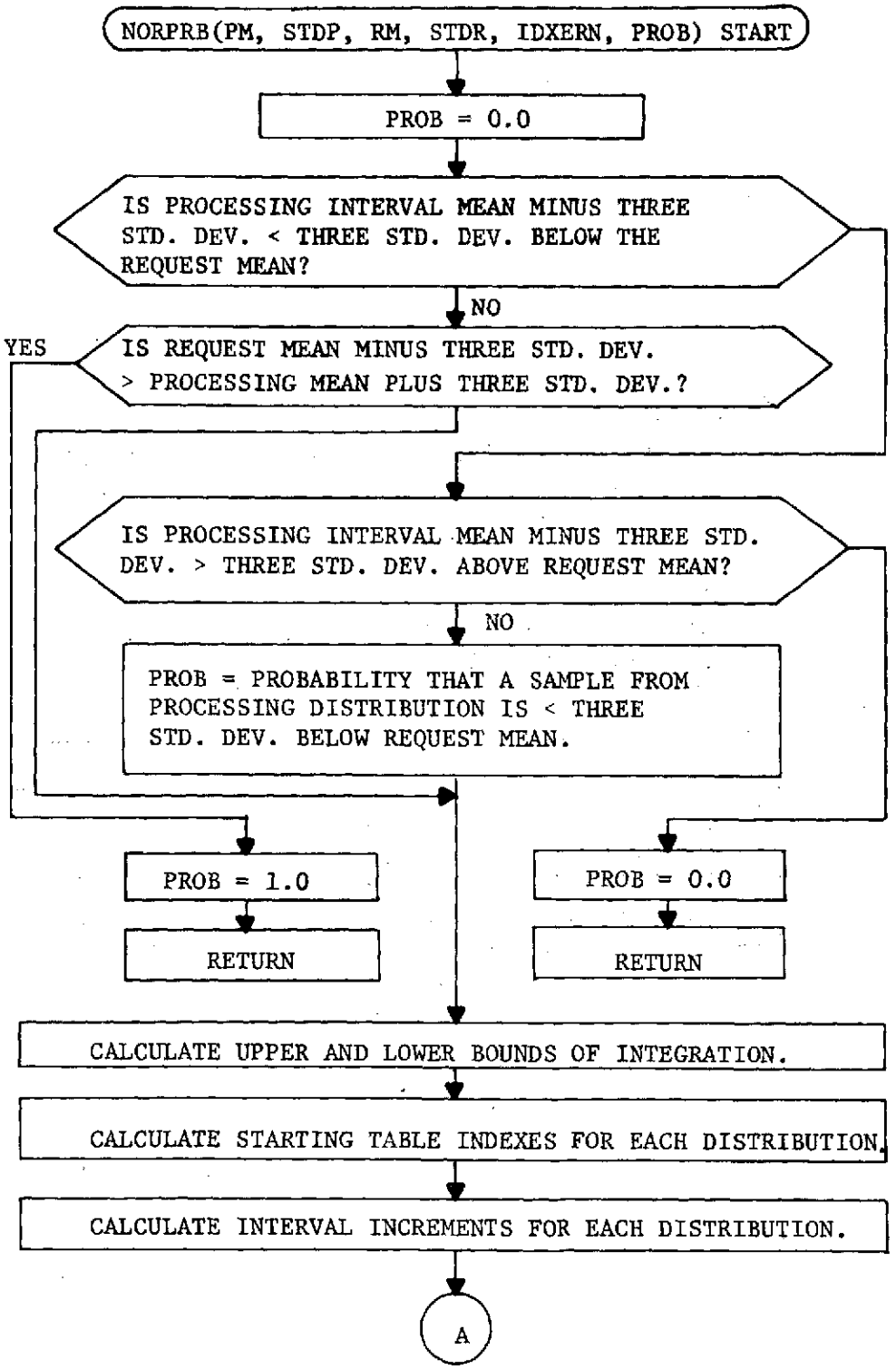


Fig. B-11 NORPRB subroutine (sheet 1 of 2)

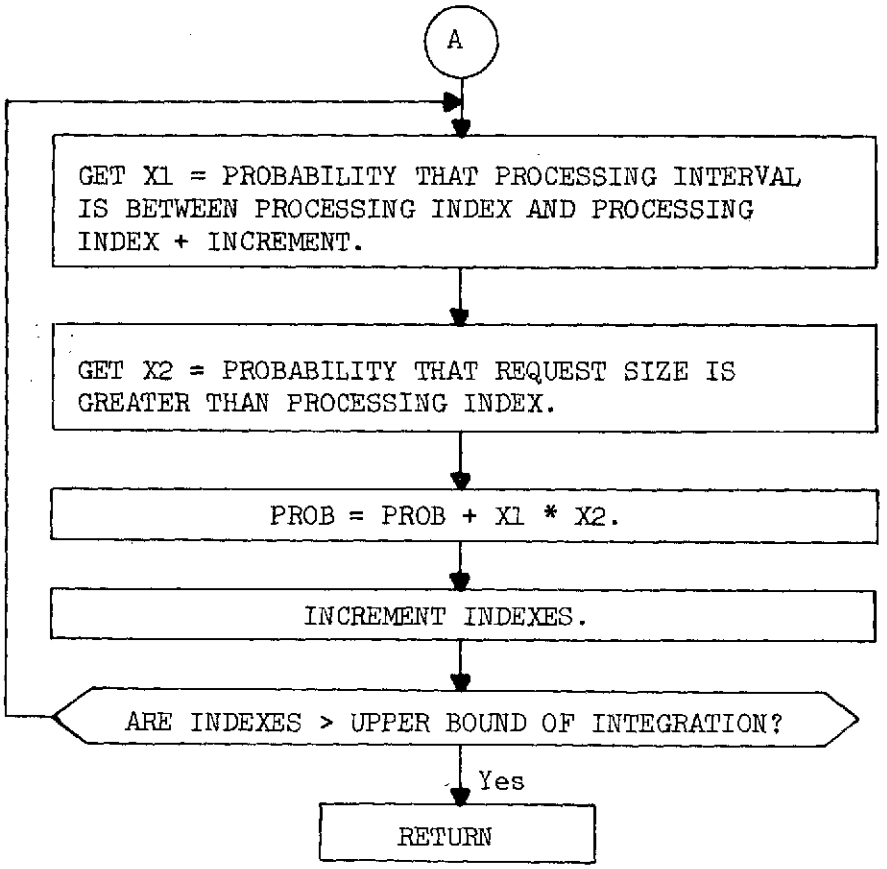


Fig. B-11 NORPRB subroutine (sheet 2 of 2)

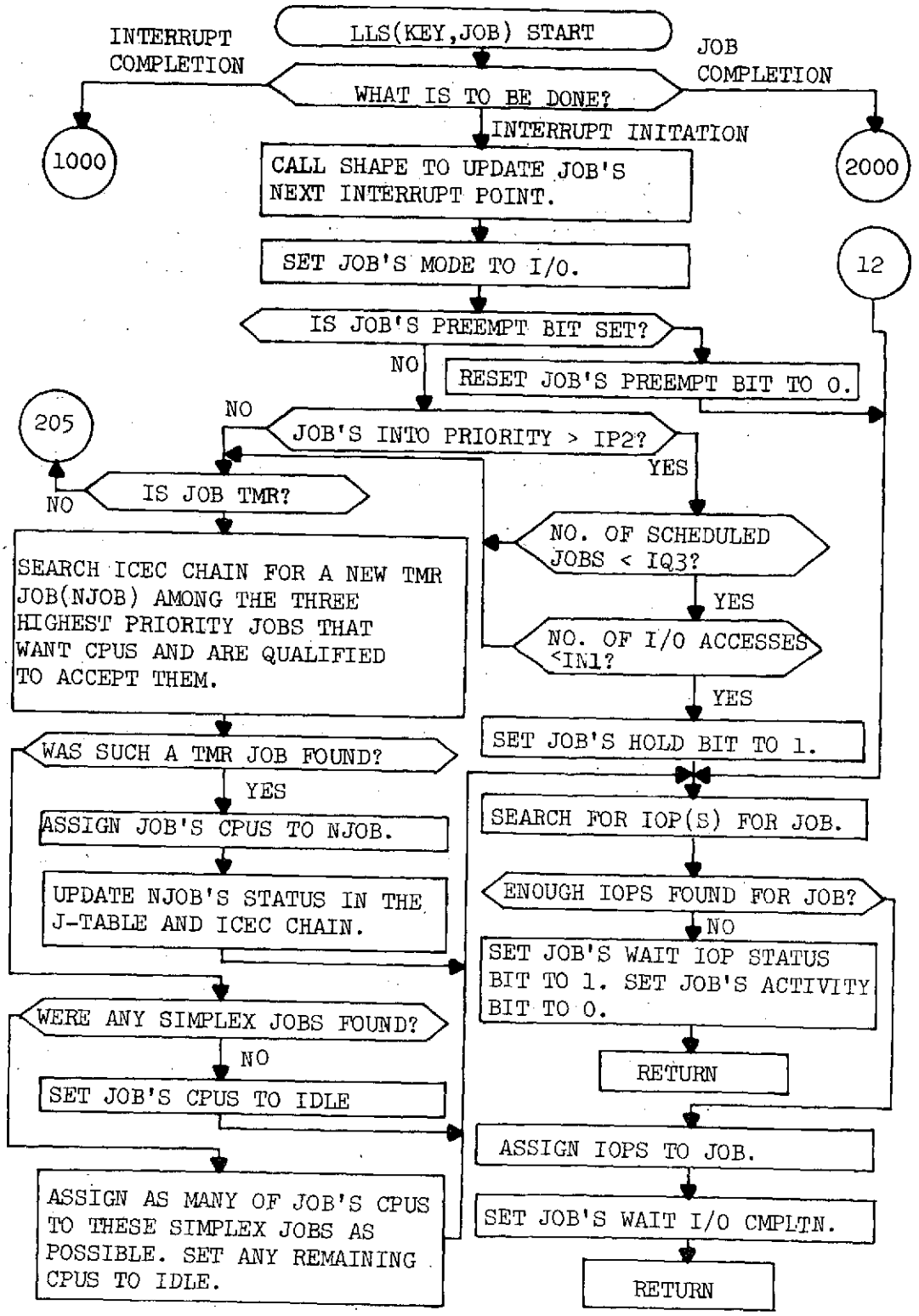


Fig. B-12 LLS subroutine (sheet 1 of 8)

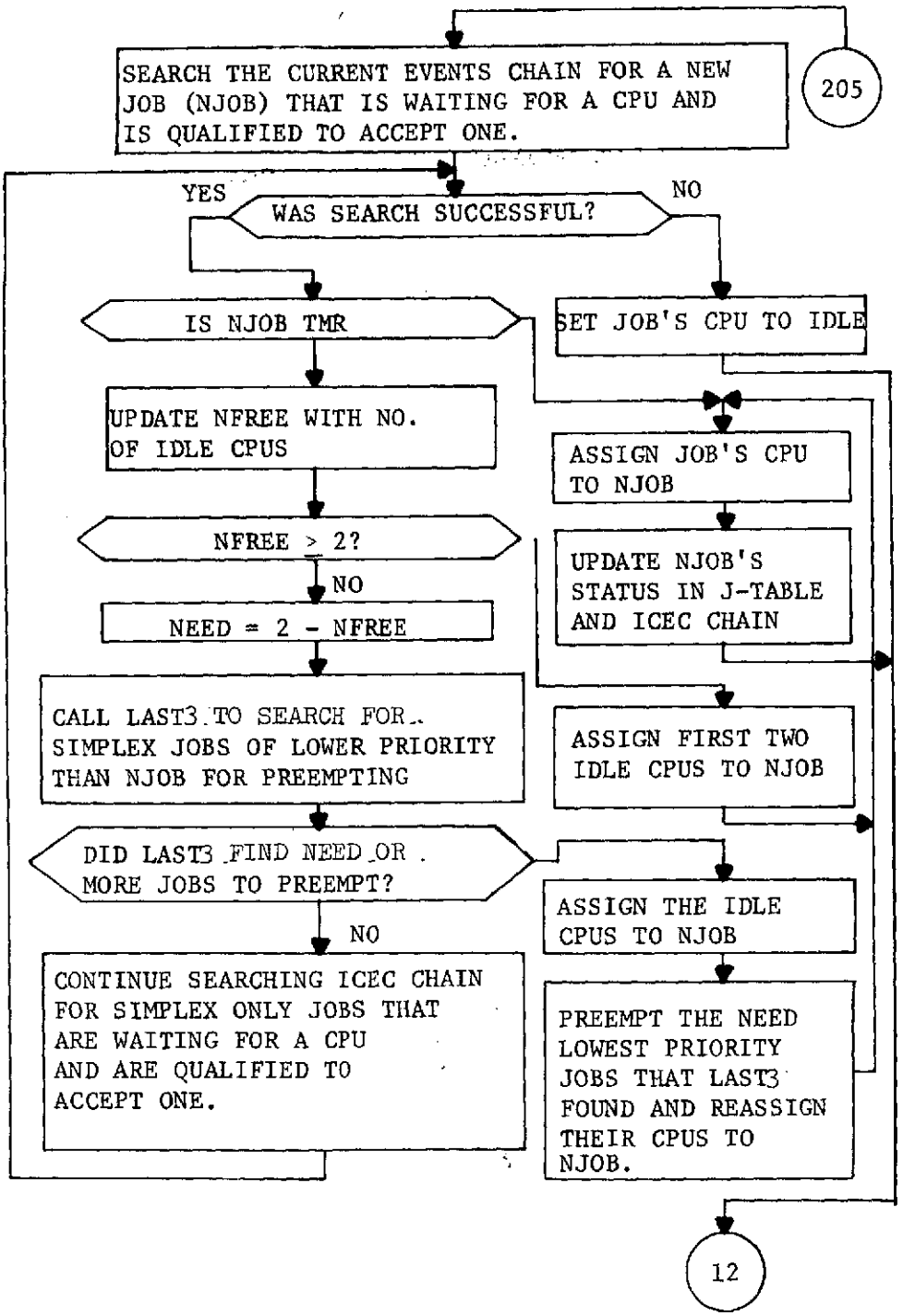


Fig. B-12 LLS subroutine (sheet 2 of 8)

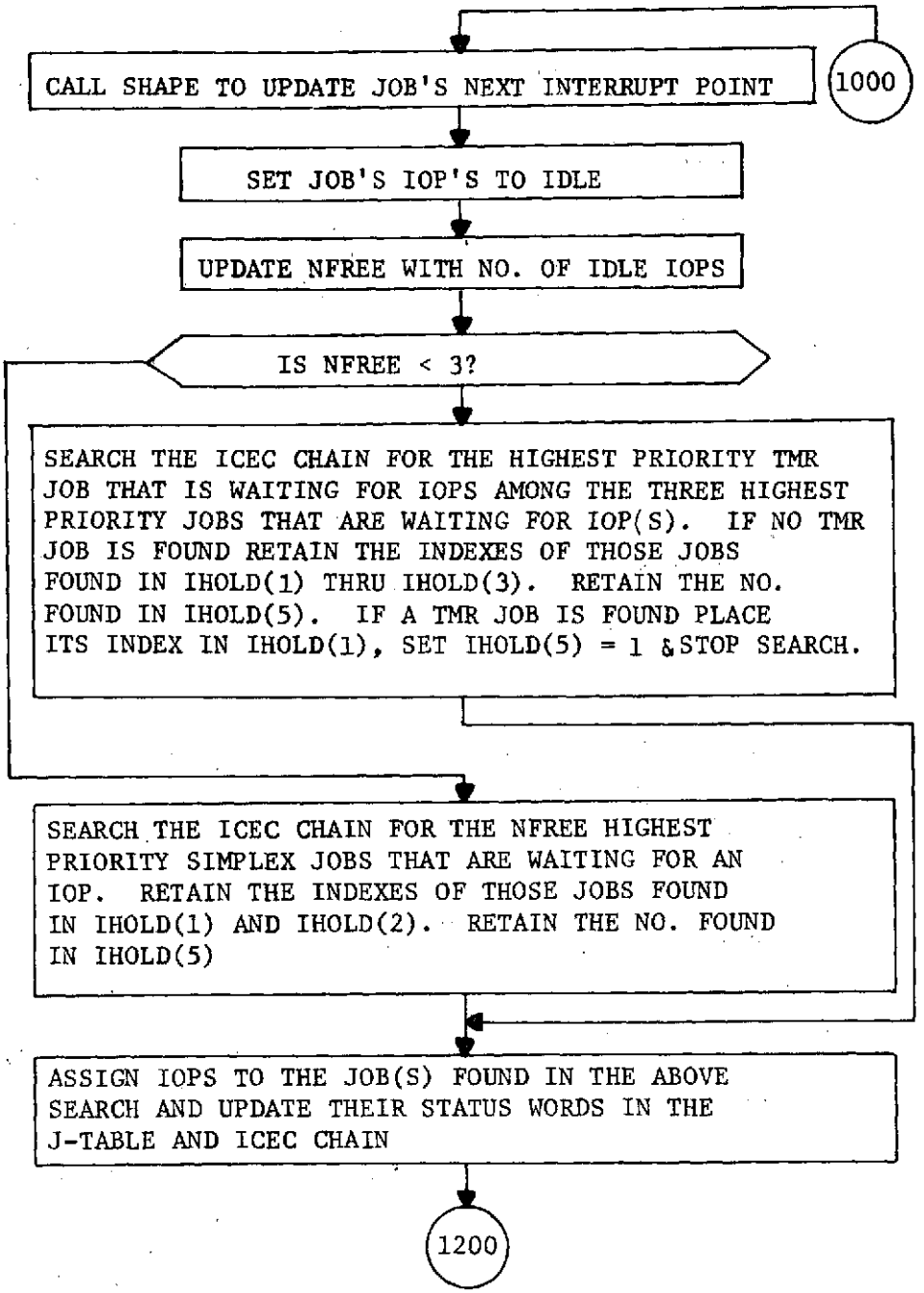


Fig. B-12 LLS subroutine (sheet 3 of 8)

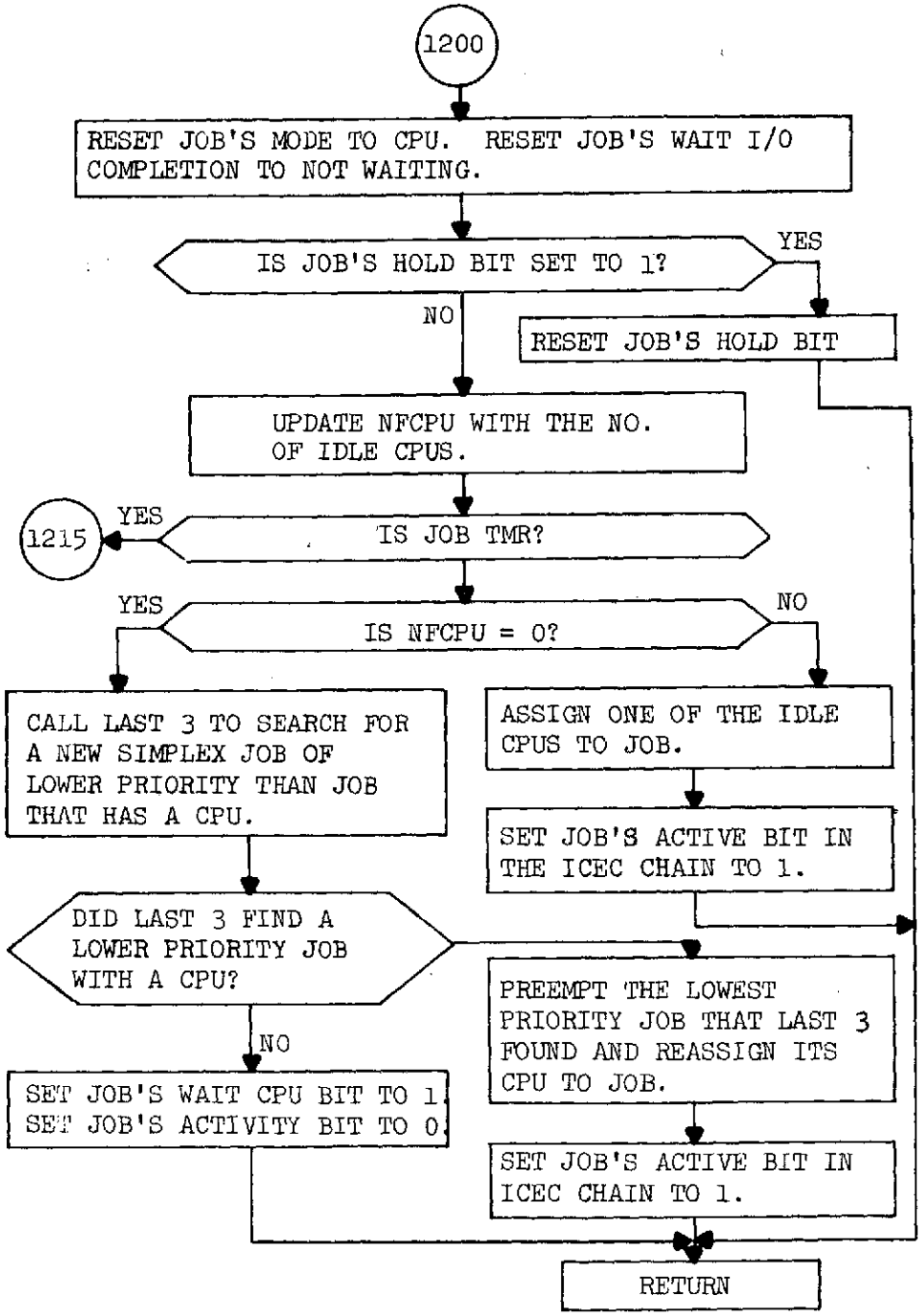


Fig. B-12 LLS subroutine (sheet 4 of 8)

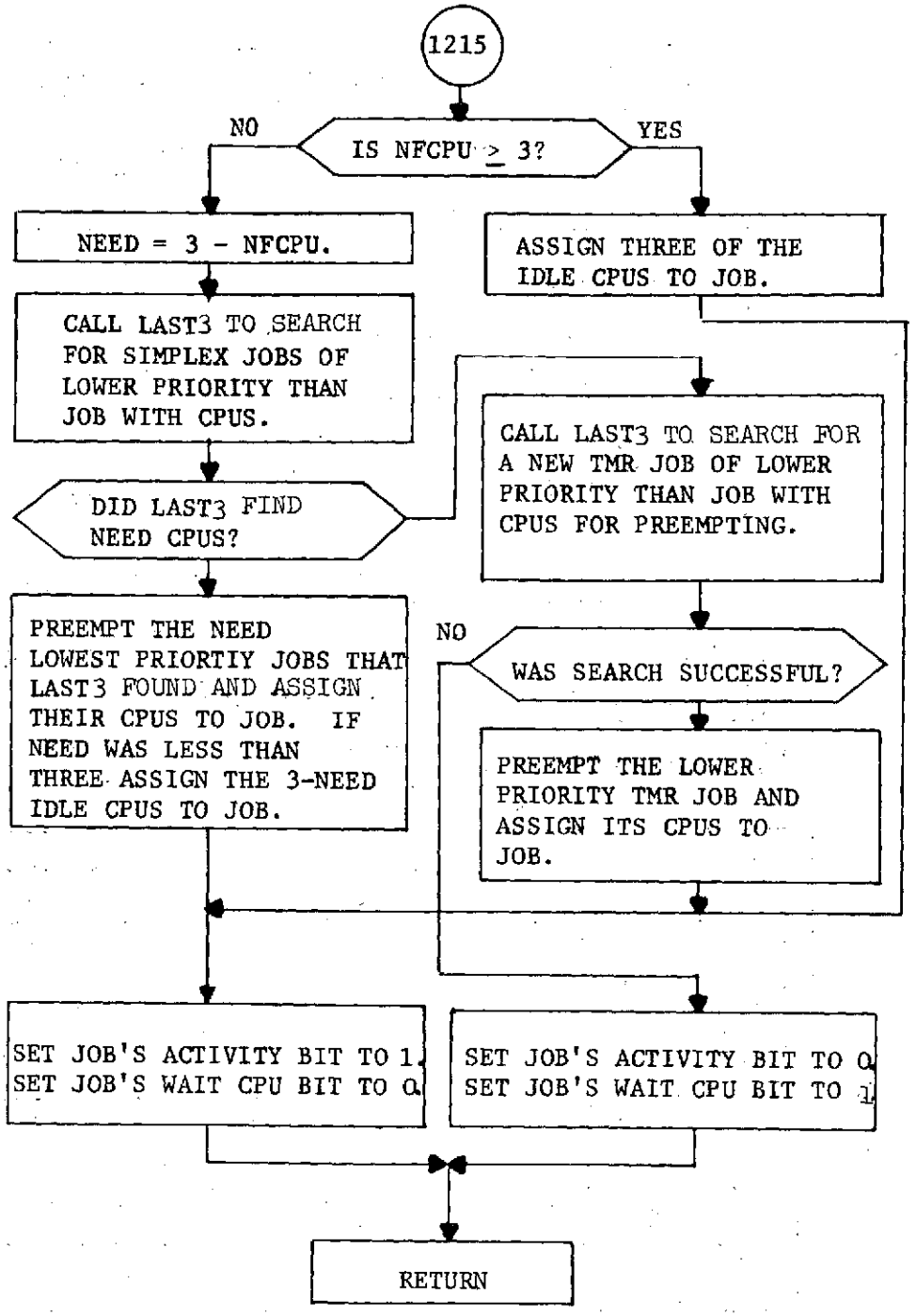


Fig. B-12 LLS subroutine (sheet 5 of 8)

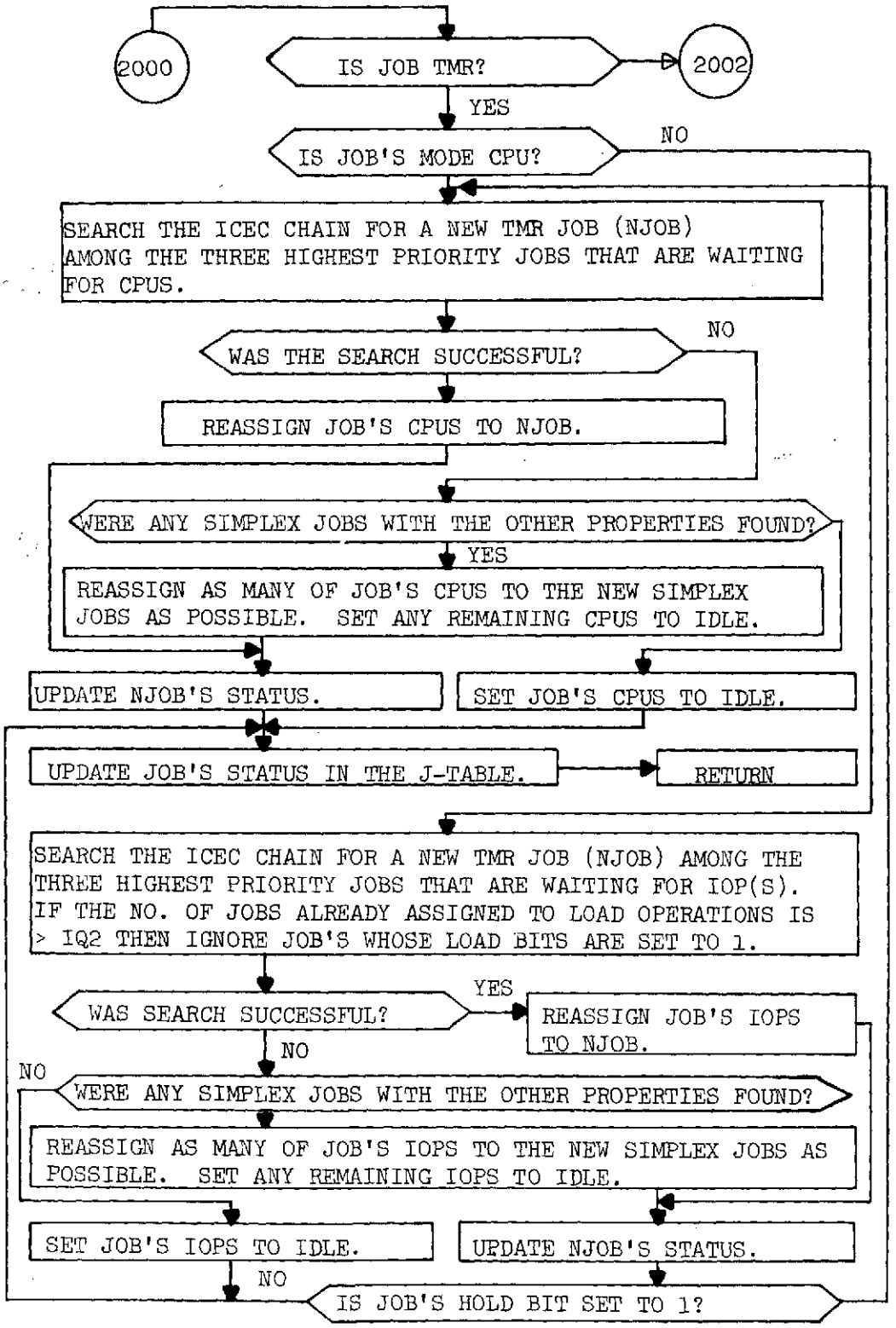


Fig. B-12 LLS subroutine (sheet 6 of 8)

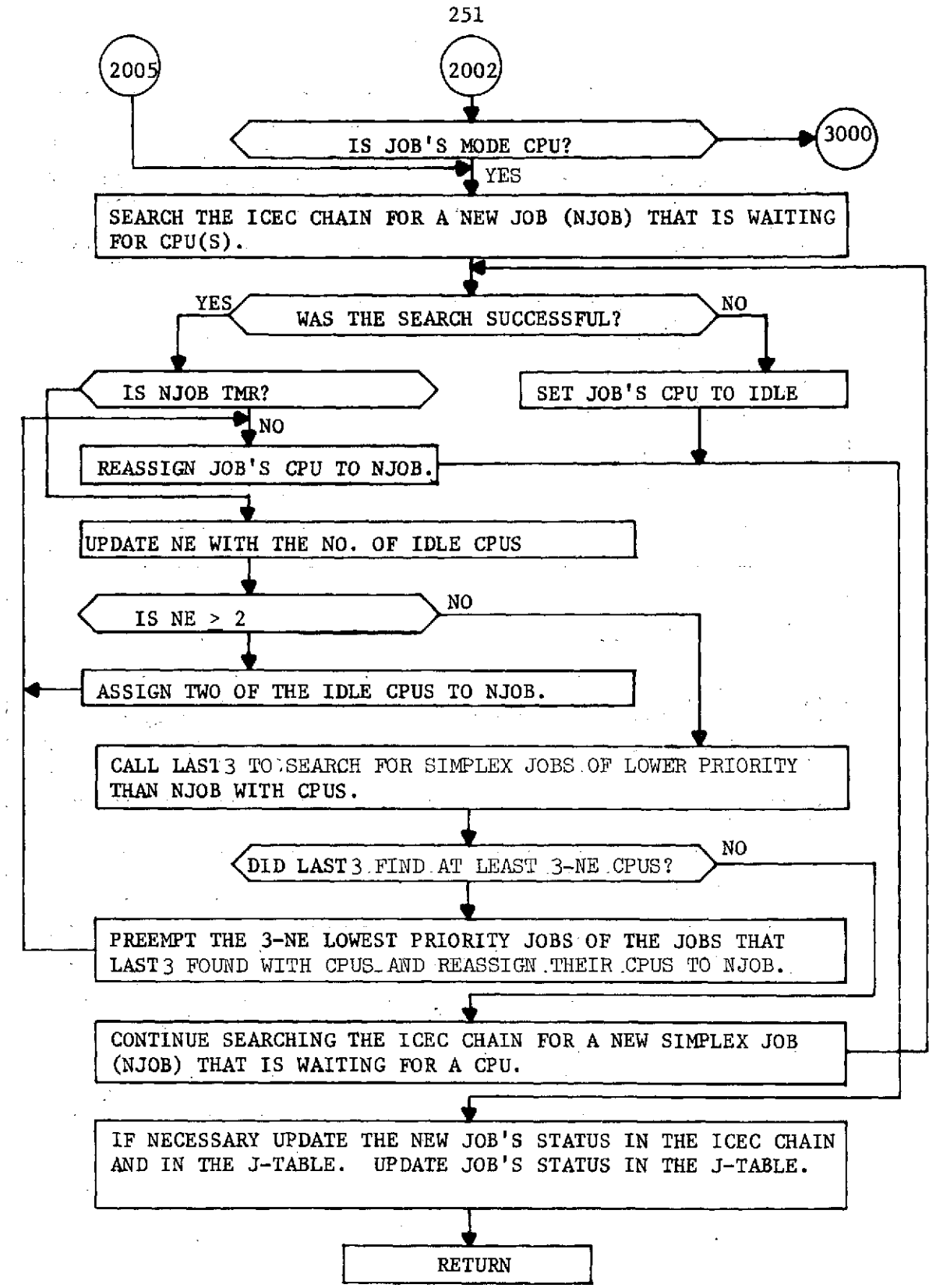


Fig. B-12 LLS subroutine (sheet 7 of 8)

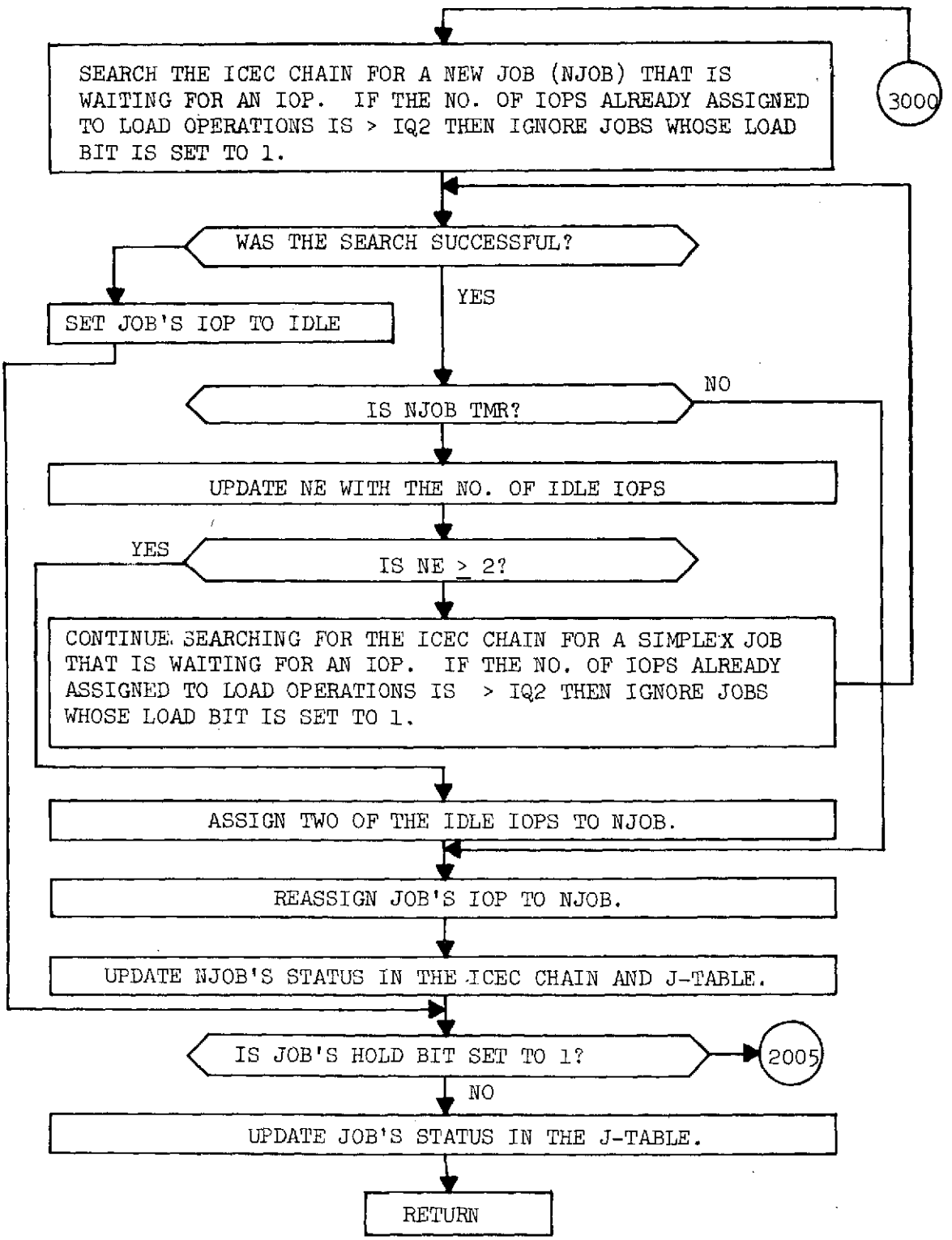


Fig. B-12 LLS subroutine (sheet 8 of 8)

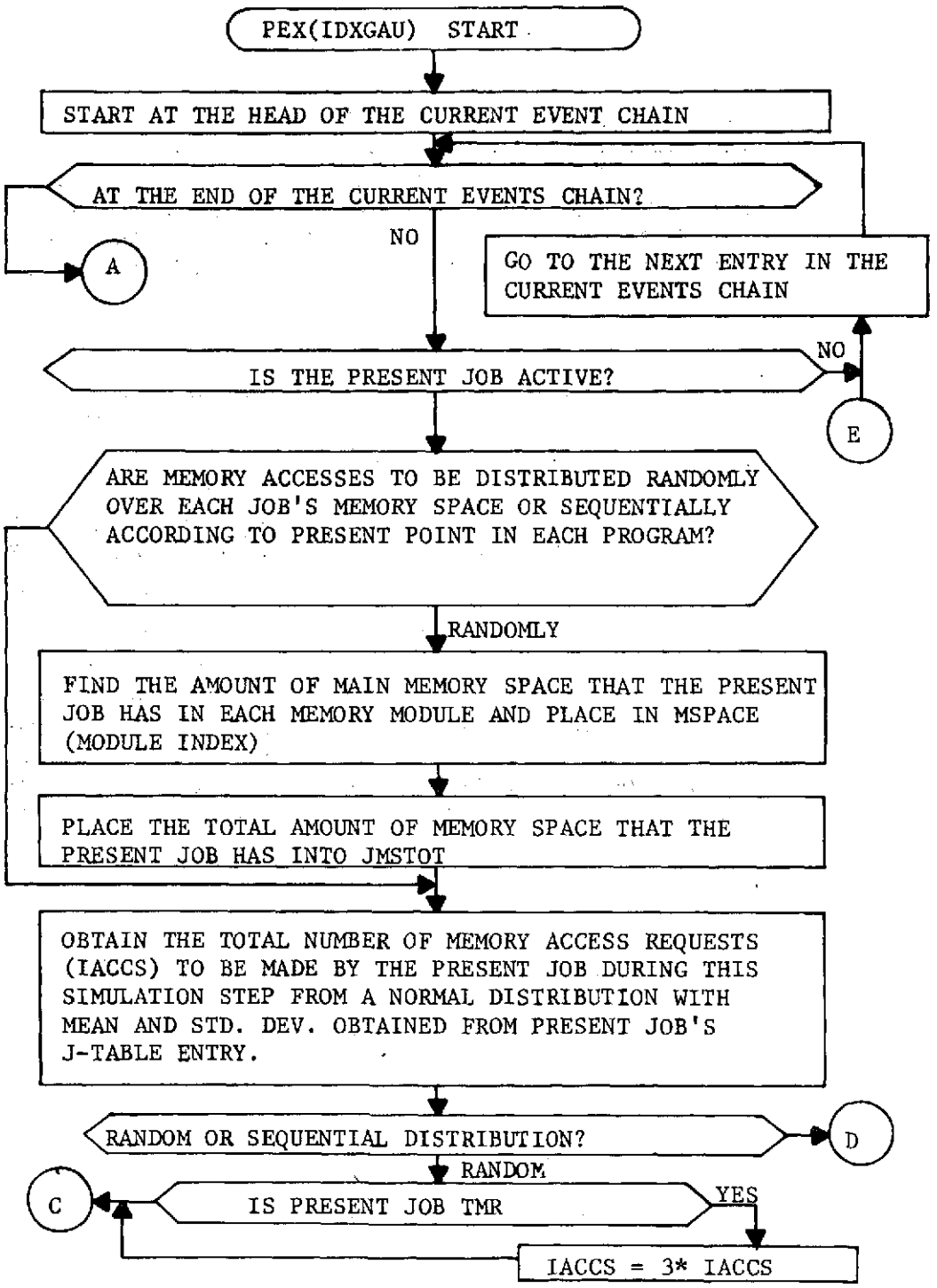


Fig. B-13 PEX subroutine (sheet 1 of 4)

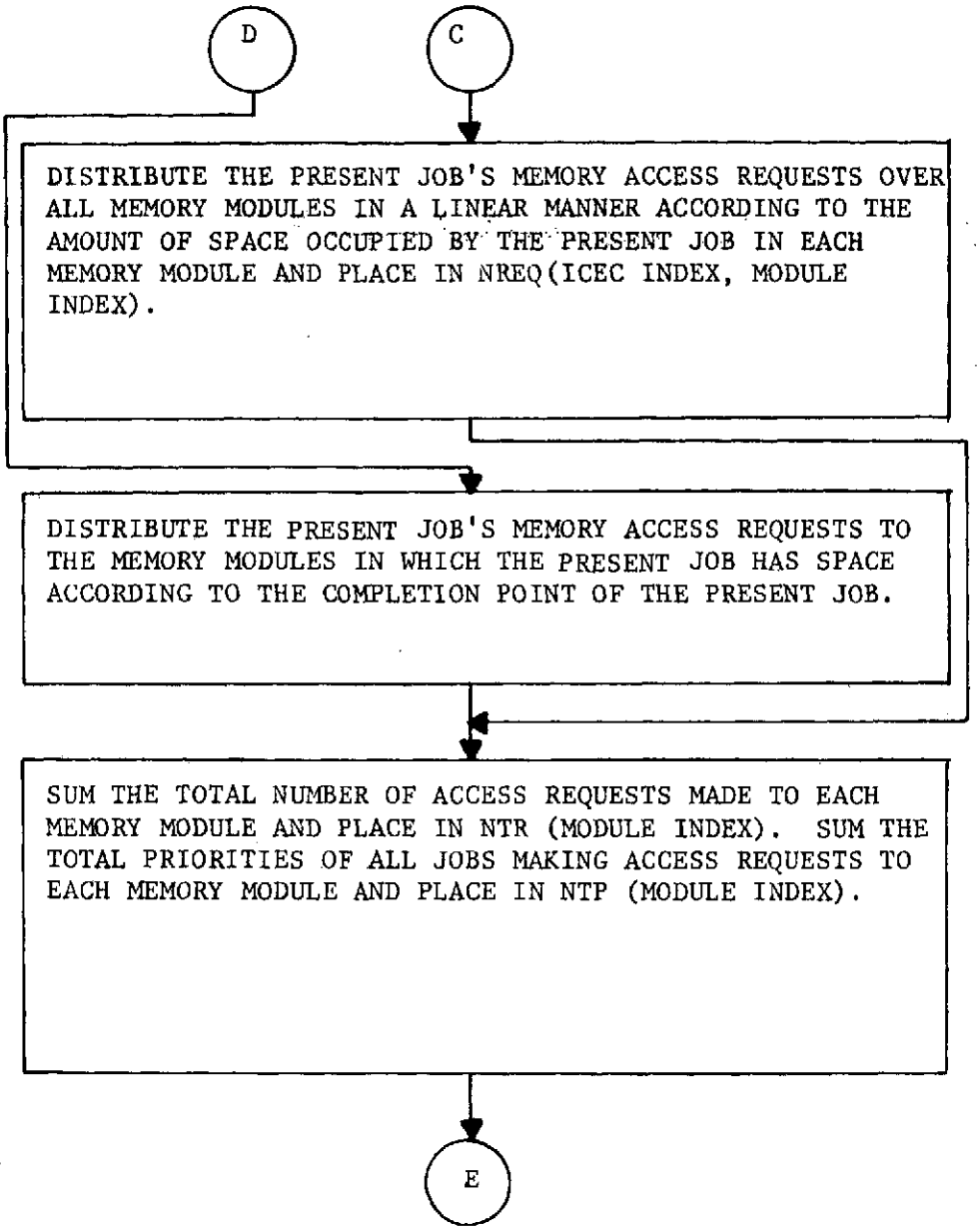


Fig.B-13 PEX subroutine (sheet 2 of 4)

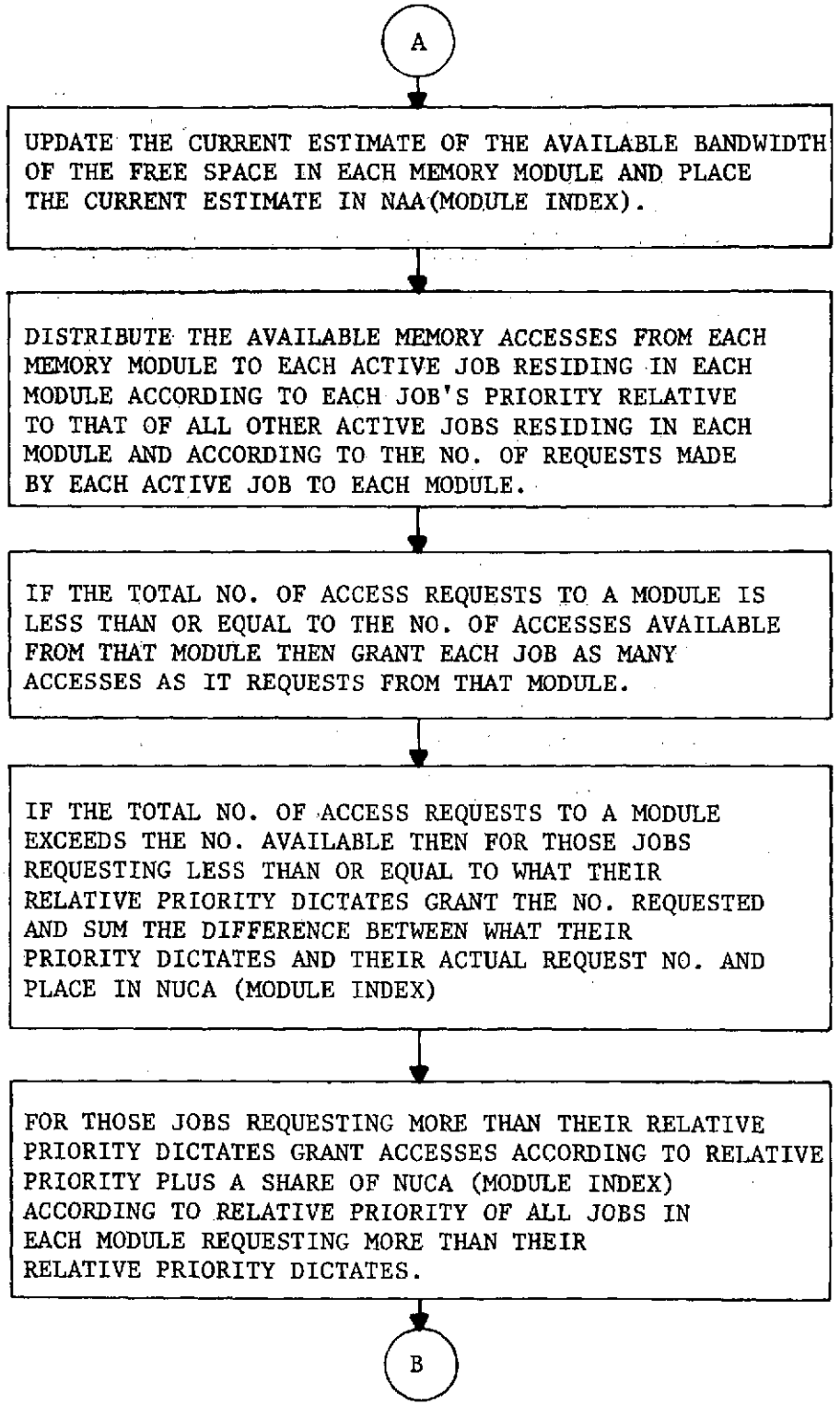


Fig. B-13 PEX subroutine (sheet 3 of 4)

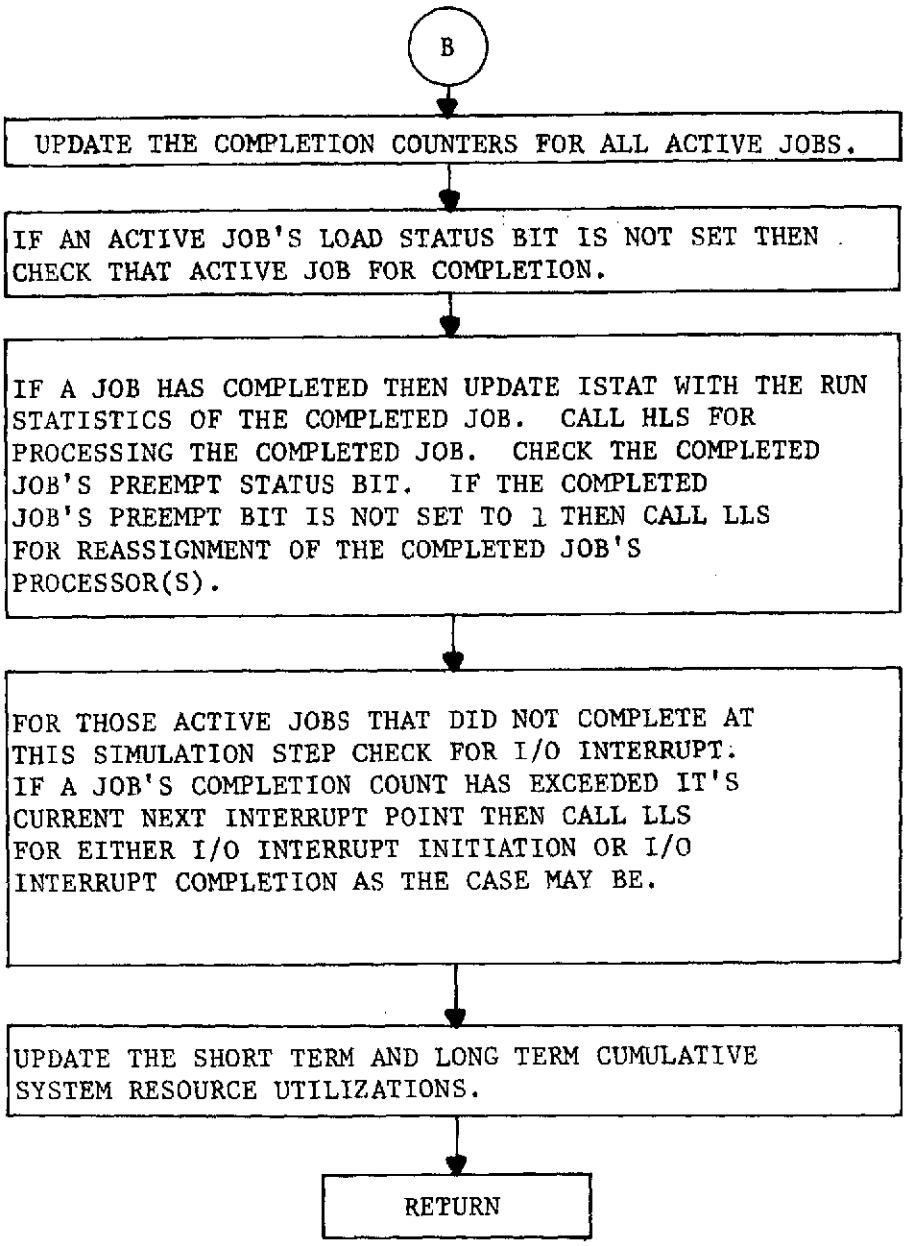


Fig. B-13 PEX subroutine (sheet 4 of 4)

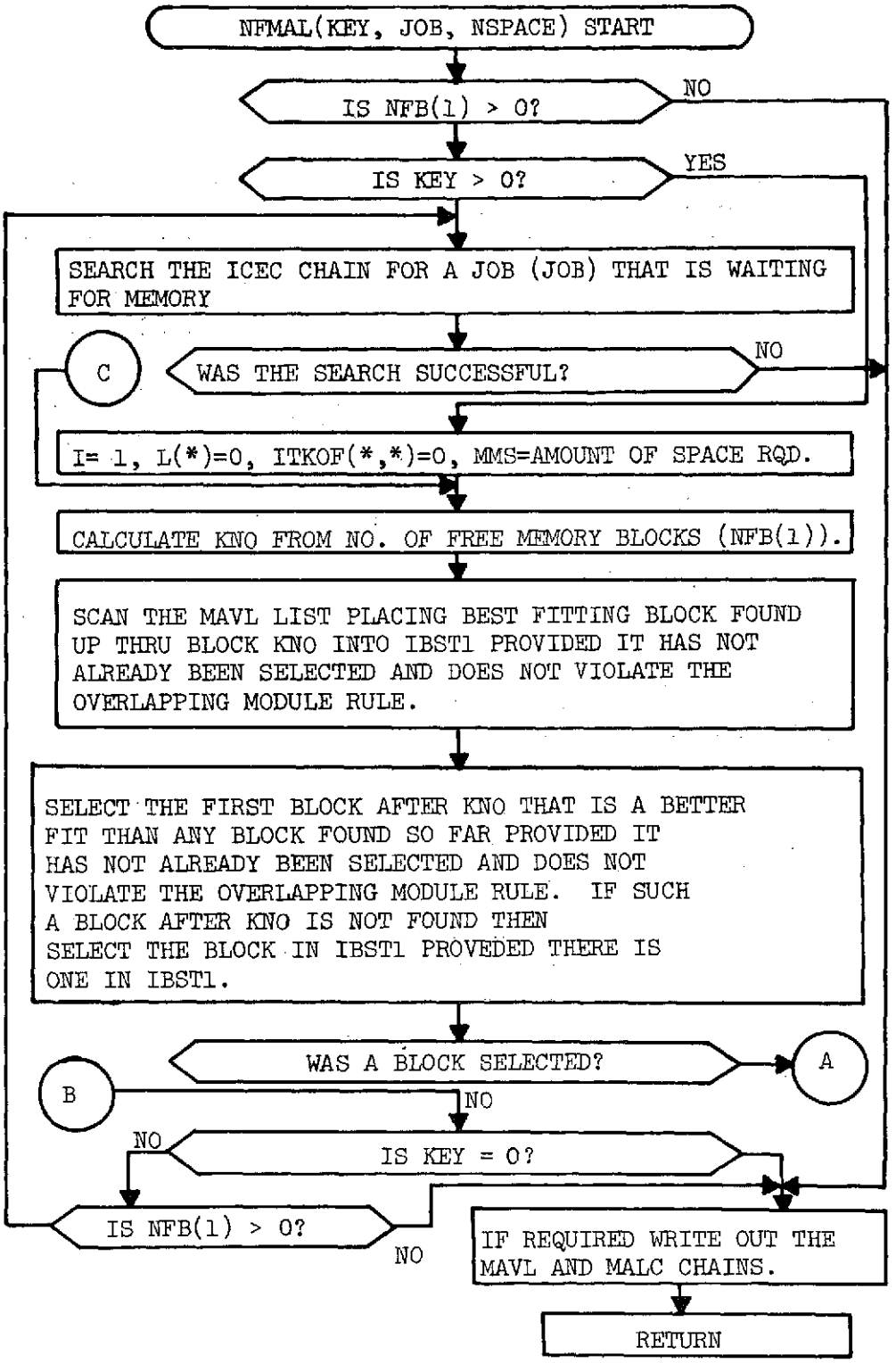


Fig. B-14 NFMAL subroutine (sheet 1 of 2)

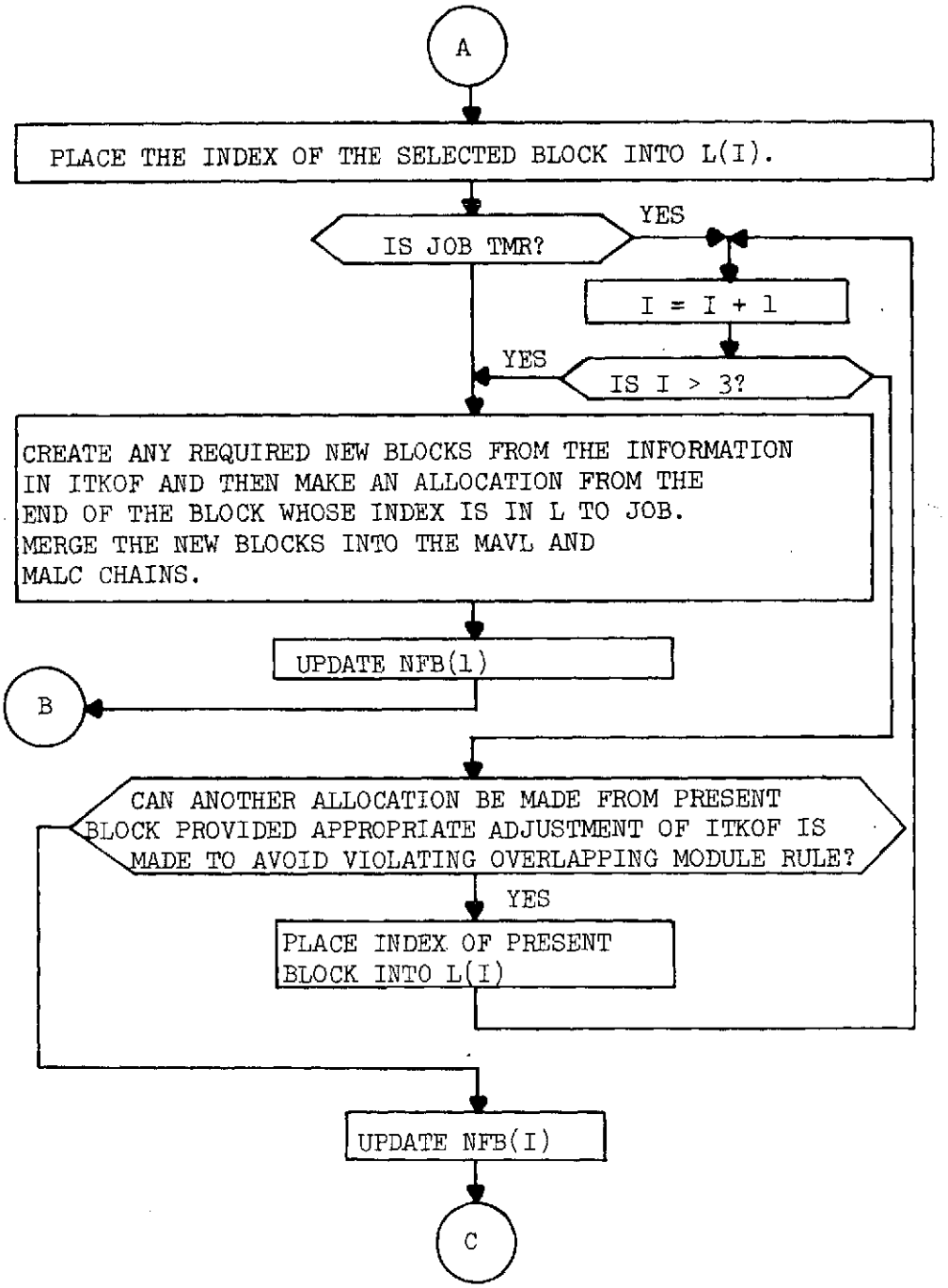


Fig. B-14 NFMAL subroutine (sheet 2 of 2)

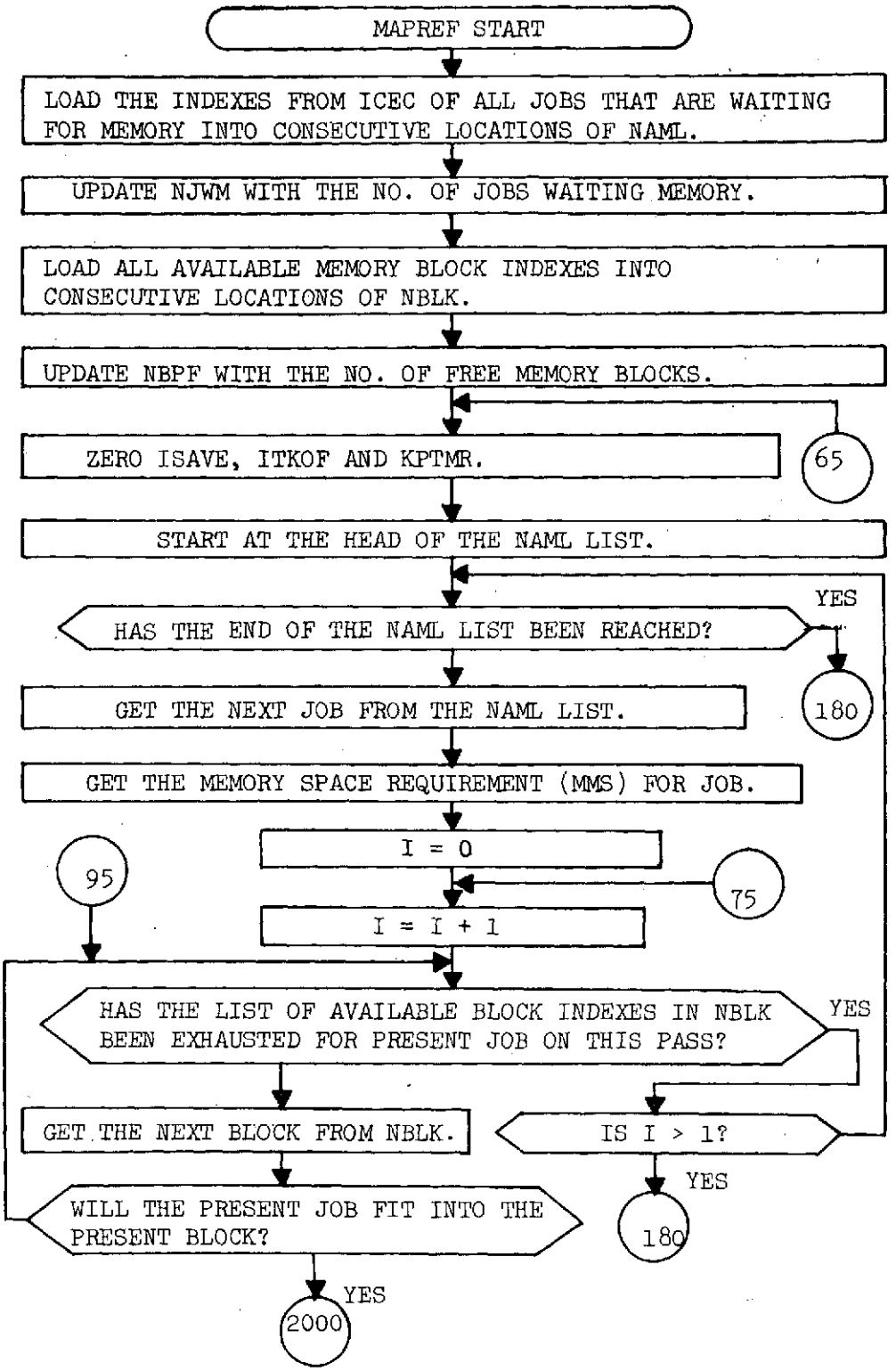


Fig. B-15 MAPREF subroutine (sheet 1 of 5)

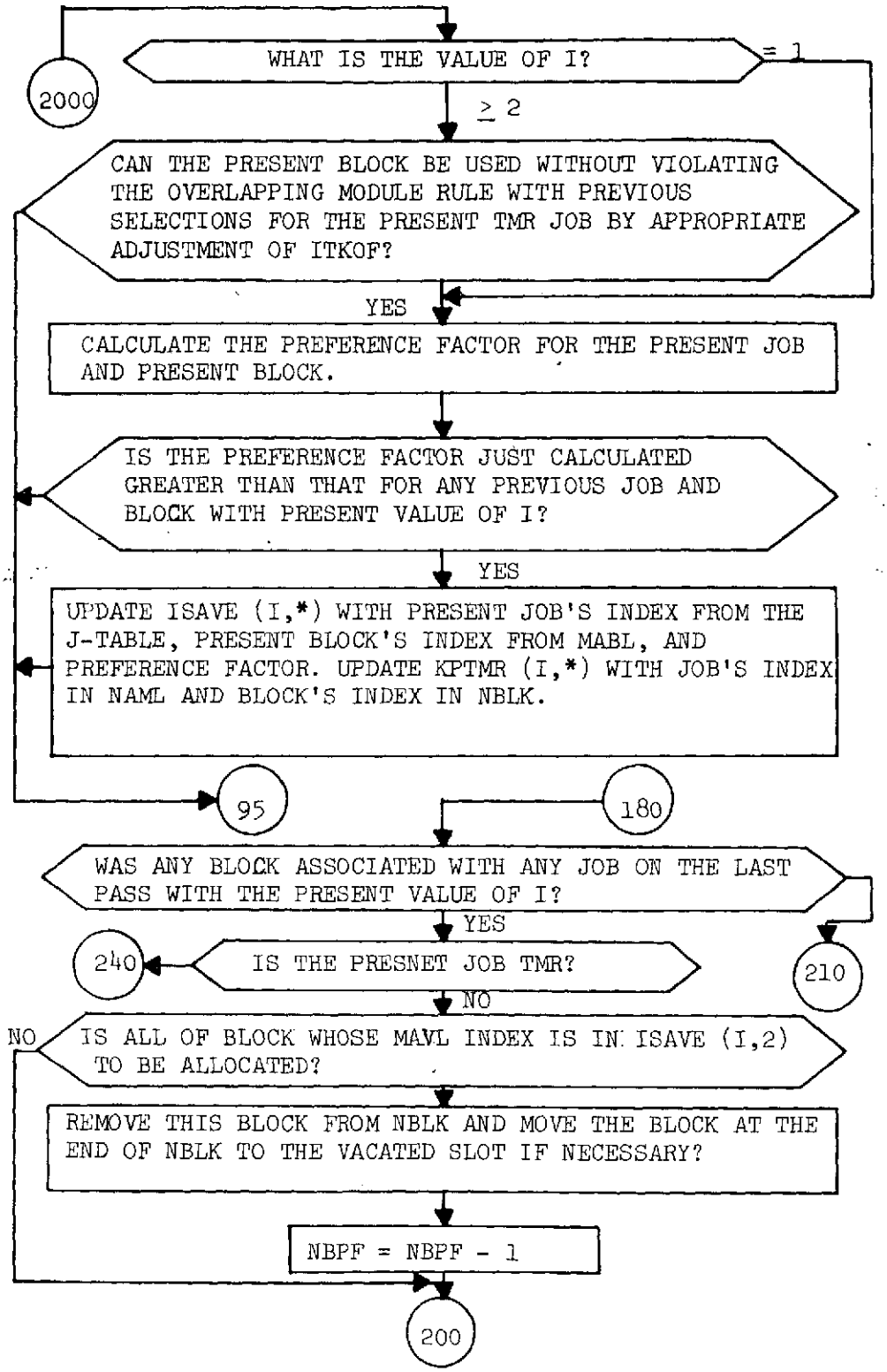


Fig. B-15 NAPREF subroutine (sheet 2 of 5)

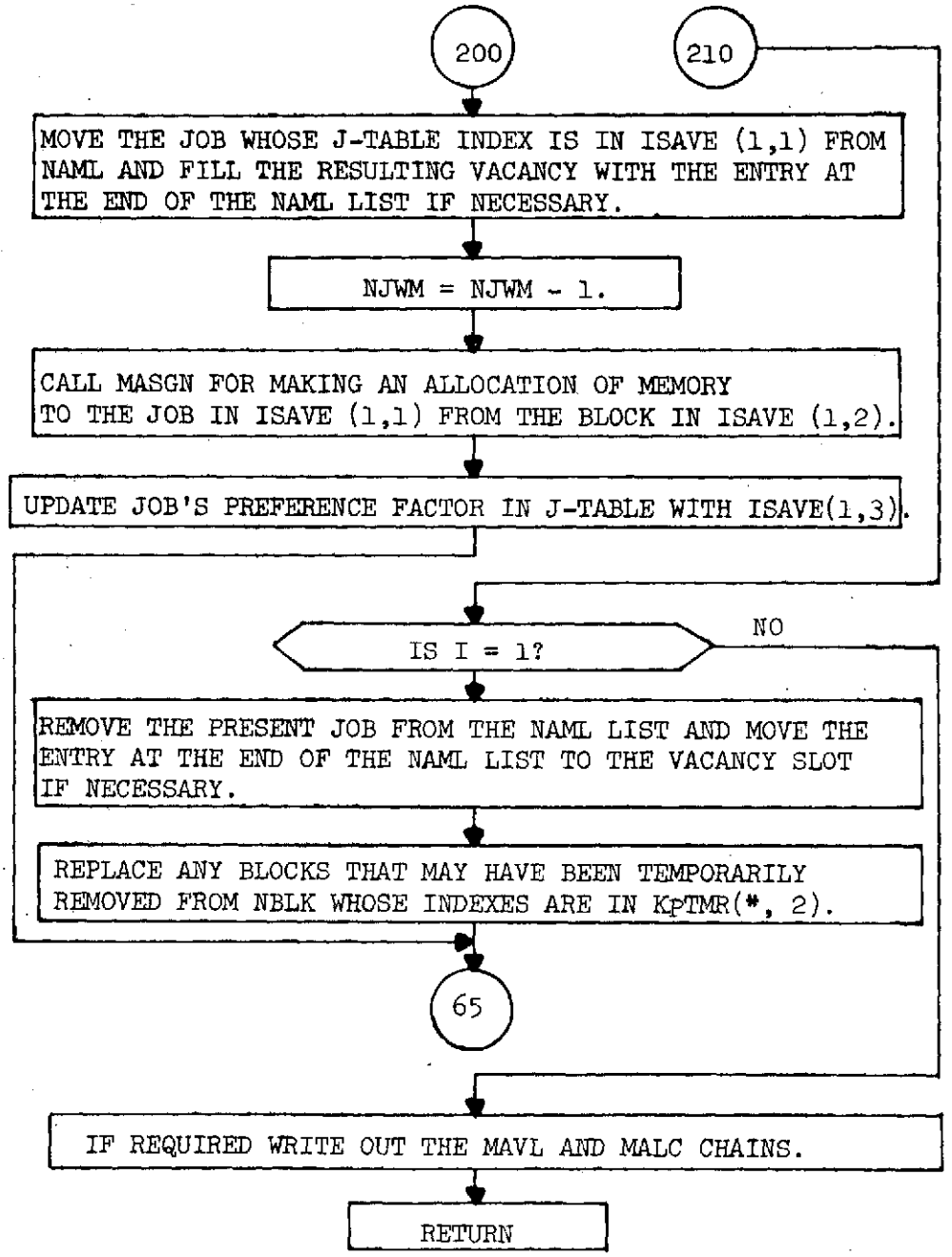


Fig. B-15 MAPREF subroutine (sheet 3 of 5)

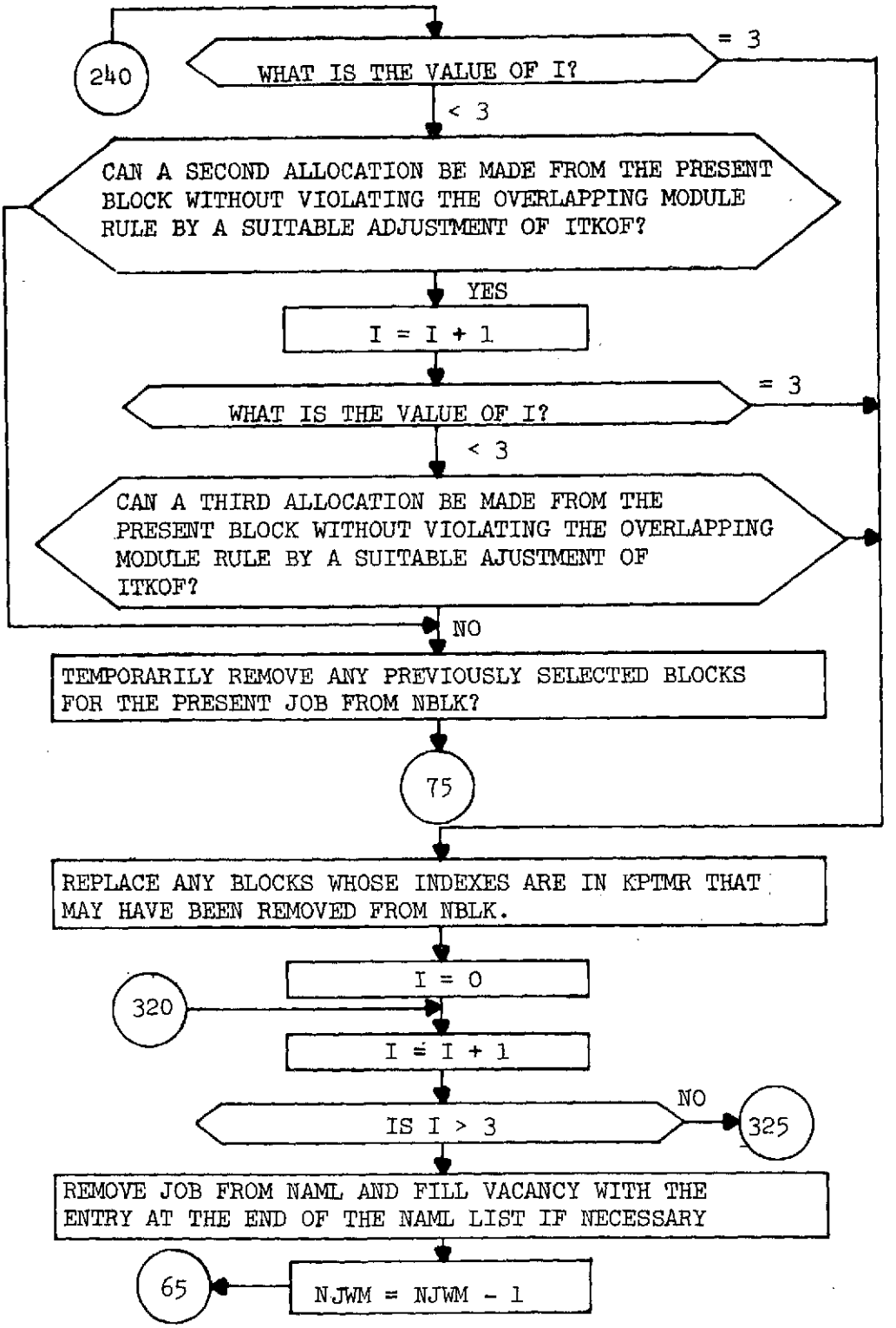


Fig. B-15 MAPREF subroutine (sheet 4 of 5)

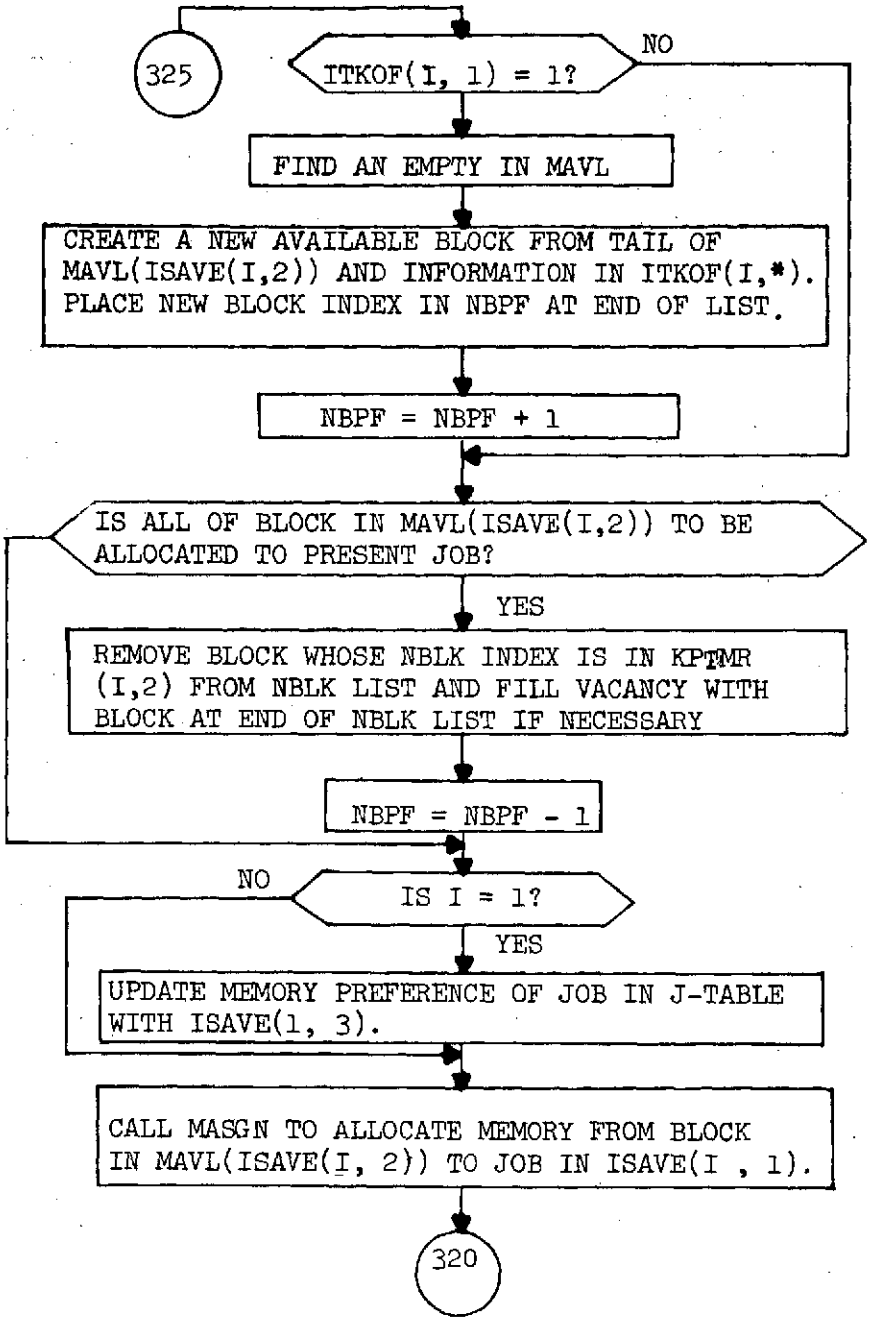


Fig. B-15 MAPREF subroutine (sheet 5 of 5)

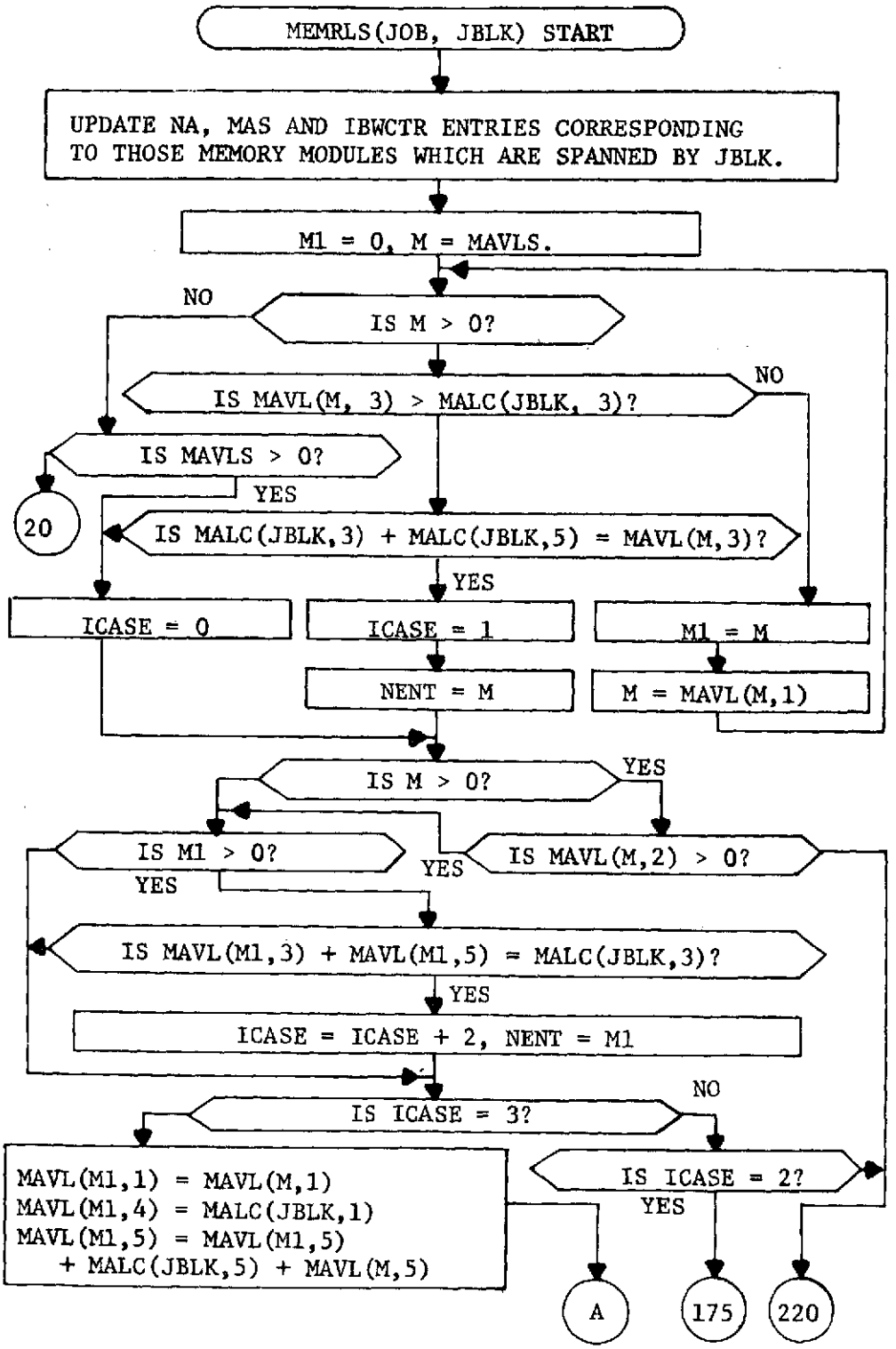


Fig. B-16 MEMRLS subroutine (sheet 1 of 4)

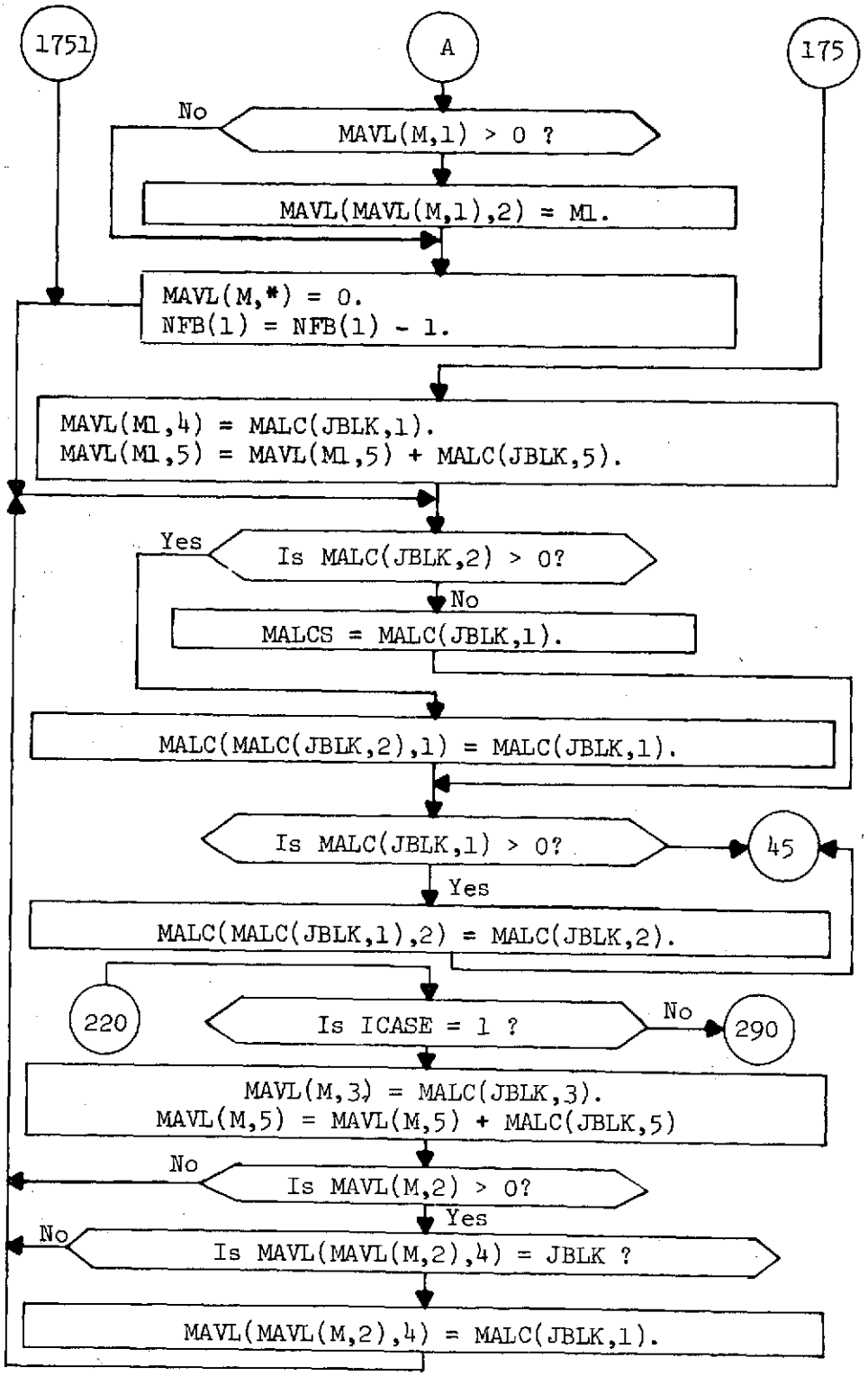


Fig. B-16 MEMRLS Subroutine (sheet 2 of 4)

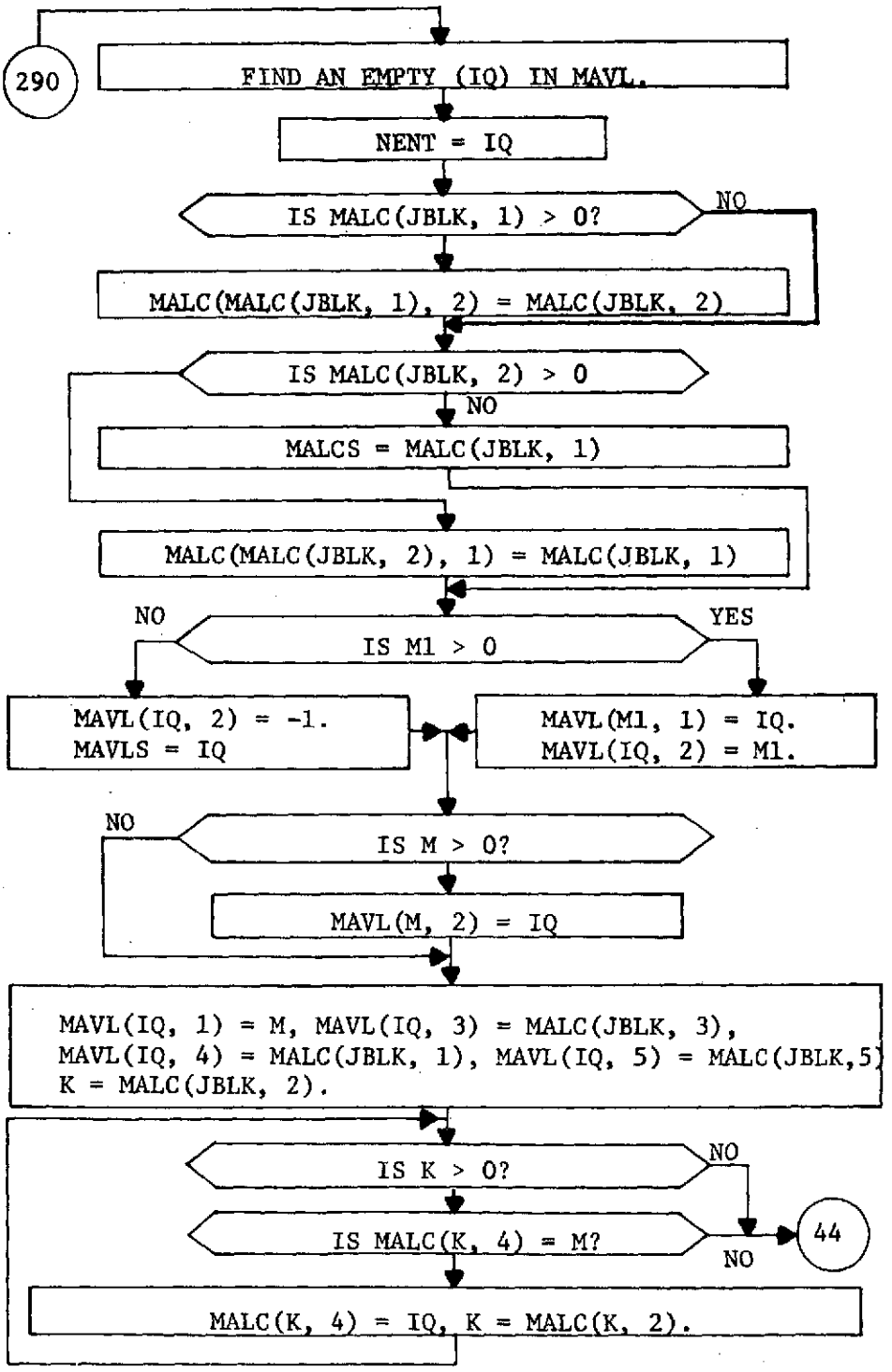


Fig. B-16 MEMRLS subroutine (sheet 3 of 4)

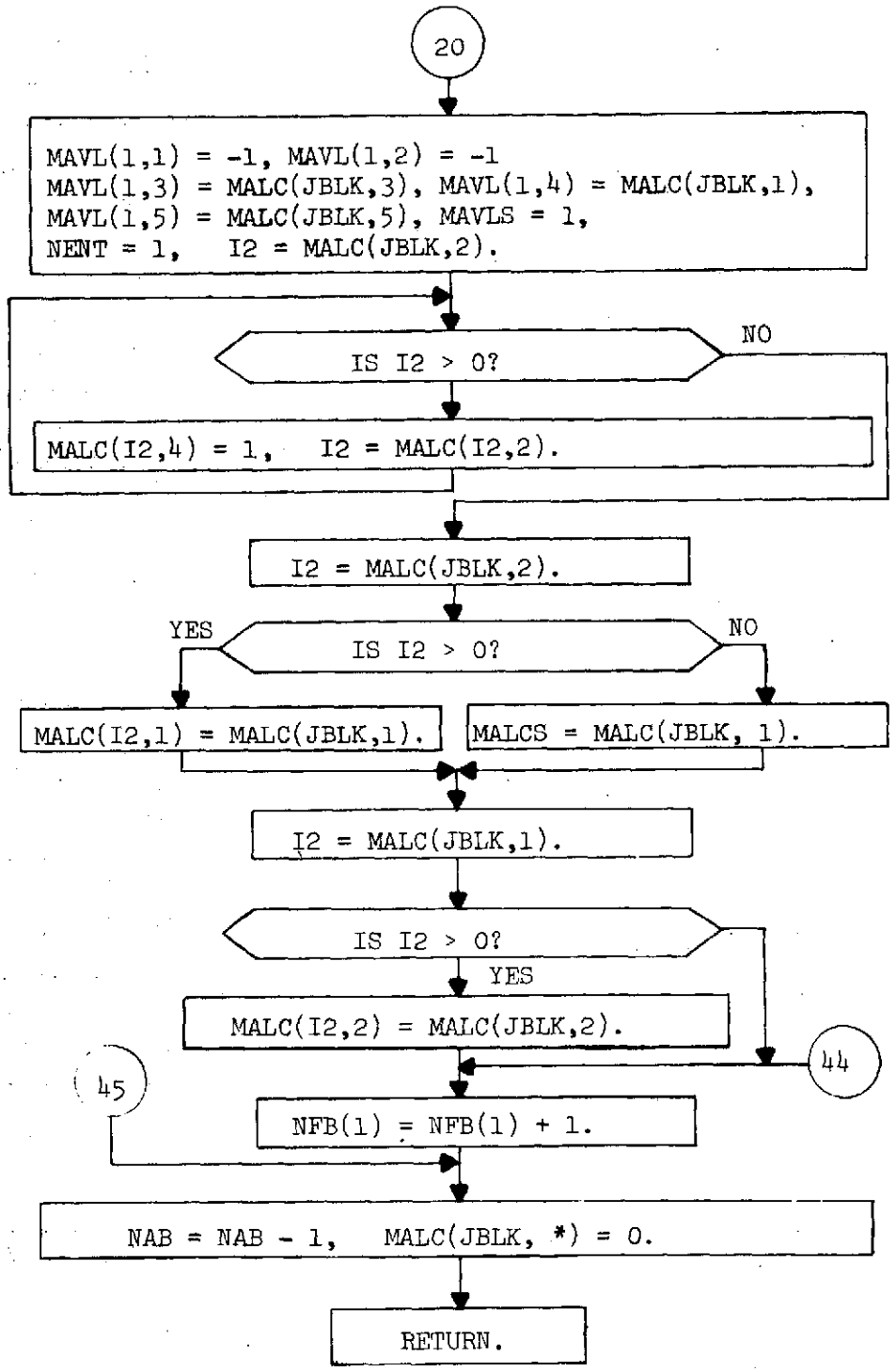


Fig. B-16 MEMRLS subroutine (sheet 4 of 4)

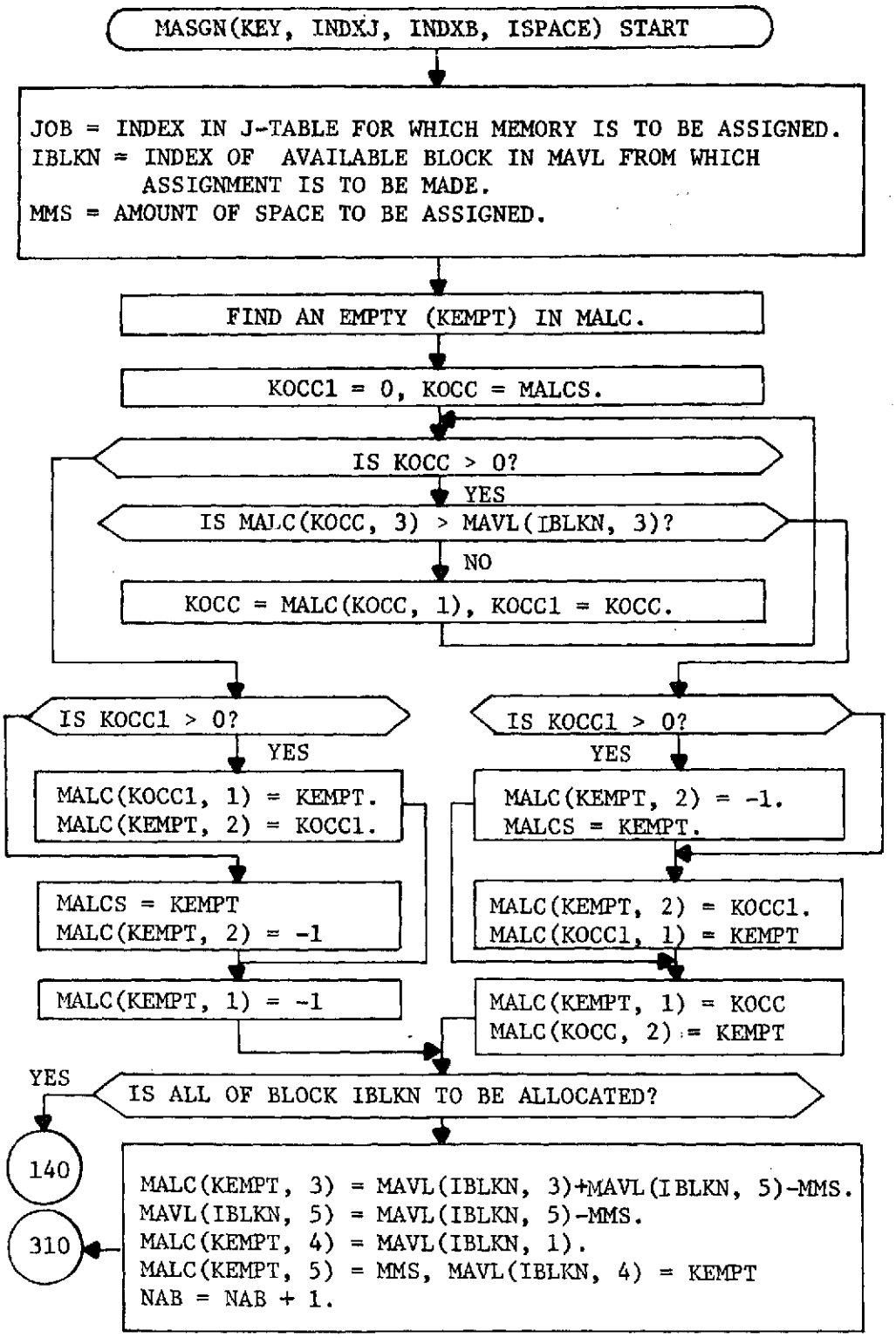


Fig. B-17 MASGN subroutine (sheet 1 of 2)

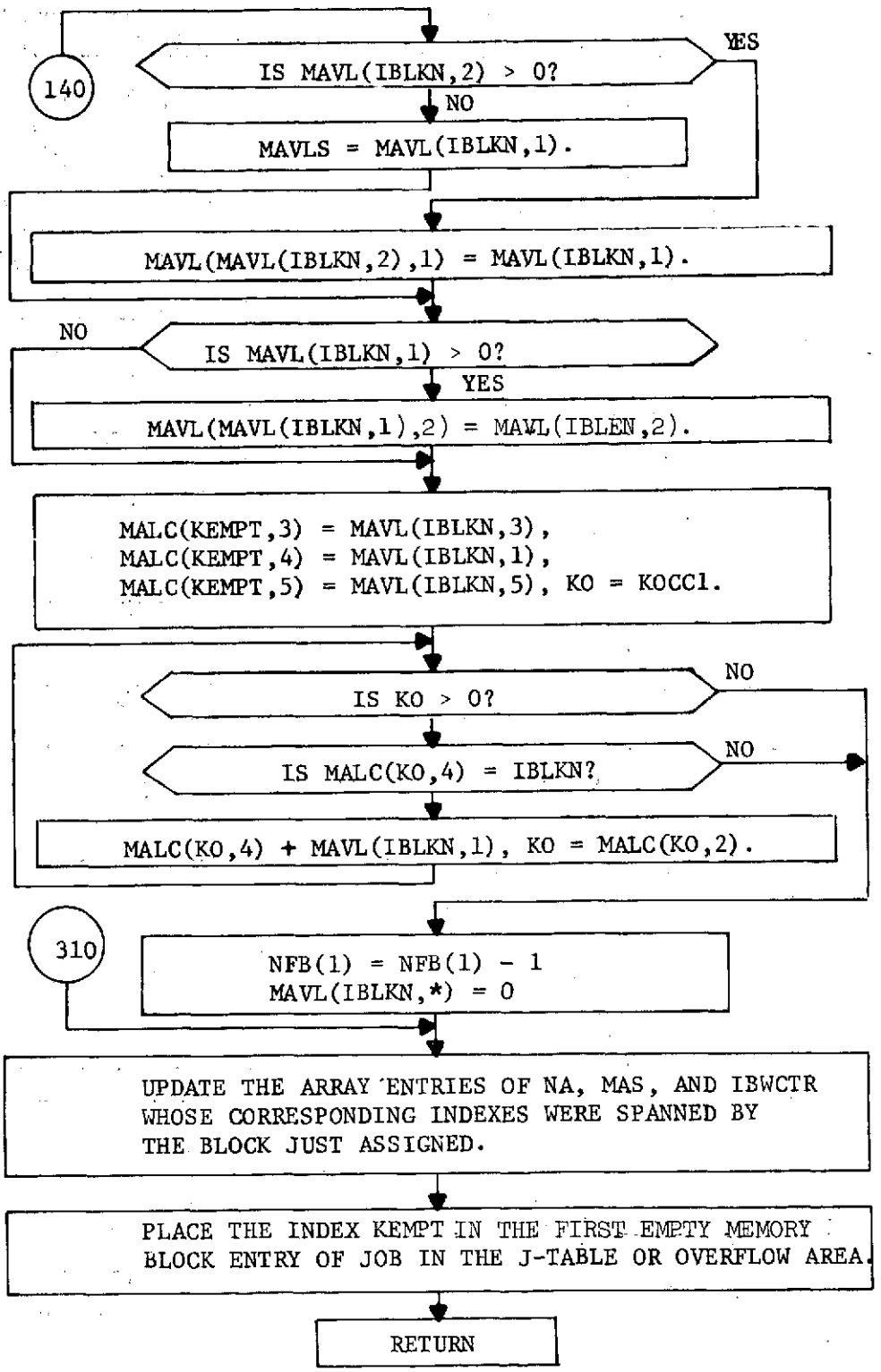


Fig. B-17 MASGN subroutine (sheet 2 of 2)

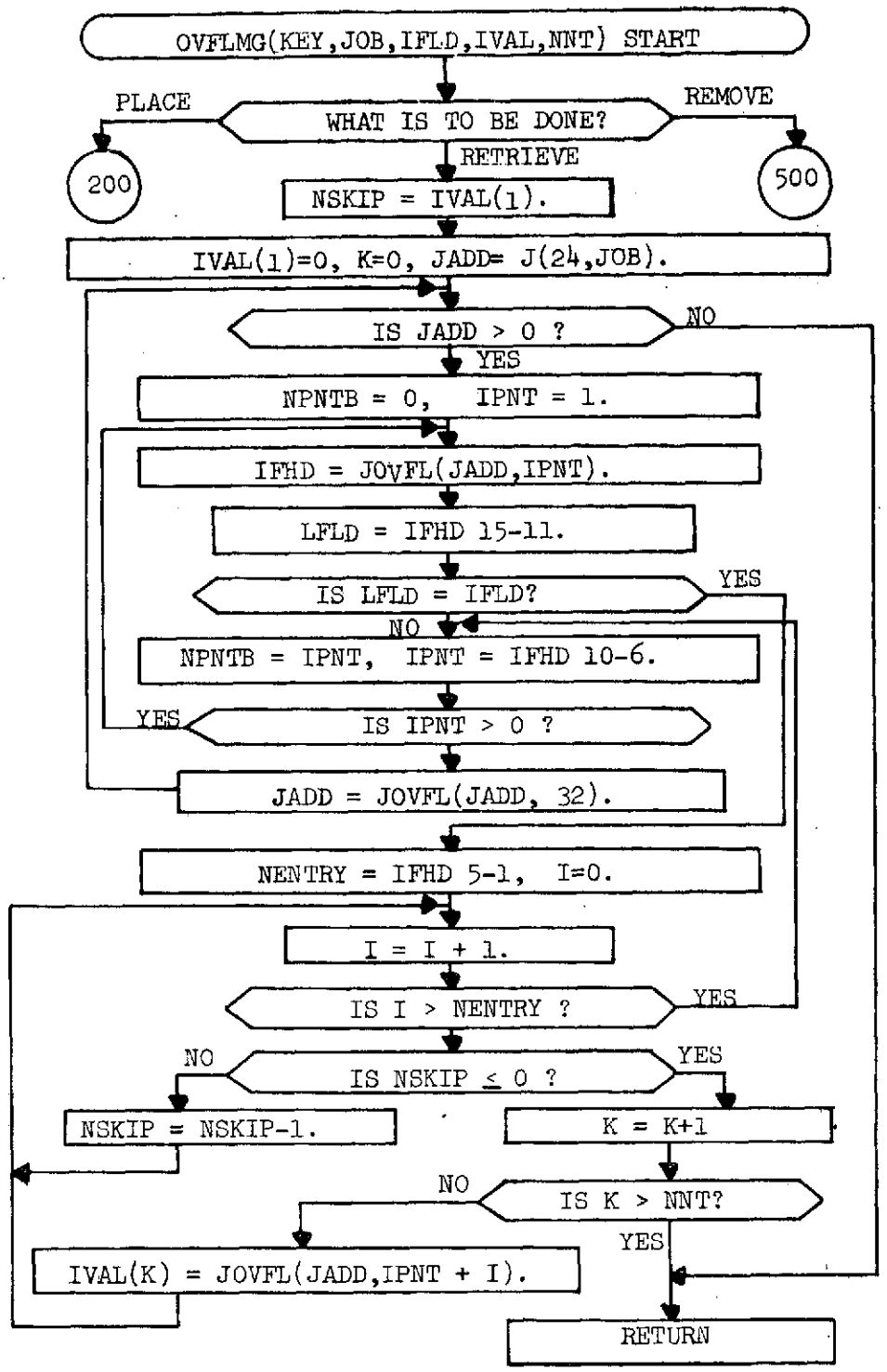


Fig. B-18 OVFLMG subroutine (sheet 1 of 5)

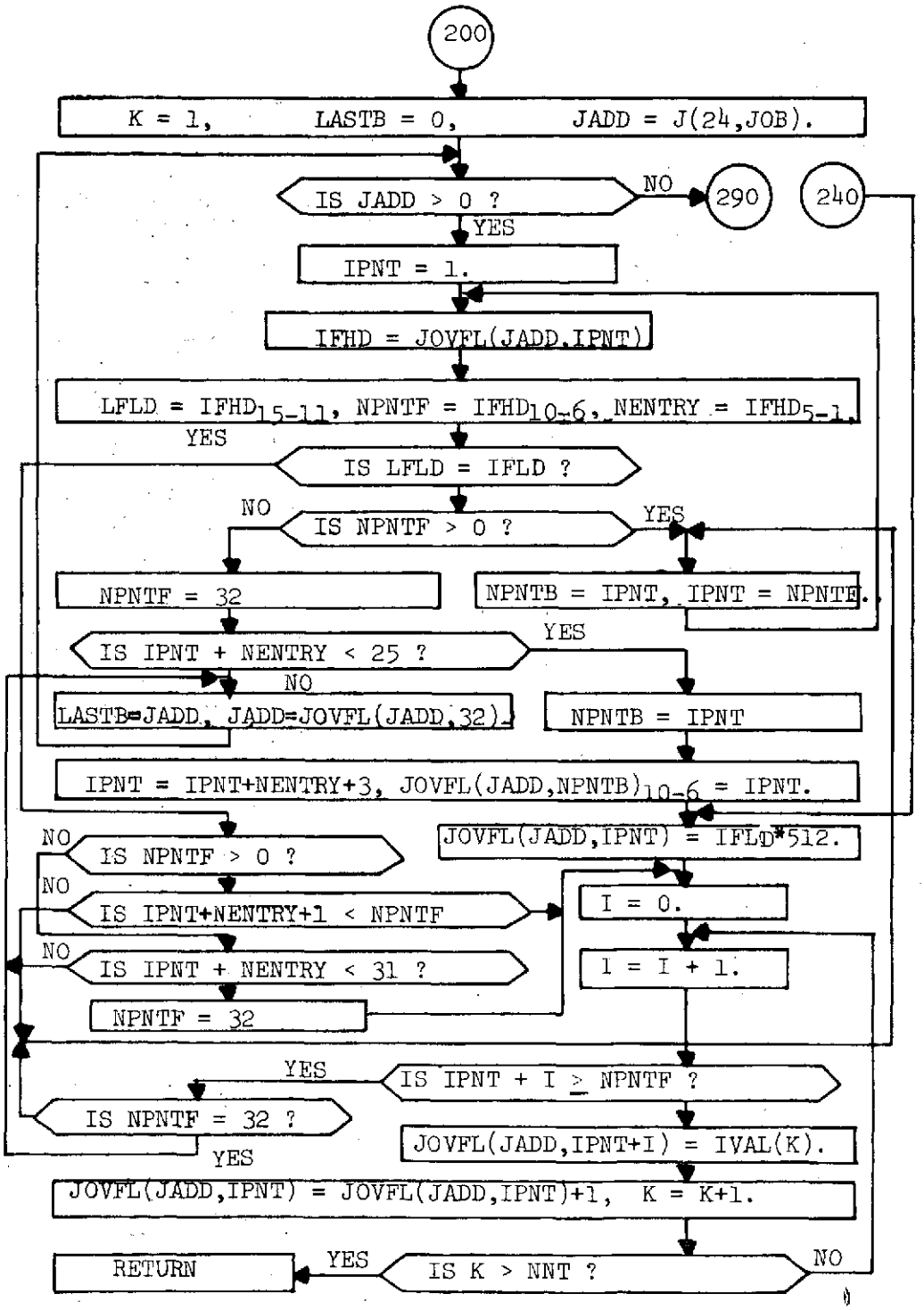


Fig. B-18 OVFLMG subroutine (sheet 2 of 5)

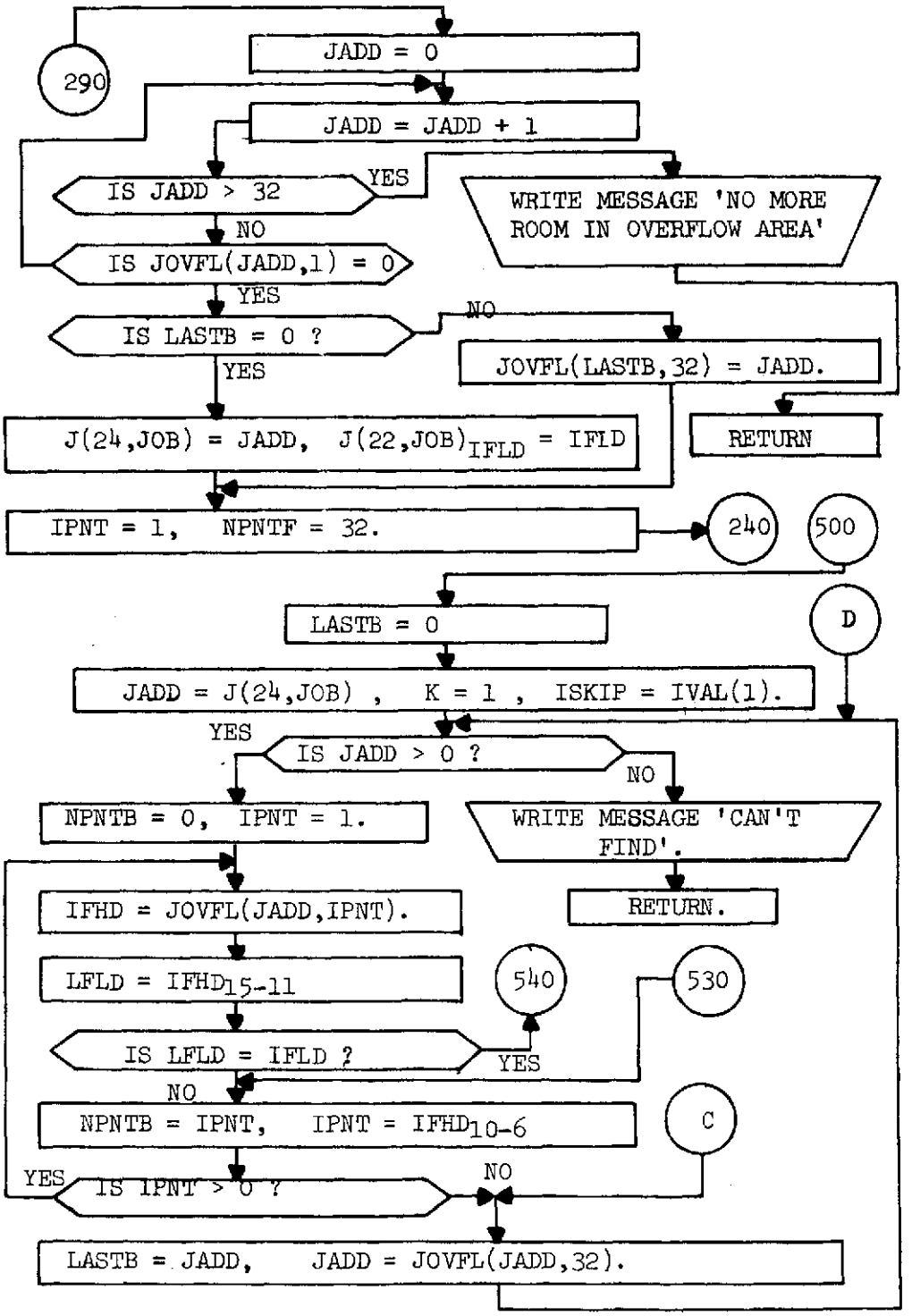


Fig. B-18 OVFLMG subroutine (sheet 3 of 5)

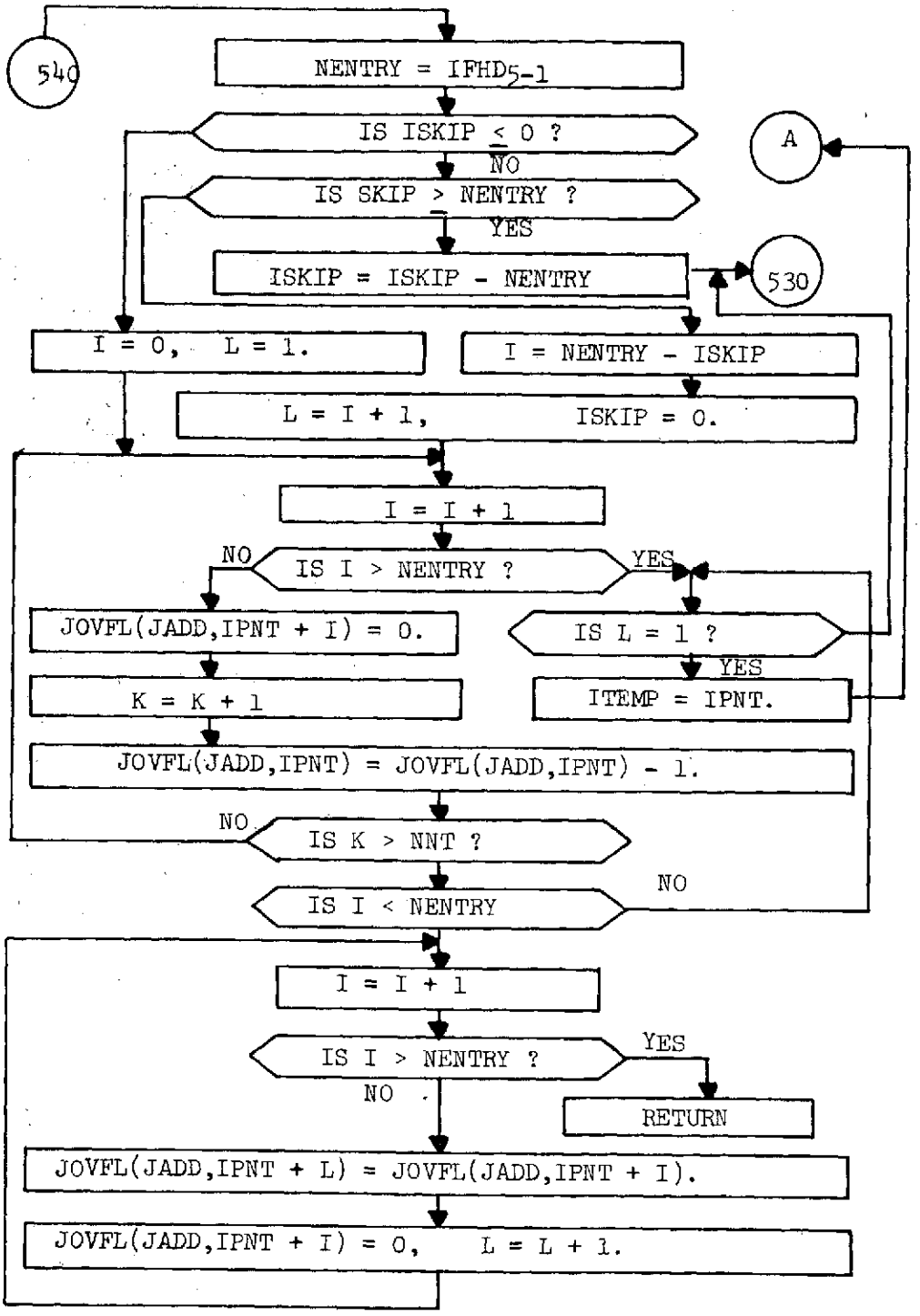


Fig. B-18 OVFLMG subroutine (sheet 4 of 5)

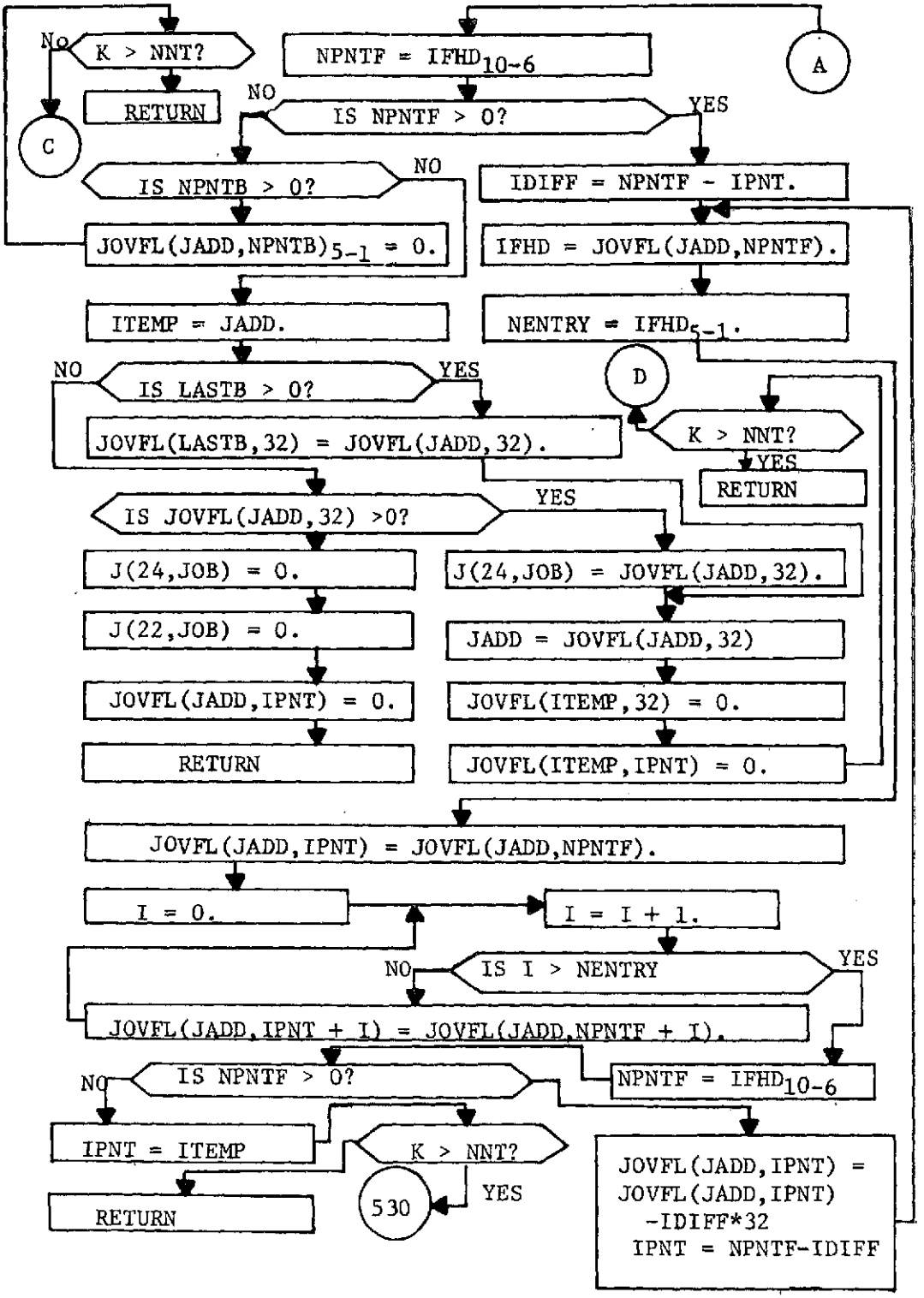


Fig.B-18 OVFLMG subroutine (sheet 5 of 5)

APPENDIX C

PROGRAM LISTINGS

**** WARNING ****
IN THE FOLLOWING PROGRAM LISTINGS THE NORMAL FORTRAN SEVENTY-TWO SPACE LINE HAS BEEN REPLACED BY A SIXTY SPACE LINE WHICH IS SUITABLE FOR AN EIGHT AND ONE-HALF INCH WIDE PAGE WITH A ONE AND ONE-HALF INCH LEFT HAND MARGIN AND A ONE INCH RIGHT HAND MARGIN. THEREFORE IN THESE LISTINGS THE FIRST CARD OF A FORTRAN STATEMENT BEGINS IN COLUMN SIXTEEN, PROCEEDS THROUGH COLUMN SEVENTY-FIVE AND CONTINUES ON COLUMN TWENTY-TWO OF THE FOLLOWING LINE AND ENDS ON COLUMN THIRTY-THREE OF THE FOLLOWING LINE PROVIDED THE ORIGINAL CARD HAD ANY NONBLANK CHARACTERS IN COLUMNS SIXTY-ONE THROUGH SEVENTY-TWO OR THE FOLLOWING CARD WAS A CONTINUATION OF THIS STATEMENT. OTHERWISE IT ENDS IN THE FIRST LINE OF THE STATEMENT IN THIS LISTING. THE SECOND AND SUBSEQUENT CARDS OF A STATEMENT BEGINS IN THE FIRST COLUMN AFTER THE END OF THE PRECEEDING CARD IN THESE LISTINGS. THE CONTINUATION SYMBOLS OF THE ORIGINAL CARD STATEMENTS HAVE BEEN REMOVED AND SUITABLE CONTINUATION SYMBOLS HAVE BEEN PLACED AS APPROPRIATE IN COLUMN TWENTY-ONE OF THESE LISTINGS

```
 BLOCK DATA
 COMMON/BLK1/IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQL
C,IDST
 COMMON/BLK1/INAME,IDATA,MJDT,IBST,IEXIT,IFLD
 COMMON/BLK15/KNAME(46)
 COMMON/BLK18/RUTL(15)
 DATA RUTL/15*0.0/
 DATA IASK,IDSH,LIND,LINA/'****','----',33*'----',33*'*
C***'/
 DATA IBLK,IDLR,IEQL,IDST/'    ','$   ','=   ','DIST'/
 DATA INAME,IDATA,MJDT,IBST/'NAME','DATA','MJDT','BST '
C/
 DATA IEXIT,IFLD/'EXIT','*FLD'/
 DATA KNAME/'CESZ','CON ','N1  ','FECZ','PCT ','P1  ',
C           'P2  ','Q1  ','Q2  ','Q3  ','R   ','S   ',
C'TA  ','MNBK',          'MMCT','MMST','MTTL','NI  ','N
CICS','NMOD','NPCL','NPCS','NCPU',   'IFBK','IRN1','IRN
C2','IRN3','IRN4','IRN5','IRN6','IRN7','IRN8',   'IRN9'
C,'MSPF','MAPF','RP01','RP02','RP03','RP04','RP05','RP0
C6',   'RP07','RP08','RP09','RP10','RP11'/
C
 END
 COMMON/BLK1/IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQL
C,IDST
 COMMON/BLK1/INAME,IDATA,MJDT,IBST,IEXIT,IFLD
 COMMON/BLK2/IWRD(100),FLD(50)
 COMMON /BLK3/ IDIST(25),F(25,41),NDIST
 COMMON/BLK4/NAME(75,8),DATA(75,9),NJOB,NJOBER
 DIMENSION NEXC(64)
 INTEGER*2 NEXC,NJOB,NJOBER
 COMMON/BLK5/ SET
 COMMON/BLK7/ JM(27,64)
 INTEGER*2 JM
 COMMON/BLK8/ TIME
 COMMON/BLK11/ RTIME
 COMMON/BLK12/PARM(35)
 COMMON/BLK13/RPAR(11)
 COMMON/BLK14/IFLAG(47),ITAG(46)
 COMMON/BLK15/KNAME(46)
 COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
CS(20),      IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
COV(10),MALC(256,5),     MALCS,MAS(24),MAVL(128,5),MAVL
CS,MODNM(24),MTP1(24),NA(24),        NAA(24),NAB,NAG,NA
CML(40),NBLK(128),NFB(3),NJWM,NSCHED,            NREQ(4
C0,24),NTP(24),NTR(24),NUCA(24)
```

```
      C
       INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
      CS,           ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
      CNM,MTPI,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
      CNREQ,NTP,NTR,NUCA
       COMMON/BLK17/ ISTCNT,ISTAT(6,400),IUTL(6,200)
       INTEGER*2 ISTAT,IUTL
       COMMON/BLK18/ AN,CUU,PUU,BWU,SPU,BCUU,BPUU,BBWU,BSPU,A
      CCUU,APUU,    ABWU,ASPU,BQSIZ,AQSIZ
      C
       DIMENSION IOPT(4)
       INTEGER*2 SET,TIME,HFIX,IFLAG,ITAG
       INTEGER PARM,GAUS/'GAUS'/
       DIMENSION NSST(7,100)
       INTEGER*2 NSST
       DIMENSION SMEAN(7)
       DATA IENTR,IDLET,ISTOP,IXEQ/'ENTR','DLET','STOP','XEQ
      C'/
       DATA ISIMN,IERAS,IPRT/'SIMN','ERAS','PRNT'/
       DATA IOPT/'STIM','FTIM','DELT','PDEL'/,LETL/'L   '/,IS
      CTR/'*   '/
       DATA IERR,INIT,ISYS/'*ERR','INIT','SYS '/
       INTEGER NCNT/0/
       DO 24 I=1,46
   24  IFLAG(I)=0
       IFLAG(47)=-1
    1  STRTIM=0.0
       FINTIM=10.0
       DELTA=0.01
       PRTDEL=0.5
       TIME=HFIX(STRTIM)
       RTIME=STRTIM
       NSSTC=0
    2  CALL ICARD
       SET=3
       IF(IWRD(1).NE.IDLR) GO TO 3
       IF(IWRD(2).EQ.IENTR) GO TO 4
       IF((IWRD(2).EQ.INIT).AND.(IWRD(3).EQ.ISYS)) CALL SETUP
       IF(SET.EQ.4) GO TO 100
       IF((IWRD(2).EQ.IERAS).AND.(IWRD(3).EQ.MJDT)) CALL CLEA
      CR
       IF(SET.EQ.4) GO TO 100
       IF((IWRD(2).EQ.IXEQ).AND.(IWRD(3).EQ.ISIMN)) GO TO 5
       IF((IWRD(2).EQ.IPRT).AND.(IWRD(3).EQ.MJDT)) CALL PRNT
       IF(SET.EQ.4) GO TO 100
       IF(IWRD(2).EQ.ISTOP) GO TO 99
```

```
      GO TO (1,2,3,100),SET
3     WRITE(6,201) IASK,IASK,IDSH
201   FORMAT(//,' ',A4,'ERROR',A4,'IMPROPERLY FORMATED,UNDEF
     CINED,OR MISSING COMMAND OR DATA CARD',A4,'SIMULATION T
     CERMINATED')
      GO TO 100
4     IF(IWRD(3).EQ.IDST) CALL STAT
      IF(IWRD(3).EQ.IBST) CALL BULK
      IF(IWRD(3).EQ.MJDT) CALL ENTER
      GO TO (1,2,3,100),SET
5     IF(IFLAG(47).NE.1) GO TO 98
      IF(IWRD(4).EQ.0) GO TO 21
      IF((IWRD(4).NE.LETL).OR.(IWRD(5).NE.ISTR).OR.(IWRD(6).
     CNE.IFLD)) GO TO 20
     C
      IF(IWRD(7).NE.1) GO TO 20
      IF(FLD(1).LT.1.0) GO TO 20
      IF(FLD(1).LE.4.0) GO TO 6
      FLD(1)=4.0
      WRITE(6,203) IASK,IASK,IDSH
203   FORMAT(//,' ',A4,'WARNING',A4,' ONLY 4 LOCAL OPTIONS A
     CVAILABLE AT EXECUTION',A4,'LOCAL OPTION SPEC. ASSUMED
     CTO BE L*4')
6     NOPT=IFIX(FLD(1))
      DO 19 I=1,NOPT
      CALL ICARD
      DO 7 J=1,4
      IF(IWRD(1).EQ.IOPT(J)) GO TO 9
      IF(IWRD(1).EQ.IDLR) GO TO 97
7     CONTINUE
8     WRITE(6,204) IASK,IASK,IDSH
204   FORMAT(//,' ',A4,'WARNING',A4,' INVALID OPTION CARD',A
     C4,'PASSED')
      GO TO 19
9     IF((IWRD(2).NE.IEQL).OR.(IWRD(3).NE.IFLD).OR.(IWRD(4).
     CNE.1)) GO TO 8
     C
      GO TO(10,12,15,17),J
10    IF(FLD(1).GE.0.0) GO TO 11
      WRITE(6,205) IASK,IASK,IOPT(J),IDSH
205   FORMAT(//,' ',A4,'WARNING',A4,' ILLEGAL VALUE SPECIFIE
     CD FOR ',2A4, 'DEFAULT OPTION USED')
     C
      GO TO 19
11    STRTIM=FLD(1)
      GO TO 19
```

```
12    IF(FLD(1).GT.0.0) GO TO 13
      WRITE(6,205) IASK,IASK,IOPT(J),IDSH
      GO TO 14
13    FINTIM=FLD(1)
14    IF(STRTIM.LT.FINTIM) GO TO 19
      WRITE(6,206) IASK,IASK,IDSH
206   FORMAT(//,' ',A4,'WARNING',A4,' STIM HAS BEEN SPECIFIE
     CD TO BE GREATER THAN FTIM',A4,'FTIM SET TO 2*STIM')
     C
      FINTIM=2.0*STRTIM
      GO TO 19
15    IF(FLD(1).GE.0.001) GO TO 16
      WRITE(6,205) IASK,IASK,IOPT(J),IDSH
      GO TO 19
16    DELTA=FLD(1)
      GO TO 19
17    IF(FLD(1).GE.0.0) GO TO 18
      WRITE(6,205) IASK,IASK,IOPT(J),IDSH
      GO TO 19
18    PRTDEL=FLD(1)
19    CONTINUE
      GO TO 21
20    WRITE(6,207) IASK,IASK,IDSH
207   FORMAT(//,' ',A4,'WARNING',A4,' INVALID LOCAL OPTION S
     CPEC.',A4,'ALL DEFAULT VALUES IN EFFECT')
     C
21    WRITE(6,208) LIND,LIND
208   FORMAT(' ',33A4)
      WRITE(6,209) IASK,IASK,IASK,IASK,STRTIM,FINTIM,DELTA,P
     CRTDEL
209   FORMAT(//,' ',A4,' BEGINNING EXECUTION ',A4,//,' ',A4,
     C' VALUES OF LOCAL OPTIONS ',A4,//,10X,'STIM =',F8.3,/,
     C10X,'FTIM =',F8.3,/,10X, 'DELT =',F8.3,/,10X,'PDEL =',
     CF8.3,//)
      WRITE(6,212) IASK,IASK
212   FORMAT(//,' ',A4,' VALUES OF SYSTEM PARAMETERS ',A4)
      DO 25 I=1,35
      IF(ITAG(I).EQ.0) WRITE(6,213) KNAME(I),PARM(I),ISTR
25    IF(ITAG(I).NE.0) WRITE(6,213) KNAME(I),PARM(I)
213   FORMAT(10X,A4,' =',I10,A4)
      DO 26 I=36,46
      IF(ITAG(I).EQ.0) WRITE(6,214)KNAME(I),RPAR(I-35),ISTR
26    IF(ITAG(I).NE.0)WRITE(6,214) KNAME(I),RPAR(I-35)
214   FORMAT(10X,A4,' =',F14.4,A4)
      WRITE(6,208) LIND,LIND
      PRTTIM=STRTIM+PRTDEL
```

```
      RTIME=STRTIM
      DO 300 KKK=1,25
      IF(IDIST(KKK).EQ.GAUS) GO TO 301
300   CONTINUE
301   IDXGAU=KKK
      CALL UNMIX(3,2,PARM(12),IWRT3)
1000  WRITE(6,1100)
1100  FORMAT(' MAIN WILL NOW CALL HLS FOR INITIAL J-TABLE SC
     CAN')
1001  CALL HLS(1,0)
22    IF(RTIME.LT.PRTTIM) GO TO 23
      CALL PRNT
      PRTTIM=PRTTIM+PRTDEL
23    IPASS(6)=TIME
      NSSTC=NSSTC+1
      NSST(1,NSSTC)=TIME
      NSST(2,NSSTC)=NJWM
      NSST(3,NSSTC)=NFB(1)
      NSST(4,NSSTC)=IPASS(10)
1003  IF(PARM(24)) 1010,1005,1010
1005  WRITE(6,1110) TIME
1110  FORMAT(' TIME = ',I5,' MAIN TO CEC FOR NEW SCHED,NFBK'
     C)
1006  CALL CEC(2,0)
      NSCHED=0
1009  WRITE(6,1160) TIME
1160  FORMAT(' TIME = ',I5,' MAIN TO NFMAL')
      CALL NFMAL(1,0,0)
1007  WRITE(6,1200) TIME
1200  FORMAT(' TIME = ',I5,' MAIN TO HLS FOR PROCESSOR ASN.,
     CNFBK')
      CALL HLS(2,0)
      GO TO 1029
1010  WRITE(6,1113) TIME
1113  FORMAT(' TIME = ',I5,' MAIN TO MAPREF')
      CALL MAPREF
      WRITE(6,1120) TIME
1120  FORMAT(' TIME =',I5,' MAIN TO CEC FOR NEW SCHEDULE,FBK
     C')
      CALL CEC(2,0)
      NSCHED=0
1012  WRITE(6,1210) TIME
1210  FORMAT(' TIME = ',I5,' MAIN TO HLS FOR PROCESSOR ASN.,
     CFBK')
      CALL HLS(2,0)
1029  NSST(5,NSSTC)=NJWM
```

```
      NSST(6,NSSTC)=NFB(1)
      NSST(7,NSSTC)=IPASS(10)
1030  IF(IPASS(3).GT.0) GO TO 1033
      IF(NCNT.GT.0) GO TO 1031
      NCNT=NCNT+1
      IF(IPASS(1).GT.0) GO TO 1003
      GO TO 1032
1031  NCNT=0
      WRITE(6,1400)
1400  FORMAT(' ****ERROR**** SCHEDULER HUNG UP, NO ACTIVE JO
     CBS BUT'/     ' JOBS IN ICEC ARE NOT BEING SCHEDULED')
     C
1032  CALL FEC(3,0,0,0)
      IF(IPASS(8).EQ.-1) GO TO 99
      LLOW=(TIME+19)/20
      LHIGH=IPASS(8)
      RTIME=FLOAT(LHIGH)
      AN=RTIME
      LHIGH=LHIGH/20
      IF(LLOW.GT.LHIGH) GO TO 1710
      DO 1705 KKK=LLOW,LHIGH
      IUTL(1,KKK)=KKK*20
      DO 1705 KK=2,6
      IUTL(KK,KKK+100)=0
1705  IUTL(KK,KKK)=0
1710  WRITE(6,1220) TIME,IPASS(8)
1220  FORMAT(' MAIN WILL NOW ADVANCE CLOCK FROM',I6,' TO',I6
     C)
      TIME=IPASS(8)
      CALL FEC(2,0,0,0)
      GO TO 1003
1140  FORMAT(/,' MAIN WILL NOW CALL PEX, TIME=',I8)
1333  CALL PEX(IDXGAU)
1500  FORMAT(10(2X,I4))
1505  FORMAT(16(2X,I4))
1510  FORMAT(' IPASS(1-4),TIME',5(2X,I5))
      TIME =TIME+HFIX(DELTA)
      RTIME=RTIME+DELTA
      IF(TIME.GE.0) GO TO 50
      CALL RESET
50    IF(RTIME.LE.FINTIM) GO TO 1040
      IF(IWRT3.EQ.1) CALL PRNT
      WRITE(6,1520)
1520  FORMAT('1')
      WRITE(6,1525)
1525  FORMAT(7X,'JOB',3X,'IN ICEC',5X,'CMPLT',2X,'MIN PROS',
```

```
     C              1X,'TARG TIME',2X,'ACT PRDS',/)
     C
1530  FORMAT(6(4X,I6))
      LANUM=0
      LASUM=0
      LASQR=0
      LAMAX=0
      IRNUM=0
      IRSUM=0
      IRSQR=0
      IRMAX=0
      DO 375 KK=1,64
375   NEXC(KK)=0
      DO 400 KK=1,ISTCNT
      WRITE(6,1530) (ISTAT(IX,KK),IX=1,6)
      NEXC(ISTAT(1,KK))=NEXC(ISTAT(1,KK))+1
      IDIFF=ISTAT(3,KK)-ISTAT(5,KK)
      IF(IDIFF) 390,400,380
380   LANUM=LANUM+1
      LASUM=LASUM+IDIFF
      LASQR=LASQR+IDIFF*IDIFF
      IF(IDIFF.GT.LAMAX) LAMAX=IDIFF
      GO TO 400
390   IDIFF=-IDIFF
      IRNUM=IRNUM+1
      IRSUM=IRSUM+IDIFF
      IRSQR=IRSQR+IDIFF*IDIFF
      IF(IDIFF.GT.IRMAX) IRMAX=IDIFF
400   CONTINUE
      IF(LANUM.EQ.0) GO TO 405
      MNLA=LASUM/LANUM
      LAVAR=(LASQR-2*MNLA*LASUM)/LANUM+MNLA*MNLA
      WRITE(6,1560) LANUM,LAMAX,MNLA,LAVAR
1560  FORMAT(' NO. LATE JOBS,MAX LATE,MEAN LATE,AND VAR. ARE
     C',4(3X,I6))
405   IF(IRNUM.EQ.0) GO TO 410
      MNIR=IRSUM/IRNUM
      IRVAR=(IRSQR-2*MNIR*IRSUM)/IRNUM+MNIR*MNIR
      WRITE(6,1570) IRNUM,IRMAX,MNIR,IRVAR
1570  FORMAT(/,' NO.EARLY JOBS,MAX EARLY,MEAN&VAR.ARE',4(3X,
     CI6))
410   ITIME=TIME
      ARRVLM=FLOAT(ITIME)/FLOAT(ISTCNT)
      WRITE(6,1580) ISTCNT,ARRVLM
1580  FORMAT(/,' TL.NO.JOBS RUN&MN.INTCMP.TIME ARE',3X,I6,3X
     C,F10.2)
```

```
      WRITE(6,1585)
1585  FORMAT(//,' RUN FREQUENCY FOR EACH JOB')
      WRITE(6,1590)
1590  FORMAT(/,3X,'JOB LOC.',3X,'NO. RUNS',6X,'LDACCS',6X,
     C            'NPACCS',6X,'NTACCS',9X,'DMD')
     C
      TDMD=0.0
      LDAT=0
      NPAT=0
      NTAT=0
      DO 415 KKK=1,NJOBER
      LDACCS=0
      NPACCS=0
      IND=JM(17,KKK)
      CALL UNMIX(11,0,IND,MMS)
      IND=JM(27,KKK)
      CALL UNMIX(11,10,IND,ITMR)
      CALL UNMIX(10,9,IND,IRES)
      CALL UNMIX(2,1,IND,INT)
      CALL UNMIX(14,13,IND,LOAD)
      CALL UNMIX(5,2,IND,IND53)
      IF(LOAD.EQ.1) LDACCS=LDACCS+JM(18,KKK)
      IF(IRES.EQ.0) GO TO 411
      IF(NEXC(KKK)) 412,412,4121
411   LDACCS=LDACCS+IFIX(RPAR(8)*(MMS*PARM(14)*NEXC(KKK)))
412   IF(INT.NE.1.AND.IND53.NE.1) GO TO 413
4121  LDACCS=LDACCS+IFIX(RPAR(8)*(MMS*PARM(14)))
413   JPACCS=JM(9,KKK)+JM(15,KKK)
      NPACCS=JPACCS*NEXC(KKK)
      IF(INT.EQ.1) NPACCS=NPACCS+JM(18,KKK)
      IF(ITMR.EQ.0) GO TO 414
      LDACCS=3*LDACCS
      NPACCS=3*NPACCS
414   NTACCS=LDACCS+NPACCS
      LDAT=LDAT+LDACCS
      NPAT=NPAT+NPACCS
      NTAT=NTAT+NTACCS
      MIT=JM(14,KKK)
      RRUN=RTIME/MIT
      DLDS=1*IRES+(1-IRES)*RRUN
      DMD=(RPAR(8)*DLDS*PARM(14)*MMS+RRUN*JPACCS)*(1+2*ITMR)
      TDMD=TDMD+DMD
      WRITE(6,1595) KKK,NEXC(KKK),LDACCS,NPACCS,NTACCS,DMD
1595  FORMAT(4X,I4,8X,I4,6X,I8,4X,I8,4X,I8,4X,F8.0)
415   CONTINUE
      WRITE(6,1596) LDAT,NPAT,NTAT,TDMD
```

```
1596  FORMAT(9X,'TOTALS',7X,3(1X,I11),2X,F10.0)
      WRITE(6,1600)
1600  FORMAT(//,10X,'NEW SCHEDULE STATISTICS')
      WRITE(6,1605)
1605  FORMAT(/,6X,'TONS',4X,'NJWMS',7X,'NFBS',4X,'MSPACES',
     C             6X,'NJWME',7X,'NFBE',4X,'MSPACEE')
     C
      DO 416 IX=1,7
416   SMEAN(IX)=0.0
1610  FORMAT(7(1X,I9))
      DO 417 KKK=1,NSSTC
      IF(KKK.EQ.1) GO TO 4161
      SMEAN(1)=SMEAN(1)+NSST(1,KKK)-NSST(1,KKK-1)
4161  DO 4163 KKX=2,7
4163  SMEAN(KKX)=SMEAN(KKX)+NSST(KKX,KKK)
      WRITE(6,1610) (NSST(IX,KKK),IX=1,7)
417   CONTINUE
      DO 4171 KKX=1,7
4171  SMEAN(KKX)=SMEAN(KKX)/NSSTC
      WRITE(6,4175) SMEAN
4175  FORMAT(7(1X,F9.4))
      WRITE(6,1535)
1535  FORMAT(/,10X,'CUMULATIVE UTILIZATIONS AT 20 STEP INTER
     CVALS',/)
      WRITE(6,1540)
1540  FORMAT(6X,'TIME',13X,'CPU',17X,'IOP',17X,'MBW',
     C             16X,'MSPACE',11X,'IN QUEUE')
     C
      WRITE(6,1542)
1542  FORMAT(16X,'LONG',5X,'SHORT',6X,'LONG',5X,'SHORT',
     C             6X,'LONG',5X,'SHORT',6X,'LONG',5X,'SHORT',
     C6X,'LONG',                 5X,'SHORT')
     C
      INDX=TIME/20
      DO 420 KK=1,INDX
      WRITE(6,1544) IUTL(1,KK),((IUTL(IX,KK),IUTL(IX,KK+100)
     C),IX=2,6)
1544  FORMAT(11(4X,I6))
420   CONTINUE
      GO TO 1
1180  FORMAT(' MAIN NOW CALLS FEC FOR RMVL OF LEAD ENTRIES,1
     C(4I)')
1040  CALL FEC(2,0,0,0)
      IF(IPASS(6)+10-TIME) 501,1030,1030
501   IF(2*IPASS(10).GE.PARM(17).AND.2*NJWM.GE.IPASS(2)) GO
     CTO 22
```

```
      IF(BCUU-RPAR(1)*ACUU) 505,502,502
502   IF(2*IPASS(7)-PARM(23)) 1030,1030,505
505   IF(BPUU-RPAR(1)*APUU) 510,507,507
507   IF(2*IPASS(9)-PARM(19)) 1030,1030,510
510   IF(IPASS(11).GT.0.AND.4*IPASS(10).GE.PARM(17)) GO TO 2
     C2
      IF(NJWM-IPASS(11).GT.0.AND.4*IPASS(10).GE.PARM(17)) GO
     C TO 22
      IF(IPASS(1)-IPASS(2)-NJWM+IPASS(11).GT.0) GO TO 22
      GO TO 1030
98    WRITE(6,210) IASK,IASK,IDSH,IDSH
210   FORMAT(//,' ',A4,'ERROR',A4,' ATTEMPT TO EXECUTE SIMUL
     CATION BEFORE INITIALIZING SYS. PARAMETERS',A4,'EXECUTI
     CON SUPPRESSED',A4,'SIMULATION TERMINATED')
     C
      GO TO 100
97    WRITE(6,211) IASK,IASK,IDSH,IDSH
211   FORMAT(//,' ',A4,'ERROR',A4,' COMMAND CARD ENCOUNTERED
     C',A4,'PROBABLE CAUSE IS MISSING LOCAL OPTION CARDS',A4
     C,'SIMULATION TERMINATED')
     C
      GO TO 100
99    WRITE(6,202) IASK,IASK
202   FORMAT(//,' ',A4,'END OF SIMULATION',A4)
100   WRITE(6,999)
999   FORMAT('1')
      STOP
      END
```

```
      SUBROUTINE SETUP
      COMMON/BLK1/ IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQ
     CL,IDST
      COMMON/BLK1/ INAME,IDATA,MJDT,IBST,IEXIT,IFLD
      COMMON/BLK2/ IWRD(100),FLD(50)
      COMMON/BLK5/ SET
      COMMON/BLK12/PARM(35)
      COMMON/BLK13/RPAR(11)
      COMMON/BLK14/IFLAG(47),ITAG(46)
      COMMON/BLK15/KNAME(46)
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),      IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),     MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),        NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,            NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER PARM
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,          ISAVE,IVALCV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,     NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 HFIX,IFLAG,I,J,ITAG,SET
      COMMON/BLK17/ ISTCNT,ISTAT(6,400),IUTL(6,200)
      INTEGER*2 ISTAT,IUTL
      DATA ISTR,IEND,IERR/'*   ','END ','*ERR'/
      COMMON/BLK18/RUTL(15)
      WRITE(6,201) LIND,LIND
201   FORMAT(' ',33A4)
      WRITE(6,202) IASK,IASK,IASK,IASK
202   FORMAT(//,' ',A4,' BEGINNING SYSTEM INITIALIZATION ',A
     C4,//,' ',A4,' INPUT PARAMETERS ',A4)
     C
      DO 1 I=1,46
1     ITAG(I)=0
10    CALL ICARD
      IF((IWRD(1).EQ.ISTR).AND.(IWRD(2).EQ.IEND)) GO TO 12
      IF(IWRD(1).EQ.IDLR) GO TO 16
      DO 11 I=1,46
      IF(IWRD(1).NE.KNAME(I)) GO TO 11
      IF((IWRD(2).NE.IEQL).OR.(IWRD(3).NE.IFLD).OR.(IWRD(4).
     CNE.1)) GO TO 17
     C
      IFLAG(I)=1
      ITAG(I)=1
```

```
      IF(I.LT.36) PARM(I)=FLD(1)
      IF(I.GE.36) RPAR(I-35)=FLD(1)
11    CONTINUE
      GO TO 10
12    DO 2 I=1,24
      IBWCTR(I)=0
      MAS(I)=PARM(17)/PARM(20)
      MODNM(I)=0
      MTP1(I)=0
      NA(I)=0
      NAA(I)=PARM(2)/PARM(15)
      NTP(I)=0
      NTR(I)=0
      NUCA(I)=0
      DO 2 J=1,40
2     NREQ(J,I)=0
      DO 3 I=1,10
      IPASS(I)=0
      IPASS(I+10) =0
      ICPU(I)=0
      IVALOV(I)=0
3     CONTINUE
      DO 300 I=1,3
      DO 300 J=1,3
300   ISAVE(I,J)=0
      DO 310 I=1,16
310   IPROS(I)=0
      DO 4 I=1,40
      DO 5 J=1,5
5     ICEC(J,I)=0
      DO 6 J=1,4
6     IFEC(I,J)=0
4     NAML(I)=0
      DO 8 I=1,128
      NBLK(I)=0
      DO 8 J=1,5
      MAVL(I,J)=0
      MALC(I,J)=0
8     MALC((I+128),J)=0
      DO 9 I=1,3
      NFB(I)=0
9     CONTINUE
      DO 99 I=1,15
99    RUTL(I)=0.0
      IPASS(7)=PARM(23)
      IPASS(9)=PARM(19)
```

```
      IPASS(10)=PARM(17)
      ISTCNT=0
      ICECS=-1
      IFECS=-1
      MALCS=-1
      MAVLS=1
      MAVL(1,1)=-1
      MAVL(1,2)=-1
      MAVL(1,3)=0
      MAVL(1,4)=-1
      MAVL(1,5)=PARM(17)
      NFB(1)=1
      NAB=0
      NAG=0
      NDLN=0
      NJWM=0
      NSCHED=0
      IF(IFLAG(47).NE.-1) GO TO 14
      J=0
      DO 13 I=1,46
13    IF(IFLAG(I).EQ.0) J=1
      IF(J.EQ.1) GO TO 18
14    SET=2
      IFLAG(47)=1
      RETURN
16    WRITE(6,203) IASK,IASK,IDSH
203   FORMAT(//,' ',A4,'ERROR',A4,' COMMAND CARD ENCOUNTERED
     C DURING SYSTEM INITIALIZATION',A4,'SIMULATION TERMINAT
     CED')
      SET=4
      RETURN
17    WRITE(6,204) IASK,IASK,IDSH
204   FORMAT(//,' ',A4,'ERROR',A4,' INVALID SYSTEM SPEC. CAR
     CD OR ILLEGAL SPEC. VALUE',A4,'SIMULATION TERMINATED')
     C
      SET=4
      RETURN
18    WRITE(6,205) IASK,IASK
205   FORMAT(//,' ',A4,'ERROR',A4,' THE FOLLOWING PARAMETERS
     C ARE UNDEFINED')
     C
      DO 19 I=1,46
19    IF(IFLAG(I).EQ.0) WRITE(6,206) IDSH,KNAME(I),IDSH
206   FORMAT(/,' ',3A4)
      SET=4
      RETURN
```

END

```
      SUBROUTINE ICARD
      COMMON/BLK2/IWRD(100),FLD(50)
      INTEGER*2 K(80),ISP(6)/'+','-','.',' ','*',',','/,MAXA/'
     CZ'/,MINN/'0'/,MAXN/'9'/,MINS/' '/,I,IW,NF,IL,IT,K1,NDG
     CT,ISET
      DATA IFLD,IERR,ICMT1,ICMT2/'*FLD','*ERR','C   ','*   '
     C/
1     DO 2 I=1,100
2     IWRD(I)=0
      DO 3 I=1,50
3     FLD(I)=0.0
      IW=1
      NF=1
4     IC=0
      READ(5,201) K
201   FORMAT(80A1)
      WRITE(6,202) K
202   FORMAT(' ',80A1)
5     IC=IC+1
      IF((IWRD(1).EQ.ICMT1).AND.(IWRD(2).EQ.ICMT2)) GO TO 1
      IF(IC.GT.80) RETURN
6     IF(K(IC).EQ.ISP(4)) GO TO 5
      IF(K(IC).LE.MAXA) GO TO 8
      IF((K(IC).GE.MINN).AND.(K(IC).LE.MAXN)) GO TO 13
      DO 7 I=1,3
      IF(K(IC).EQ.ISP(I)) GO TO 13
7     CONTINUE
      IF((IC.EQ.80).AND.(K(IC).EQ.ISP(5))) GO TO 4
      IWRD(IW)=K(IC)*(256**2)+ISP(4)
      IW=IW+1
      GO TO 5
8     IL=1
      IT=8
9     IF(IC.GT.80) RETURN
      IWRD(IW)=IWRD(IW)+(K(IC)/256)*(256**(4-IL))
      IF(IC.EQ.80) GO TO 10
      IF(K(IC+1).GE.MINS) IT=5
      IF(IL.EQ.4) GO TO 12
      IF(IT.EQ.5) GO TO 10
      IC=IC+1
      IL=IL+1
      GO TO 9
10    IF((4-IL).EQ.0) GO TO 12
      K1=4-IL
      DO 11 I=1,K1
```

```
11    IWRD(IW)=IWRD(IW)+(ISP(4)/256)*(256**(4-IL-I))
      IWRD(IW)=IWRD(IW)-(256**(4-IL))
      IW=IW+1
      GO TO 5
12    IWRD(IW)=IWRD(IW)-1
      IW=IW+1
      IF(IT.EQ.5) GO TO 5
      IC=IC+1
      GO TO 8
13    IF(IW.LT.3) GO TO 14
      IF(IWRD(IW-2).EQ.IFLD) IW=IW-2
14    IWRD(IW)=IFLD
      IW=IW+1
      NDGT=0
      IDEC=0
      SIGN=1.0
      FCTR=1.0
      IF(K(IC).EQ.ISP(2)) SIGN=-1.0
      IF((K(IC).EQ.ISP(1)).OR.(K(IC).EQ.ISP(2))) IC=IC+1
      IF(IC.GT.80) GO TO 19
15    IF((K(IC).GE.MINN).AND.(K(IC).LE.MAXN)) GO TO 16
      IF(K(IC).EQ.ISP(3)) GO TO 17
      IF(NDGT.EQ.0) GO TO 19
      IF((K(IC).EQ.ISP(4)).OR.(K(IC).EQ.ISP(6))) GO TO 18
      IF((K(IC).EQ.ISP(5)).AND.(IC.EQ.80)) GO TO 4
      GO TO 19
16    FLD(NF)=FLD(NF)*10.0+FLOAT((K(IC)+4032)/256)
      IC=IC+1
      IF(IC.GT.80) GO TO 17
      NDGT=NDGT+1
      GO TO 15
17    IF(IDEC.GT.0) GO TO 19
      IC=IC+1
      IDEC=IC
      IF(IC.GT.80) GO TO 18
      GO TO 15
18    IF(IDEC.GT.0) FCTR=10.0**FLOAT(IC-IDEC)
      FLD(NF)=SIGN*FLD(NF)/FCTR
      NF=NF+1
      IWRD(IW)=IWRD(IW)+1
      IW=IW+1
      GO TO 5
19    IWRD(IW)=IERR
20    IF(K(80).NE.ISP(5)) RETURN
      READ(5,201) K
      GO TO 20
```

```
      SUBROUTINE BULK
      COMMON/BLK1/IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQL
     C,IDST
      COMMON/BLK1/INAME,IDATA,MJDT,IBST,IEXIT,IFLD
      COMMON/BLK2/IWRD(100),FLD(50)
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      COMMON/BLK4/NAME(75,8),DATA(75,9),NJOB
      COMMON/BLK5/ISET
      DIMENSION IAPX(9),FMAX(9),FMIN(9)
      INTEGER*2 NDIST,NJOB,ISET,IJOB,I,J,N,M
      REAL NA/' NA '/
      DATA ISTR,ICMA,IEND,IBLC/'*   ',',   ','END ','BLOC'/
      DATA IFX,IFN,ISEED,IFCN/'FMAX','FMIN','SEED','FCTN'/
      DATA IERR/'*ERR'/
      ISC=1
      NJOB=C
      IJOB=1
      DO 1 I=1,75
      DO 2 J=1,8
      NAME(I,J)=3
2     CONTINUE
1     CONTINUE
3     WRITE(6,201) LINA
201   FORMAT(' ',33A4)
      WRITE(6,202) IASK,IASK
202   FORMAT(/,' ',41X,A4,' JOB GENERATOR INPUT PARAMETERS '
     C,A4)
4     NJOB=NJOB+1
      CALL ICARD
      IF(IWRD(1).GT.0) GO TO 11
      IF((IWRD(1).NE.INAME).OR.(IWRD(2).NE.IEQL)) GO TO 9
      IF(NJOB.GT.75) GO TO 10
      N=3
      M=1
      IF(IWRD(N).NE.ISTR) GO TO 6
      NAME(NJOB,M)=IWRD(N)
5     N=N+1
6     M=M+1
      IF(IWRD(N).GE.0) GO TO 8
      IF(M.GT.8) GO TO 37
      NAME(NJOB,M)=IWRD(N)
      N=N+1
      IF(IWRD(N).EQ.0) GO TO 7
      IF(IWRD(N).NE.ICMA) GO TO 8
      GO TO 5
```

```
7       IF((IWRD(1).EQ.ISTR).AND.(N.EQ.4)) GO TO 9
        IF(N.EQ.3) GO TO 9
        GO TO 4
8       WRITE(6,203) IASK,IASK,IDSH
203     FORMAT(//,' ',A4,'ERROR',A4,' IMPROPER NAME,MISSING CO
       CMMA,OR INVALID DATA ENTRY',A4,'SIMULATION TERMINATED')
       C
        ISET=4
        RETURN
9       WRITE(6,204) IASK,IASK,IDSH
204     FORMAT(//,' ',A4,'ERROR',A4,' PARAMETER DEFINITION CAR
       CD IS INVALID,MISSING,OR OUT OF ORDER',A4,'SIMULATION T
       CERMINATED')
        ISET=4
        RETURN
10      WRITE(6,205) IASK,IASK,IDSH
205     FORMAT(//,' ',A4,'WARNING',A4,' ATTEMPT TO GENERATE MO
       CRE THAN 75 JOBS',A4,'EXCESS ENTRIES IGNORED')
       C
        GO TO 4
11      IF(IWRD(1).NE.IDLR) GO TO 100
        IF((IWRD(2).EQ.IEXIT).AND.(IWRD(3).EQ.IBST)) GO TO 34
        IF((IWRD(2).EQ.IEND).AND.(IWRD(3).EQ.IBLC)) GO TO 12
130     WRITE(6,206) IASK,IASK,IDSH
206     FORMAT(//,' ',A4,'ERROR',A4,' INVALID EXIT, INVALID EN
       CD BLOC, OR ATTEMPT TO EXECUTE NEW COMMAND BEFORE EXIT
       CFROM BST ROUTINE',A4)
        WRITE(6,207) IDSH
207     FORMAT(' ',A4,'SIMULATION TERMINATED')
        ISET=4
        RETURN
12      CALL ICARD
        NJOB=NJOB-1
        IF((IWRD(1).NE.ISEED).OR.(IWRD(2).NE.IEQL)) GO TO 13
        IF((IWRD(3).NE.IFLD).OR.(IWRD(4).EQ.IERR)) GO TO 8
        IF(IWRD(4).GT.1) WRITE(6,208) IASK,IASK,IDSH
208     FORMAT(//,' ',A4,'WARNING',A4,' MORE THAN 1 VALUE HAS
       CBEEN SPECIFIED FOR SEED VARIABLE',A4,'EXCESS ENTRIES I
       CGNORED')
        ISD=IFIX(FLD(1))
        CALL ICARD
13      IF((IWRD(1).NE.IFCN).OR.(IWRD(2).NE.IEQL)) GO TO 9
        N=3
        M=1
14      IF(M.GT.9) GO TO 15
        IF(IWRD(N).GE.0) GO TO 8
```

```
      IAPX(M)=IWRD(N)
      N=N+1
      IF(IWRD(N).EQ.0) GO TO 16
      IF(IWRD(N).NE.ICMA) GO TO 8
      N=N+1
      M=M+1
      GO TO 14
15    WRITE(6,209) IASK,IASK,IDSH
209   FORMAT(//,' ',A4,'WARNING',A4,' MORE THAN 9 DIST. APPR
     COXIMATION CURVES SPECIFIED',A4,'EXCESS ENTRIES IGNORED
     C')
16    IF(M.LT.9) GO TO 21
      CALL ICARD
      IF((IWRD(1).NE.IFX).OR.(IWRD(2).NE.IEQL)) GO TO 9
      IF((IWRD(3).NE.IFLD).OR.(IWRD(4).EQ.IERR)) GO TO 8
      IF(IWRD(4).LT.9) GO TO 22
      IF(IWRD(4).EQ.9) GO TO 17
      WRITE(6,210) IASK,IASK,IDSH
210   FORMAT(//,' ',A4,'WARNING',A4,' MORE THAN 9 MAX. OR MI
     CN. PARAMETER VALUES SPECIFIED',A4,'EXCESS ENTRIES IGNO
     CRED')
17    DO 18 I=1,9
      IF(FLD(I).LT.0) GO TO 9
      FMAX(I)=FLD(I)
18    CONTINUE
      CALL ICARD
      IF((IWRD(1).NE.IFN).OR.(IWRD(2).NE.IEQL)) GO TO 9
      IF((IWRD(3).NE.IFLD).OR.(IWRD(4).EQ.IERR)) GO TO 8
      IF(IWRD(4).LT.9) GO TO 22
      IF(IWRD(4).EQ.9) GO TO 19
      WRITE(6,210) IASK,IASK,IDSH
19    DO 20 I=1,9
      IF(FLD(I).LT.0) GO TO 9
      FMIN(I)=FLD(I)
20    CONTINUE
      GO TO 23
21    WRITE(6,211) IASK,IASK,IDSH
211   FORMAT(//,' ',A4,'ERROR',A4,' FEWER THAN 9 DIST. APPRO
     CXIMATIONS SPECIFIED',A4,'UNABLE TO GENERATE JOB PARAME
     CTERS')
      ISET=4
      RETURN
22    WRITE(6,212) IASK,IASK,IDSH
212   FORMAT(/,' ',A4,'ERROR',A4,' FEWER THAN 9 MAX. OR MIN.
     C PARAMETER  VALUES SPECIFIED',A4,'UNABLE TO GENERATE J
     COB PARAMETERS')
```

```
      ISET=4
      RETURN
23    DO 24 I=1,9
      DO 25 J=1,NDIST
      IF(IDIST(J).EQ.IAPX(I)) GO TO 26
25    CONTINUE
      WRITE(6,213) IASK,IASK,IAPX(I),IDSH
213   FORMAT(//,' ',A4,'ERROR',A4,' UNABLE TO LOCATE ',A4,'
     CON STAT. DISTRIBUTION TABLE',A4,'UNABLE TO GENERATE JO
     CB PARAMETERS')
      ISET=4
      RETURN
26    SMALL=F(J,2)
      BIG=SMALL
      DO 27 K=3,21
      IF(F(J,K).LT.SMALL) SMALL=F(J,K)
      IF(F(J,K).GT.BIG) BIG=F(J,K)
27    CONTINUE
      SCALE=FMAX(I)
      IF(BIG.NE.SMALL) SCALE=(FMAX(I)-FMIN(I))/(BIG-SMALL)
      DO 29 K=IJOB,NJOB
      IY=ISD*65539
      IF(IY.GT.0) GO TO 28
      IY=IY+2147483647+1
28    YFL=FLOAT(IY)*.4656613E-9
      ISD=IY
      INDX=IFIX(19.999*YFL)+2
      DATA(K,I)=FMIN(I)+SCALE*(F(J,INDX)-SMALL)
      IF((I.GE.1).AND.(I.LE.5)) DATA(K,I)=FLOAT(IFIX(DATA(K,
     CI)))
31      IF((NAME(K,1).NE.ISTR).AND.(I.EQ.7)) DATA(K,I)=NA
      IF((NAME(K,1).EQ.ISTR).AND.(I.EQ.8)) DATA(K,I)=NA
29    CONTINUE
24    CONTINUE
214   FORMAT(//,' ',A4,' COMMON STATISTICS JOB BLOCK ',A4)
215   FORMAT(/,' ',A4,'JOB ',2A4,3X,'EP=',F7.0,10X,'TMR=',F7
     C.0,/,20X     ,'CR=',F7.0,10X,'I-O=',F7.0,/,20X,'NI=',F7
     C.0,5X,'I-O DIST=',2X,A4,/,19X,'MIX=',3X,F4.2,11X,'RR='
     C,F7.1,/,19X,'IAT=',3X,A4,10X,'DVT=',F7.1)
     C
216   FORMAT(' PREDECESSORS',5(2X,A4))
217   FORMAT(/,' ',A4,'JOB ',2A4,3X,'EP=',F7.0,10X,'TMR=',F7
     C.0,/,20X     ,'CR=',F7.0,10X,'I-O=',F7.0,/,20X,'NI=',F7
     C.0,5X,'I-O DIST=',2X,A4,/,19X,'MIX=',3X,F4.2,11X,'RR='
     C,3X,A4,/,19X,'IAT=',F7.1,10X,'DVT=',F7.1)
     C
```

```
      WRITE(6,218) IDSH,IAPX
218   FORMAT(/,' DISTRIBUTION APPROXIMATIONS USED',A4,2X,'EP
     C-- ',A4,9X, 'TMR--',A4,/,39X,'CR-- ',A4,9X,'I-O-- ',A4
     C,/,39X,'NI-- ',A4,9X,'MIX-- ',A4,/,39X,'RR-- ',A4,9X,'
     CIAT-- ',A4,/,38X,'DVT-- ',A4)
      WRITE(6,219) IDSH,IDSH
219   FORMAT(/,' ',A4,'END OF JOB BLOCK',A4,/)
      WRITE(6,201) LIND
      IJOB=NJOB+1
      GO TO 4
34    WRITE(6,999)
999   FORMAT('1')
      NJOB=NJOB-1
      WRITE(6,201) LINA,LINA
      WRITE(6,220) IASK,IASK
22    FORMAT(//,48X,A4,' COMPLETE BULK STORAGE TABLE ',A4,//
     C)
      WRITE(6,201) LIND
      DO 36 I=1,NJOB
      IF(NAME(I,1).NE.ISTR) GO TO 35
      WRITE(6,215) IDSH,NAME(I,2),IDSH,(DATA(I,J),J=1,5),NAM
     CE(I,3),(DATA(I,J),J=6,9)
     C
      WRITE(6,216)(NAME(I,J),J=4,8)
      GO TO 36
35    WRITE(6,217) IDSH,NAME(I,2),IDSH,(DATA(I,J),J=1,5),NAM
     CE(I,3),(DATA(I,J),J=6,9)
     C
      WRITE(6,216)(NAME(I,J),J=4,8)
36    CONTINUE
      WRITE(6,221) IDSH,IDSH
221   FORMAT(//,' ',A4,'END OF BST',A4,//)
      WRITE(6,201) LIND,LINA,LINA
      ISET=2
      WRITE(6,999)
      RETURN
37    WRITE(6,222) IASK,IASK,NAME(NJOB,2),IDSH
222   FORMAT(//,' ',A4,'WARNING',A4,' ATTEMPT TO ENTER MORE
     CTHAN 5 PREDECESSORS IN PRED. FIELD OF JOB ',2A4,'EXCES
     CS ENTRIES IGNORED')
      GO TO 4
      END
```

```
      SUBROUTINE STAT
      COMMON/BLK1/IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQL
     C,IOST
      COMMON/BLK1/INAME,IDATA,MJDT,IBST,IEXIT,IFLD
      COMMON/BLK2/IWRD(100),FLD(50)
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      COMMON/BLK5/ISET
      INTEGER*2 NDIST,I,J,K,ISET,L
      DATA IERR/'*ERR'/
      DO 1 I=1,25
      IDIST(I)=0
      DO 1 J=1,41
      F(I,J)=0.0
1     CONTINUE
      NDIST=1
2     IF(NDIST.GT.25) GO TO 12
13    CALL ICARD
      IF(IWRD(1).GT.0) GO TO 10
      IF((IWRD(1).NE.INAME).OR.(IWRD(2).NE.IEQL)) GO TO 8
      IF(IWRD(4).NE.0) GO TO 8
      IF(IWRD(3).GT.0) GO TO 9
      IDIST(NDIST)=IWRD(3)
      CALL ICARD
      IF((IWRD(1).NE.IDATA).OR.(IWRD(2).NE.IEQL)) GO TO 8
      IF((IWRD(3).NE.IFLD).OR.(IWRD(4).EQ.IERR)) GO TO 9
      K=IWRD(4)
      IF(K-41) 3,5,7
3     WRITE(6,201) IASK,IASK,IDIST(NDIST),IDSH
201   FORMAT(//,' ',A4,'WARNING',A4,' FEWER THAN 41 DATA POI
     CNTS HAVE BEEN SPECIFIED FOR ',2A4,'DEFAULT VALUE=0.0')
     C
5     DO 6 I=1,K
      F(NDIST,I)=FLD(I)
6     CONTINUE
      NDIST=NDIST+1
      GO TO 2
7     WRITE(6,202) IASK,IASK,IDIST(NDIST),IDSH
202   FORMAT(//,' ',A4,'WARNING',A4,' MORE THAN 41 DATA POIN
     CTS HAVE BEEN SPECIFIED FOR ',2A4,'EXCESS POINTS IGNORE
     CD')
      K=41
      GO TO 5
8     WRITE(6,203) IASK,IASK,IDSH
203   FORMAT(//,' ',A4,'ERROR',A4,' IMPROPERLY FORMATED OR M
     CISSING NAME OR DATA DEFINITION CARD',A4,'SIMULATION TE
```

```
     CRMINATED')
      ISET=4
      RETURN
9     WRITE(6,204) IASK,IASK,IDSH
204   FORMAT(//,' ',A4,'ERROR',A4,' IMPROPER DISTRIBUTION NA
     CME OR ERROR IN DATA FIELD',A4,'SIMULATION TERMINATED')
     C
      ISET=4
      RETURN
10    IF(IWRD(1).NE.IDLR) GO TO 11
      IF((IWRD(2).NE.IEXIT).OR.(IWRD(3).NE.IDST)) GO TO 11
      NDIST=NDIST-1
      DO 102 I=1,NDIST
      DO 102 J=1,NDIST
      IF((IDIST(I).NE.IDIST(J)).OR.(I.EQ.J)) GO TO 102
      WRITE(6,207) IASK,IASK,IDIST(J),IDSH
207   FORMAT(//,' ',A4,'WARNING',A4,' DIST. NAMED ',A4,' HAS
     C BEEN ASSIGNED MORE THAN ONE SET OF DATA POINTS',A4,'F
     CIRST SET USED')
      K=J+1
      DO 101 L=K,NDIST
      IDIST(L-1)=IDIST(L)
      DO 101 M=1,41
      F((L-1),M)=F(L,M)
101   CONTINUE
      NDIST=NDIST-1
102   CONTINUE
      ISET=2
      RETURN
11    WRITE(6,205) IASK,IASK,IDSH
205   FORMAT(//,' ',A4,'ERROR',A4,' INVALID EXIT OR ATTEMPT
     CTO EXECUTE NEW COMMAND BEFORE EXIT FROM DIST ROUTINE',
     CA4,'SIMULATION TERMINATED')
     C
      ISET=4
      RETURN
12    WRITE(6,206) IASK,IASK,IDSH
206   FORMAT(//,' ',A4,'WARNING',A4,' ATTEMPT TO ENTER MORE
     CTHAN 25 DISTRIBUTIONS ON STAT. TABLE',A4,'EXCESS ENTRI
     CES IGNORED')
      GO TO 13
      END
```

```
      SUBROUTINE PRNT
      COMMON/BLK1/ IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQ
     CL,IDST
      COMMON/BLK1/ INAME,IDATA,MJDT,IBST,IEXIT,IFLD
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      COMMON/BLK5/ SET
      COMMON/BLK6/ MNAME(8,64)
      COMMON/BLK7/ JM2(27,64)
      COMMON/BLK11/ RTIME
      INTEGER*2  SET,JM2,MIX,IP,IOSH,EP,MREQ,NDIST,KY,ISUM,I
     C,J,K,L,M
      DIMENSION NBITS(4),IVALOV(10),KY(14)
      INTEGER*2 IVALOV
      DATA ISTR/'*   '/
      WRITE(6,999)
999   FORMAT('1')
      WRITE(6,201) LINA
      WRITE(6,216) IASK,RTIME,IASK
216   FORMAT(/,' ',A4,' TIME =',F8.3,A4,/)
      WRITE(6,201) LINA,LINA
201   FORMAT(' ',33A4)
      WRITE(6,202) IASK,IASK
202   FORMAT(/,48X,A4,' MAIN JOB DESCRIPTION TABLE ',A4)
      DO 1 I=1,64
      IF(MNAME(2,I).NE.0) GO TO 2
1     CONTINUE
      WRITE(6,203) IDSH,IDSH
203   FORMAT(/,' ',A4,' NO JOBS ON MJDT',A4,/)
      GO TO 15
2     WRITE(6,1000)
1000  FORMAT(/,10X,'STATUS WORD BITS RIGHT TO LEFT'/,
     C             10X,' 1  READY',11X,' 2  INITIATED',7X,' 3
     C  WAITING CPU'/,         10X,' 4  WAITING IOP',5X,' 5
     CWAIT MEMORY',5X,' 6  IN HOLD'/,      10X,' 7  PREEMPTED
     C',7X,' 8  MODE,CPU/IOP',4X,' 9  PERIODIC'/,      10X,'1
     C  RESIDENT',8X,'11  SIMPLEX/TMR',5X,'12  WAIT I/O COM
     CP.'/, 10X,'13  IN IFEC',9X,'14  LOAD',12X,'15  NOT USE
     CD'/,                10X,'16  NOT USED')
     C
      DO 14 I=1,64
      IF(MNAME(2,I).EQ.0) GO TO 14
      WRITE(6,204) IDSH,MNAME(2,I),IDSH,I
204   FORMAT(/,' ',A4,' JOB ',2A4,3X,'LOCATION = ',I2)
      ISUM=0
      DO 4 J=1,4
```

```
      IF(JM2(J+1,I)) 4,4,400
400   ISUM=ISUM+1
      KY(ISUM)=JM2(J+1,I)
4     CONTINUE
      N=1
405   J22=JM2(22,I)
      NL=N-1
      CALL UNMIX(N,NL,J22,IND)
      IF(IND) 430,430,410
410   DO 415 J=1,10
415   IVALOV(J)=0
      CALL OVFLMG(1,I,N,IVALOV,10)
      DO 425 J=1,10
      IF(IVALOV(J)) 425,425,420
420   ISUM=ISUM+1
      KY(ISUM)=IVALOV(J)
425   CONTINUE
430   GO TO (5,8),N
5     IF(ISUM.EQ.0) GO TO 6
      WRITE(6,205)
205   FORMAT(/,20X,'PREDECESSOR/LOCATION')
      WRITE(6,212)((MNAME(2,KY(J)),KY(J)),J=1,ISUM)
212   FORMAT(20X,4(A4,'/',I2,2X))
      GO TO 601
6     WRITE(6,213) IDSH,IDSH
213   FORMAT(20X,A4,'NO PREDECESSORS',A4)
601   ISUM=0
      DO 7 J=1,2
      N=JM2(J+5,I)
      CALL UNMIX(7,0,N,IND1)
      CALL UNMIX(14,7,N,IND2)
      IF(IND1) 701,701,700
700   ISUM=ISUM+1
      KY(ISUM)=IND1
701   IF(IND2) 7,7,702
702   ISUM=ISUM+1
      KY(ISUM)=IND2
7     CONTINUE
      N=2
      GO TO 405
8     IF(ISUM.EQ.0) GO TO 10
      WRITE(6,206)
206   FORMAT(/,22X,'SUCCESSORS/LOCATION')
      WRITE(6,212) ((MNAME(2,KY(J)),KY(J)),J=1,ISUM)
      GO TO 11
10    WRITE(6,214) IDSH,IDSH
```

```
214   FORMAT(20X,A4,' NO SUCCESSORS',A4)
11    N=JM2(27,I)
      CALL BITWRT(N,4,NBITS)
      WRITE(6,208) NBITS
208   FORMAT(/,10X,'CURRENT STATUS WORD = ',4(1X,A4))
      IP=JM2(10,I)/128
      MIX=JM2(10,I)-IP*128
      IOSH=JM2(11,I)/1024
      EP=JM2(12,I)/1024
      MREQ=JM2(17,I)-(JM2(17,I)/2048)*2048
      MNARC=JM2(11,I)-(JM2(11,I)/1024)*1024
      ISTDVC=JM2(12,I)-(JM2(12,I)/1024)*1024
      MNARI=JM2(13,I)-(JM2(13,I)/1024)*1024
      ISTDVI=JM2(21,I)-(JM2(21,I)/1024)*1024
      MBLK1=JM2(1,I)/128
      MBLK2=JM2(19,I)-(JM2(19,I)/256)*256
      MBLK3=(JM2(17,I)/2048)*16
     C             +(JM2(19,I)-(JM2(19,I)/4096)*4096)/256
     C
      WRITE(6,209) EP,MREQ,JM2(15,I),JM2(9,I),MIX,JM2(14,I),
     CJM2(8,I),JM2(25,I),JM2(20,I),JM2(26,I),IP,JM2(18,I),MN
     CARC,ISTDVC,              MNARI,ISTDVI,MBLK1,MBLK2,MBLK3
     C,JM2(22,I)
209   FORMAT(/,20X,'EP =',I6,10X,'MMS =',I6,5X,'VSF(I-O) =',
     CI6,10X,'NMA  =',I6,/,19X,'MIX =',I6,7X,'RR/IAT =',I6,1
     CCX,'TLE =', I6,11X,'TT =',I6,/,19X,'MPT =',I6,5X,'NIP(
     CI-O) =',I6,11X,'IP =',I6,10X,'CCD =',I6,/,17X,'MNARC =
     C',I6,7X,'ISTDVC =',I6,8X,'MNARI =',I6,7X,       'ISTDV
     CI =',I6/17X,'MBLK1 =',I6,8X,'MBLK2 =',I6,8X,
     C      'MBLK3 =',I6,9X,'IFOV =',I6)
     C
      IF(IOSH.EQ.0) GO TO 13
      WRITE(6,210) IDIST(IOSH),IOSH
210   FORMAT(/,24X,'I-O DIST/LOCATION',5X,A4,'/',I2)
      WRITE(6,201) LIND
      GO TO 14
13    WRITE(6,215) IDSH,IDSH
215   FORMAT(/,24X,A4,'NO I-O SHAPE INDICATED FOR THIS JOB',
     CA4)
      IF(JM2(24,I).NE.0) WRITE(6,211) IDSH,IDSH
211   FORMAT(' ',A4,'THIS JOB CONTAINS OVERFLOW FIELDS',A4,/
     C)
      WRITE(6,201) LIND,LIND
14    CONTINUE
15    WRITE(6,201) LINA,LINA
      SET=2
```

```
      RETURN
      END
```

```
      SUBROUTINE CLEAR
      COMMON/BLK1/ IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQ
     CL,IDST
      COMMON/BLK1/ INAME,IDATA,MJDT,IBST,IEXIT,IFLD
      COMMON/BLK4/NAME(75,8),DATA(75,9),NJOB,NJOBER
      COMMON/BLK5/ SET
      COMMON/BLK6/ MNAME(8,64)
      COMMON/BLK7/ JM2(27,64)
      COMMON/BLK9/ JOVF(1024)
      COMMON/BLK10/ FLAG(64)
      INTEGER*2 SET,JM2,JOVF,I,J,FLAG,NJOB,NJOBER
      WRITE(6,201) LIND
201   FORMAT(' ',33A4)
      DO 1 I=1,64
      FLAG(I)=0
      DO 2 J=1,8
2     MNAME(J,I)=0
      DO 3 J=1,27
3     JM2(J,I)=0
1     CONTINUE
      DO 4 I=1,1024
4     JOVF(I)=0
      NJOBER=0
      WRITE(6,202) IASK,IASK
202   FORMAT(//,' ',A4,' MJDT HAS BEEN ERASED ',A4,//)
      WRITE(6,201) LIND
      SET=2
      RETURN
      END
```

```
      SUBROUTINE ENTER
      COMMON/BLK1/ IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQ
     CL,IDST
      COMMON/BLK1/ INAME,IDATA,MJDT,IBST,IEXIT,IFLD
      COMMON/BLK2/ IWRD(100),FLD(50)
      COMMON/BLK4/NAME(75,8),DATA(75,9),NJOB,NJOBER
      COMMON/BLK5/ SET
      COMMON/BLK6/ MNAME(8,64)
      INTEGER*2 NJOB,NJOBER,SET,I,J,K,N
      DATA ICMA,ISTR,IEND/',   ','*   ','END '/
      WRITE(6,201) LIND
201   FORMAT(' ',33A4)
      WRITE(6,202) IDSH,IDSH
202   FORMAT(//,' ',A4,'BEGINNING JOB ENTRY PROCESSING',A4)
1     CALL ICARD
      IF(IWRD(1).GT.0) GO TO 10
      IF((IWRD(1).NE.INAME).OR.(IWRD(2).NE.IEQL)) GO TO 11
      N=3
2     IF(IWRD(N).GT.0) GO TO 12
      IF(IWRD(N).EQ.0) GO TO 11
      DO 3 I=1,NJOB
      IF(NAME(I,2).EQ.IWRD(N)) GO TO 4
3     CONTINUE
      WRITE(6,203) IASK,IASK,IWRD(N),IDSH
203   FORMAT(//,' ',A4,'ERROR',A4,' UNABLE TO LOCATE ',A4,'
     CON BST',A4,'SIMULATION TERMINATED')
     C
      SET=4
      RETURN
4     DO 5 J=1,64
      IF(MNAME(2,J).EQ.IWRD(N)) GO TO 13
5     CONTINUE
      DO 6 J=1,64
      IF(MNAME(2,J).EQ.0) GO TO 7
6     CONTINUE
      WRITE(6,204) IASK,IASK,IDSH,IWRD(N)
204   FORMAT(//,' ',A4,'WARNING',A4,' MJDT IS FULL',A4,'UNAB
     CLE TO ENTER ',A4)
     C
      GO TO 9
7     DO 8 K=1,8
8     MNAME(K,J)=NAME(I,K)
      NJOBER=NJOBER+1
      WRITE(6,205) IDSH,IWRD(N),IDSH
205   FORMAT(' ',2A4,' ENTERED',A4)
```

```
9       N=N+1
        IF(IWRD(N).EQ.0) GO TO 1
        IF(IWRD(N).NE.ICMA) GO TO 12
        N=N+1
        GO TO 2
10      IF((IWRD(1).NE.ISTR).OR.(IWRD(2).NE.IEND)) GO TO 14
        SET=2
        CALL RECHEK
        WRITE(6,206) IDSH,IDSH
206     FORMAT(//,' ',A4,'JOB ENTRY PROCESSING COMPLETE',A4,//
       C)
        WRITE(6,201) LIND
        RETURN
11      WRITE(6,207) IASK,IASK,IDSH
207     FORMAT(//,' ',A4,'ERROR',A4,' NAME CARD (NAME=) IS INV
       CALID OR MISSING',A4,'SIMULATION TERMINATED')
       C
        SET=4
        RETURN
12      WRITE(6,208) IASK,IASK,IDSH
208     FORMAT(//,' ',A4,'ERROR',A4,' INVALID JOB NAME OR MISS
       CING COMMA BETWEEN NAME ENTRIES',A4,'SIMULATION TERMINA
       CTED')
        SET=4
        RETURN
13      WRITE(6,209) IASK,IASK,IWRD(N),IDSH
209     FORMAT(//,' ',A4,'WARNING',A4,' JOB ',A4,' HAS ALREADY
       C BEEN ENTERED ON MJDT',A4,'JOB ENTRY IGNORED')
       C
        GO TO 9
14      WRITE(6,210) IASK,IASK,IDSH
210     FORMAT(//,' ',A4,'ERROR',A4,' END OF JOB ENTRIES CARD
       C(*END) IS MISSING OR INVALID',A4,'SIMULATION TERMINATE
       CD')
        SET=4
        RETURN
        END
```

```
      SUBROUTINE RECHEK
      COMMON/BLK1/ IASK,IDSH,LIND(33),LINA(33),IBLK,IDLR,IEQ
     CL,IDST
      COMMON/BLK1/ INAME,IDATA,MJDT,IBST,IEXIT,IFLD
      COMMON/BLK4/ NAME(75,8),DATA(75,9),NJOB
      COMMON/BLK5/ SET
      COMMON/BLK6/ MNAME(8,64)
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      COMMON/BLK7/ JM2(27,64)
      COMMON/BLK8/ TIME
      COMMON/BLK10/ FLAG(64)
      COMMON/BLK12/PARM(35)
      INTEGER PARM
      DIMENSION LIST(63),IDPL(32,2)
      INTEGER*2 HFIX,SET,JM2,I,N,NT,J,K,L,M,NDPL,NJOB,NDIST,
     CTIME,FLAG
      DATA ISTR,INONE/'*   ','NONE'/
      ISD=PARM(33)
      DO 1 I=1,63
1     LIST(I)=0
      N=1
      NT=0
      NDPL=0
2     DO 13 I=1,64
      DO 13 J=4,8
      IF(MNAME(J,I).EQ.0) GO TO 13
      IF(MNAME(J,I).NE.MNAME(2,I)) GO TO 3
      WRITE(6,201) IASK,IASK,MNAME(J,I),IDSH,IDSH
201   FORMAT(//,' ',A4,'WARNING',A4,' JOB ',A4,' HAS BEEN LI
     CSTED AS ITS OWN PREDECESSOR',A4,'FIXUP MADE',A4,'MJDT
     CLOAD CHECK CONTINUING')
      MNAME(J,I)=0
3     DO 4 K=1,64
      IF(MNAME(J,I).EQ.MNAME(2,K)) GO TO 10
4     CONTINUE
      DO 5 K=1,NJOB
      IF(NAME(K,2).EQ.MNAME(J,I)) GO TO 6
5     CONTINUE
      WRITE(6,202) IASK,IASK,MNAME(J,I),MNAME(2,I),IDSH
202   FORMAT(//,' ',A4,'ERROR',A4,' JOB ',A4,' LISTED AS PRE
     CDECESSOR TO ',A4,' CANNOT BE LOCATED ON BST',A4,'SIMUL
     CATION TERMINATED')
      SET=4
      RETURN
6     DO 7 L=1,64
```

```
      IF(MNAME(2,L).EQ.0) GO TO 8
7     CONTINUE
      WRITE(6,203) IASK,IASK,MNAME(J,I),MNAME(2,I),IDSH
203   FORMAT(//,' ',A4,'ERROR',A4,' MJDT IS FULL. ',A4,' LIS
     CTED AS PREDECESSOR TO ',A4,' CANNOT BE ENTERED',A4,'SI
     CMULATION TERMINATED')
      SET=4
      RETURN
8     DO 9 M=1,8
9     MNAME(M,L)=NAME(K,M)
      N=N+1
      LIST(N)=MNAME(J,I)
      GO TO 13
10    DO 11 M=1,32
      IF((MNAME(2,I).EQ.IDPL(M,1)).AND.(MNAME(2,K).EQ.IDPL(M
     C,2))) GO TO  13
     C
      IF((MNAME(2,I).EQ.IDPL(M,2)).AND.(MNAME(2,K).EQ.IDPL(M
     C,1))) GO TO  13
     C
11    CONTINUE
      DO 12 M=4,8
      IF(MNAME(M,K).NE.MNAME(2,I)) GO TO 12
      NDPL=NDPL+1
      IDPL(NDPL,1)=MNAME(2,K)
      IDPL(NDPL,2)=MNAME(2,I)
12    CONTINUE
13    CONTINUE
      IF(N.EQ.NT) GO TO 14
      NT=N
      GO TO 2
14    IF(N.EQ.0) GO TO 17
      WRITE(6,204) IASK,IASK
204   FORMAT(//,' ',A4,' NON-PRESENT JOBS REFERRENCED ',A4)
      DO 15 I=1,N
15    WRITE(6,205) IDSH,LIST(I),IDSH
205   FORMAT(//,' ',3A4)
      WRITE(6,206) IASK,IASK
206   FORMAT(//,' ',A4,' ALL NON-PRESENT REFERRENCED JOBS EN
     CTERED ',A4)
      IF(NDPL.EQ.0) GO TO 17
      WRITE(6,207) IASK,IASK,IDSH
207   FORMAT(//,' ',A4,'WARNING',A4,' THE FOLLOWING PAIRS OF
     C JOBS FORM DIRECT PREDECESSOR LOOPS',A4,'POSSIBLE FREE
     CZE CONDITION MAY EXIST')
      DO 16 I=1,NDPL
```

```
16      WRITE(6,208) IDSH,IDPL(I,1),IDPL(I,2),IDSH
208     FORMAT(//,' ',2A4,2X,2A4)
17      DO 29 I=1,64
        IF((MNAME(2,I).EQ.0).OR.(FLAG(I).EQ.1)) GO TO 29
        FLAG(I)=1
        DO 18 J=1,NJOB
        IF(NAME(J,2).EQ.MNAME(2,I)) GO TO 19
18      CONTINUE
19      IF(MNAME(1,I).EQ.ISTR) JM2(27,I)=JM2(27,I)+256
        IF(DATA(J,2).GT.0) JM2(27,I)=JM2(27,I)+1024
        JM2(1,I)=I
        IF(MNAME(3,I).EQ.INONE) GO TO 21
        DO 20 IDX=1,NDIST
        IF(IDIST(IDX).EQ.MNAME(3,I)) GO TO 36
20      CONTINUE
21      WRITE(6,209) IASK,IASK,MNAME(3,I),MNAME(2,I),IDSH
209     FORMAT(//,' ',A4,'ERROR',A4,' DISTRIBUTION NAMED ',A4,
     C' LISTED FOR JOB ',A4,' CANNOT BE LOCATED ON DIST. TAB
     CLE',A4,'SIMULATION TERMINATED')
     C
        SET=4
        RETURN
36      JM2(11,I)=IDX*1024
39      K=1
        DO 24 L=4,8
        IF(MNAME(L,I).EQ.0) GO TO 24
        DO 23 M=1,64
        IF(MNAME(L,I).NE.MNAME(2,M)) GO TO 23
        K=K+1
        IF(K.GT.5) GO TO 22
        JM2(K,I)=M
        GO TO 23
22      CALL OVFLMG(2,I,1,M,1)
23      CONTINUE
24      CONTINUE
        JM2(8,I)=TIME
        JM2(9,I)=HFIX(DATA(J,5))
        JM2(10,I)=HFIX(100.0*DATA(J,6)+0.1)
        JM2(12,I)=JM2(12,I)+(HFIX(DATA(J,1)))*1024
        JM2(17,I)=HFIX((DATA(J,3)/1000.0)+0.999)
        IF(MNAME(1,I).EQ.ISTR) JM2(14,I)=HFIX(DATA(J,7))
        IF(MNAME(1,I).NE.ISTR) JM2(14,I)=HFIX(DATA(J,8))
        IF(F(IDX,1).GE.1.) GO TO 25
        IF(DATA(J,4).LE.F(IDX,1)*DATA(J,5)) GO TO 25
        JM2(15,I)=HFIX(F(IDX,1)*DATA(J,5))
        GO TO 29
```

```
25      JM2(15,I)=HFIX(DATA(J,4))
29      CONTINUE
        N=5
        R=5.5
        DO 32 I=1,64
        IF(MNAME(2,I).EQ.0) GO TO 32
        DO 27 L=1,64
        IF(MNAME(2,L).EQ.0) GO TO 27
        DO 26 M=4,8
        IF(MNAME(M,L).NE.MNAME(2,I)) GO TO 26
        DO 31 J=6,7
        K1=JM2(J,I)/128
        IF(K1.EQ.L) GO TO 27
        K1=JM2(J,I)-(K1*128)
        IF(K1.EQ.L) GO TO 27
31      CONTINUE
        J=6
33      K1=JM2(J,I)/128
        IF(K1.NE.0) GO TO 34
        JM2(J,I)=JM2(J,I)+(L*128)
        GO TO 27
34      K1=JM2(J,I)-(K1*128)
        IF(K1.NE.0) GO TO 35
        JM2(J,I)=JM2(J,I)+L
        GO TO 27
35      J=J+1
        IF(J.LE.7) GO TO 33
        CALL OVFLMG(2,I,2,L,1)
        GO TO 27
26      CONTINUE
27      CONTINUE
32      CONTINUE
        RETURN
        END
```

```
BITWRT   START   0          BITWRT(KWORD,LENGTH,NBITS)
R        EQU     10         KWORD CONTAINS INFO. TO BE WRITT
         BC      15,12(15)        EN.  LENGTH IS NO. OF
         DC      X'7'        CHARACTERS TO BE WRITTEN
         DC      CL7'BITWRT '      DIVIDED BY 4(4 CHARACTE
         STM     2,1[illegible],28(13)    RS PER HALF-WORD, 8 CHAR
         BALR    8,0          ACTERS FOR FULL WORD)
         USING   *,8          NBITS IS AN ARRAY THAT HOLDS
         L       9,0(0,1)      THE OUTPU WORD(4 FULL WORDS
         L       2,0(0,9)        ARE REQUIRED IF KWORD
         L       4,=X'FFFFFFFF'        IS A HALF-WORD,
         L       9,4(0,1)           8 FULL WORDS IF KWORD
         L       5,0( ,9)           IS A FULL-WORD
         LR      7,5
         SLL     7,2
         L       9,8(0,1)
         LA      8,4
LOAD     LA      6,4
GO       SRDL    2,1
         SRL     3,7
         BCT     6,GO
         OR      3,4
         SR      7,8
         ST      3,0(7,9)
         BCT     5,LOAD
         LM      2,1[illegible],28(13)
         MVI     12(13),X'FF'
         BCR     15,14
         END
```

```
UNMIX    START 0          UNMIX(INDXU,INDXL,IN,IOUT)
B        EQU   8            INDXU IS UPPER BIT POSITION OF
         BC    15,10(15)       STRING TO BE RECOVERED FROM
         DC    X'5'           INPUT WORD IN
         DC    CL5'UNMIX'       INDXL IS LOWER BIT POSITION
         STM   2,8,28(13)        MINUS 1 OF STRING TO BE
         BALR  B,0          RECOVERED FROM INPUT WORD IN.
         USING *,B           IN IS THE INPUT WORD FROM WHICH
         LM    2,5,0(1)         INFORMATION IS TO BE OBTAINED
         L     6,0(0,4)          IOUT WILL CONTAIN THE
         L     4,0(0,2)         RECOVERED STRING IN POSITIONS
         SRDL  6,0(4)          ONE THRU INDXU-INDXL
         S     4,0(0,3)         ALL PARAMETERS ARE FULL WORD
         LA    2,32
         SR    2,4
         SRL   7,0(2)
         ST    7,0(0,5)
         LM    2,8,28(13)
         MVI   12(13),X'FF'
         BCR   15,14
         END
```

```
PACK     START 0           PACK(NSHIFT,TWRD,SWRD,MASK)
R        EQU   1           NSHIFT INDICATES NO. OF BIT
         BC    15,12(15)        POSITIONS TO RIGHT THAT TWRD
         DC    X'5'          IS TO BE SHIFTED BEFORE NEW
         DC    CL5'PACK'        INFORMATION IS ENTERED FROM
         STM   2,1 ,28(13)         SWRD.
         BALR  8,0             TWRD IS THE TARGET WORD INTO
         USING *,8          WHICH THE INFORMATION FROM
         L     2,8(  ,1)        SWRD IS TO BE PACKED.
         L     4,4(  ,2)         SORC IS THE SOURCE WORD
         L     2,0(  ,1)         FROM WHICH INFORMATION IS
         L     3,0(  ,2)         OBTAINED.
         L     2,12(  ,1)        MASK IS A MASK WORD.  THE
         L     6,0(  ,2)         BIT POSITIONS OF SWRD
         LR    4,6            CORRESPONDING TO BITS OF MASK
         L     2,4(  ,1)         THAT ARE 1 WILL BE TRNSFERED
         L     8,0(  ,2)         INTO CORRESPONDING SHIFTED
         L     7,=F'2147483647'     POSITIONS OF TWRD.
         XR    6,7         BIT POSTIICNS OF THE SHIFTED TWRD
         SRDL  8,0(3)   CORRESPONDING TO POSITICNS IN WHICH
         XR    8,6           MASK HAS 1'S WILL BE UNCHANGED
         XR    8,4          ALL PARAMETERS ARE FULL WORD.
         SLDL  8,0(3)
         ST    8,0(  ,2)
         LM    2,1 ,28(13)
         MVI   12(13),X'FF'
         BCR   15,14
         END
```

```
      SUBROUTINE RANDU(IX,IY,YFL)
      IY=IX*65539
      IF(IY) 5,6,6
5     IY=IY+2147483647+1
6     YFL=IY
      YFL=YFL*.4656613E-9
      RETURN
      END
```

```
      SUBROUTINE RANDN(IX,S,AM,V)
      A=0.0
      DO 50 I=1,12
      CALL RANDU(IX,IY,Y)
      IX=IY
50    A=A+Y
      V=(A-6.0)*S+AM
      RETURN
      END
```

```
      SUBROUTINE SHAPE(JOB,MODE,NMATNI)
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      COMMON/BLK7/ J(27,64)
      INTEGER*2 J
C  SHAPE FINDS NEXT I/O INTERRUPT POINT EITHER AT I/O
C  INITIATION OR AT I/O COMPLETION
      J11=J(11,JOB)
      CALL UNMIX(15,10,J11,ISHPNO)
      I=2
      IF(J(18,JOB)) 4,5,15
4     J(18,JOB)=0
      GO TO 10
5     IF(MODE) 12,10,12
10    J(26,JOB)=IFIX(F(ISHPNO,I)*J(9,JOB))
      NMATNI=J(26,JOB)
      GO TO 55
12    J(26,JOB)=IFIX(F(ISHPNO,I+20)*J(15,JOB))
      NMATNI=J(26,JOB)
      GO TO 55
15    IF(MODE) 20,40,20
20    ITEMP=IFIX(F(ISHPNO,I+1)*J(9,JOB))
     C          +IFIX(F(ISHPNO,I+20)*J(15,JOB))
     C
      IF(J(26,JOB)-ITEMP) 25,30,30
25    NMATNI=ITEMP-J(26,JOB)
      J(26,JOB)=ITEMP
      GO TO 55
30    I=I+1
      IF(I-20) 20,20,35
35    J(26,JOB)=J(9,JOB)+J(15,JOB)
      GO TO 55
40    ITEMP=IFIX(F(ISHPNO,I)*J(9,JOB))
     C          +IFIX(F(ISHPNO,I+20)*J(15,JOB))
     C
      IF(J(26,JOB)-ITEMP) 50,45,45
45    I=I+1
      IF(I-20) 40,40,35
50    NMATNI=ITEMP-J(26,JOB)
      J(26,JOB)=ITEMP
55    RETURN
      END
```

```
      SUBROUTINE CKCPU(JOBO,JOBN)
      COMMON/BLK12/ICECSZ,ICON,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,     IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,          NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
     C
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,               NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,          ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL
      DO 10  I=1,NCPUS
      IF(ICPU(I)-JOBO)10,20,10
10    CONTINUE
      WRITE(6,12)JOBO,JOBN
12    FORMAT(' CKCPU CAN NOT FIND JOBO=',I4,' TO REPLACE BY
     CJOBN=',I4)
      RETURN
20    ICPU(I)=JOBN
      IPASS(7)=0
      DO 50 I=1,NCPUS
      IF(ICPU(I)) 50,35,50
35    IPASS(7)=IPASS(7)+1
50    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CKIOS(JOBO,JCBN)
      COMMON/BLK12/ICECSZ,ICCN,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,          NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
     C
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,              NREQ(4
     C ,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,           ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL
C TO FIND A JOB IN IPROS, REPLACE IT WITH NEW JOB(JOBN)
      DO 1  I=1,NIOS
      IF(IPROS(I)-JOBO)10,20,10
1     CONTINUE
      RETURN
2     IPROS(I)=JOBN
      IPASS(9)=0
      DO 50 I=1,NIOS
      IF(IPROS(I))50,35,50
35    IPASS(9)=IPASS(9)+1
5     CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LAST3(JOB,KEYTMR,NO1,NO2,NO3,JOBPT,JXCLD)
      COMMON/BLK7/J(27,64)
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),      IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),     MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),        NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,            NREQ(4
     CC,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,          ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,     NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL
C NO1,NO2,NO3=LAST 3 IN CHAIN-NO1 IS ON BOTTOM 0 RETURNED IF
C NONE FOUND
C JOBPT=INDEX OF JOB (PASSED BACK)
C KEYTMR=TMR OR SIMPLEX
C JOB IS THE JOB WE ARE CHECKING BELOW
C LOOK IN ICEC, FIND LAST 3 TO BOUNCE IN CPUS OR LAST IF TMR
      NO1=0
      NO2=0
      NO3=0
      IPNT=ICECS
1     NEXT=ICEC(2,IPNT)
      IF(NEXT-JOB) 2,5,2
2     IPNT=ICEC(5,IPNT)
      IF(IPNT+1) 1,3,1
3     WRITE(6,4) JOB
4     FORMAT(37H ERROR-DID NOT FIND JOB **********NO.,I8)
      RETURN
5     JOBPT=IPNT
      IF(KEYTMR) 7,7,101
C HERE FOR SIMPLEX CPU
7     IPNT=ICEC(5,IPNT)
      IF(IPNT+1) 8,99,8
8     JNAM=ICEC(2,IPNT)
      IF(JNAM.EQ.JXCLD) GO TO 7
      J27=J(27,JNAM)
      CALL UNMIX(3,2,J27,IND3)
      IF(IND3)7,9,7
9     CALL UNMIX(8,7,J27,IND8)
      IF(IND8)7,10,7
10    CALL UNMIX(11,10,J27,IND11)
      IF(IND11)11,11,7
```

```
11      CALL UNMIX(2,1,J27,IND2)
        IF(IND2)7,7,12
C CHECK FOR HOLD
12       CALL UNMIX(6,5,J27,IND6)
        IF(IND6)13,13,7
13       NO3=NO2
        NO2=NO1
        NO1=IPNT
        GO TO 7
99       RETURN
C       JOB IS TMR
101     IPNT=ICEC(5,IPNT)
        IF(IPNT+1) 102,99,102
102     JNAM=ICEC(2,IPNT)
        IF(JNAM.EQ.JXCLD) GO TO 101
        J27=J(27,JNAM)
C CHECK FOR TMR
        CALL UNMIX(11,1,J27,IND11)
        IF(IND11)101,101,103
103     CALL UNMIX(3,1,J27,IND32)
        IF(IND32-1)101,104,101
C JNAM IS INITIATED, IS NOT WAITING CPU
104     CALL UNMIX(8,7,J27,IND8)
        IF(IND8)101,105,101
105     CALL UNMIX(6,5,J27,IND6)
        IF(IND6)106,106,101
106     NO1=IPNT
        GO TO 101
         END
```

```
      SUBROUTINE FEC(KEY,ITIME,IOPRN,IOPND)
C   IF IOPRN=0 THEN OPND IS TO BE MERGED INTO ICEC
C   AFTER ICLK IS GREATER THAN OR EQUAL TO ITIME
C   IF IOPRN=1 THEN A NEW SCHEDULE IS TO BE INITIATED
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      COMMON/BLK12/ICECSZ,ICCN,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,          NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
      C
      COMMON/BLK13/ RANAI
      COMMON/BLK9/ JOVFL(1024)
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,              NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
      C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,           ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL,ICLK
      GO TO (10,200,400),KEY
C   TO HERE FOR INSERTION OF A NEW ENTRY
10    ITEMP=-1
      I=IFECS
      IF(IFECS) 13,13,15
13    K=1
      GO TO 50
15    IF(ITIME-IFEC(I,1)) 25,20,20
20    ITEMP=I
      I=IFEC(I,4)
      IF(I) 25,25,15
25    K=0
30    K=K+1
      IF(K-IFECSZ) 45,45,35
35    WRITE(6,40) IOPRN,IOPND,ITIME
40    FORMAT(20H NO ROOM IN IFEC FOR,I8,2HON,I8,2HAT,I12)
      GO TO 70
45    IF(IFEC(K,4)) 30,50,30
50    IFEC(K,1)=ITIME
      IFEC(K,2)=IOPRN
```

```
      IFEC(K,3)=IOPND
      IFEC(K,4)=I
      IF(IOPND) 54,54,52
52    J27=J(27,ICPND)
      CALL PACK(12,J27,1,1)
      J(27,IOPND)=J27
54    IF(ITEMP) 55,55,60
55    IFECS=K
      GO TO 70
60    IFEC(ITEMP,4)=K
70    RETURN
C  TO HERE FOR REMOVAL OF LEADING ENTRY OR ENTRIES
200   I=IFECS
      IF(I) 210,210,205
205   IF(IFEC(I,1)-ICLK) 215,215,210
210   RETURN
215   IFECS=IFEC(I,4)
      IF(IFEC(I,2)-1) 225,220,225
220   WRITE(6,300) ICLK
300   FORMAT(' FEC-TIME =',I5,' SETS NSCHED')
      NSCHED=1
      GO TO 285
225   IF(IFEC(I,2)) 285,230,285
230   JOBP=IFEC(I,3)
      J27=J(27,JOBP)
      CALL UNMIX(1,0,J27,IND)
      IF(IND) 235,235,280
235   IF(J(2,JOBP)) 240,270,240
240   DO 245 KX=2,5
      IF(J(KX,JOBP).EQ.0) GO TO 270
      IPRED=J(KX,JOBP)
      IF(ICLK-J(8,IPRED)-2*J(14,IPRED)) 245,245,280
245   CONTINUE
      J22=J(22,JOBP)
      CALL UNMIX(1,0,J22,IND)
      IF(IND) 270,270,250
250   DO 255 KX=1,10
255   IVALOV(KX)=0
      CALL OVFLMG(0,JOBP,1,IVALOV,10)
      DO 260 KX=1,10
      IF(IVALOV(KX).EQ.0) GO TO 270
      IPRED=IVALOV(KX)
      IF(ICLK-J(8,IPRED)-2*J(14,IPRED)) 260,260,280
260   CONTINUE
270   WRITE(6,275) ICLK,JOBP
275   FORMAT(' FEC TO CEC-TIME =',I5,' JOB = ',I5)
```

```
      CALL CEC(1,JOBP)
280   J27=J(27,JOBP)
      CALL PACK(0,J27,0,4096)
      J(27,JOBP)=J27
285   IFEC(I,1)=0
      IFEC(I,2)=0
      IFEC(I,3)=0
      IFEC(I,4)=0
      GO TO 200
C  TO HERE FOR FINDING EXIT TIME OF LEADING ENTRY
400   I=IFECS
      IF(I) 405,405,410
405   IPASS(8)=-1
      RETURN
410   IPASS(8)=IFEC(I,1)
      RETURN
      END
```

```
      SUBROUTINE CEC(KNOW,K)
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      COMMON/BLK12/ICECSZ,ICCN,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     Q2,IQ3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,          NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
     C
      COMMON/BLK13/RP01,RP02,RP03,RP04,RP05,RP06,RP07,RP08,R
     CP09,          RP10,RP11
     C
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,               NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,           ISAVE,IVALCV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL,ICLK
      GO TO (10,100,1000,5000),KNOW
C TO HERE FOR PLACING A NEW ENTRY INTO ICEC ARRAY
10    I=1
12    IF(ICEC(2,I)) 15,30,15
15    I=I+1
      IF(I-ICECSZ) 12,12,20
20    WRITE(6,22)
22    FORMAT(17H SUB CEC ALL FULL)
      RETURN
30    ICEC(2,I)=K
      ICEC(3,I)=ICLK
      ICEC(1,I)=0
      ICEC(4,I)=1
      ICEC(5,I)=0
      J27=J(27,K)
      CALL UNMIX(10,9,J27,IND)
      IF(IND.EQ.0) GO TO 32
      IF(J(8,K))32,32,33
32    J17=J(17,K)
      CALL UNMIX(11,0,J17,MMS)
      J(26,K)=IFIX(MMS*MINBLK*RP08)
      NEW=8345
```

```
      NJWM=NJWM+1
      GO TO 34
3000  FORMAT(' CEC HAS PLACED JOB',I4,' INTO ICEC POSITION',
     C(4)
3005  FORMAT(' CEC NOW CALLS SHAPE FOR JOB',I4)
33    CALL SHAPE(K,0,NMATNI)
      NEW=5
34    NSET=NEW
      CALL PACK(0,J27,NEW,NSET)
      J(27,K)=J27
      IPASS(1)=IPASS(1)+1
      KK=IPASS(1)-IPASS(2)
35    IF(IPASS(2)-IQ1) 40,40,50
40    IF(IPASS(2).GE.10) GO TO 42
      IF(IPASS(2).GE.5) GO TO 44
      IF(IPASS(2).GE.3) GO TO 46
      IF(KK) 50,50,70
42    IF(KK-5) 50,70,70
44    IF(KK-3) 50,70,70
46    IF(KK-2) 50,70,70
50    J12=J(12,K)
      CALL UNMIX(14,10,J12,IND1)
      IF(IND1-IP1) 55,70,70
55    IND=J(25,K)-ICLK
      KTEMP1=J(9,K)
      KTEMP2=ICON/MMCT
      D=4.0*FLOAT(KTEMP1)/FLOAT(KTEMP2)
      IF(IND-IFIX(D)) 70,70,75
70    NSCHED=1
75    RETURN
C TO HERE FOR A NEW SCHEDULE
100   NSET=0
      I=1
      ICECS=-1
      IPASS(2)=0
      IPASS(11)=0
      NCNT=0
103   NPL=ICECS
105   IF(ICEC(2,I)) 150,110,150
110   I=I+1
      IF(I-ICECSZ) 105,105,120
120   WRITE(6,125) NCNT
125   FORMAT(' AT THIS NEW SCHEDULE CALL TO CEC ICEC HAD',
     C          I4,1X,'ENTRIES')
     C
      IPASS(1)=NCNT
```

```
      IPASS(2)=NCNT
      IF(NCNT.EQ.0) GO TO 140
      DO 135 KKK=1,ICECSZ
      IF(ICEC(2,KKK).NE.0) WRITE(6,130) KKK,(ICEC(KY,KKK),KY
     C=1,5)
130   FORMAT(6(3X,I7))
135   CONTINUE
140   RETURN
150   JNAM=ICEC(2,I)
      NCNT=NCNT+1
C-----    INTERNAL PRIORITY CALCULATION
160   J27=J(27,JNAM)
      J12=J(12,JNAM)
      CALL UNMIX(15,16,J12,NEXP)
      INP=IFIX(NEXP*RP03)
      IPRT=J(20,JNAM)
      J25=J(25,JNAM)
      IF(J25-ICLK) 165,165,170
165   NDLN=IFIX(RP04*(RP06-(FLOAT(ICLK-J25)/FLOAT(IPRT))))
      GO TO 172
170   NDLN=IFIX(RP04*(RP06-(FLOAT(J25-ICLK)/FLOAT(IPRT))))
172   IF(NDLN.LT.0) NDLN=0
175   INP=INP+NDLN
      ITEMP=ICEC(3,I)
      TEMP=FLOAT(ICLK-ITEMP)/(FLOAT(IPRT)*RP06)
      IF(RP05.GT.1.) GO TO 180
      IWAIT=1./(RP05*(1.+TEMP))
      IF(IWAIT) 178,185,185
178   IWAIT=0
      GO TO 185
180   IF(TEMP.GT.1.) GO TO 182
      IWAIT=IFIX(TEMP*RP05)
      GO TO 185
182   IWAIT=IFIX(RP05*(1.+.5*TEMP))
185   INP=INP+IWAIT
      IF(IFEDBK) 195,195,190
190   J23=J(23,JNAM)
      CALL UNMIX(7,8,J23,MPRF)
      INP=INP+MPRF
195   IF(INP.GT.255) INP=255
      J13=J(13,JNAM)
      CALL PACK(7,J13,INP,255)
      J(13,JNAM)=J13
      ICEC(1,I)=INP
      IF(NSET.EQ.1) GO TO 530
      IPASS(2)=IPASS(2)+1
```

```
      CALL UNMIX(5,4,J27,IND)
      IF(IND.EQ.1) IPASS(11)=IPASS(11)+1
C  NOW LINK JNAM INTO CHAIN ACCORDING TO INTERNAL PRIORITY
200   NTEMP=0
      IF(ICECS) 210,210,220
210   ICECS=I
      ICEC(5,I)=-1
      GO TO 400
220   IF(ICEC(1,I)-ICEC(1,NPL)) 230,230,260
230   NTEMP=NPL
      NPL=ICEC(5,NPL)
      IF(NPL) 240,240,220
240   ICEC(5,NTEMP)=I
      ICEC(5,I)=-1
      GO TO 400
260   IF(NTEMP) 275,265,275
265   ICECS=I
      ICEC(5,I)=NPL
      GO TO 400
275   ICEC(5,NTEMP)=I
      ICEC(5,I)=NPL
400   I=I+1
      IF(I-ICECSZ) 103,103,120
C TO HERE FOR JOB COMPLETION
1000  IF(ICECS) 990,990,1005
990   WRITE(6,995)
995   FORMAT(' ','****ERROR**** CEC WAS CALLED FOR REMOVAL O
     CF A JOB     FROM ICEC AND ICEC IS EMPTY')
     C
      RETURN
1005  JX=ICECS
1010  IF(ICEC(2,JX)-K) 1020,2000,1020
1020  NTEMP=JX
      JX=ICEC(5,JX)
      IF(JX) 1030,1030,1010
1030  WRITE(6,1031) K,K
1031  FORMAT(' ****ERROR**** CEC WAS CALLED FOR COMPLETION O
     CF JOB',I4/  ' BUT JOB',I4,' WAS NOT FOUND IN ICEC CHAI
     CN,CEC WILL '/           ' CHECK TO SEE IF THE JOB IS I
     CN ICEC BUT UNCHAINED')
      I=1
1040  IF(ICEC(2,I)-K) 1060,1050,1060
1050  DO 1055 NR=1,5
1055  ICEC(NR,I)=0
      IPASS(1)=IPASS(1)-1
      RETURN
```

```
106.    I=I+1
        IF(I-ICECSZ) 1040,1040,1070
1070    RETURN
2000    IF(JX-ICECS) 2020,2010,2020
2010    ICECS=ICEC(5,JX)
        GO TO 2030
2020    ICEC(5,NTEMP)=ICEC(5,JX)
2030    DO 2040 NR=1,5
2040    ICEC(NR,JX)=0
        J27=J(27,K)
        CALL PACK(0,J27,0,1)
        J(27,K)=J27
        IPASS(1)=IPASS(1)-1
        IPASS(2)=IPASS(2)-1
        J11=J(11,K)
        CALL UNMIX(10,0,J11,IND1)
        IF(IND1-IR) 2050,2080,2080
2050    J27=J(27,K)
        CALL UNMIX(10,9,J27,IND1)
        IF(IND1) 2070,2060,2070
2060    J17=J(17,K)
        CALL UNMIX(11,0,J17,IND1)
        IF(IND1-MMST) 2071,2080,2080
2070    KK=IPASS(1)-IPASS(2)
        IF(IPASS(2)-IQ1) 2071,2071,9999
2071    IF(IPASS(2).GE.10) GO TO 2072
        IF(IPASS(2).GE.5) GO TO 2074
        IF(IPASS(2).GE.3) GO TO 2076
        IF(KK) 9999,9999,2080
2072    IF(KK-5) 9999,2080,2080
2074    IF(KK-3) 9999,2080,2080
2076    IF(KK-2) 9999,2080,2080
2080    NSCHED=1
9999    RETURN
C-----TO HERE FOR RECALCULATION OF PRIORITIES
5000    NSET=1
        IF(ICECS.LE.0.OR.ICECS.GT.ICECSZ) GO TO 5090
        KK=0
        I=ICECS
5010    IF(I.LE.0) GO TO 5020
        KK=KK+1
        NAPL(KK)=I
        I=ICEC(5,I)
        GO TO 5010
5020    LOOT=KK
        KK=0
```

```
5030  KK=KK+1
      IF(KK.GT.LCNT) GO TO 5050
      I=NAML(KK)
      JNAM=ICEC(2,I)
      GO TO 160
5050  DO 5060 I=1,LCNT
      DO 5060 KK=I,LCNT
      IF(ICEC(1,NAML(I)).GE.ICEC(1,NAML(KK))) GO TO 5060
      ITEMP=NAML(I)
      NAML(I)=NAML(KK)
      NAML(KK)=ITEMP
5060  CONTINUE
      ICECS=NAML(1)
      I=0
5070  I=I+1
      IF(I.EQ.LCNT) GO TO 5080
      ICEC(5,NAML(I))=NAML(I+1)
      GO TO 5070
5080  ICEC(5,NAML(I))=-1
5090  NSET=0
      RETURN
      END
```

```
      SUBROUTINE HLS(KEY,JOB)
      DIMENSION ISUCOV(10)
      INTEGER*2 ISUCOV
      INTEGER RVEF/'RVEF'/,STOV/'STOV'/
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      COMMON/BLK12/ICECSZ,ICCN,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,         NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,         MMAPF,MMSPF
     C
      COMMON/BLK13/RPC1,RPC2,RPC3,RPC4,RPC5,RPC6,RPC7,RPC8,R
     CPC9,         RP10,RP11
     C
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     CCV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),         NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,               NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,           ISAVE,IVALCV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL,ICLK
5     NOWRUN=0
      GO TO (10,500,1000),KEY
C  TO HERE FOR INITIAL J-TABLE SCAN
10    I=1
      DO 250 NXX=1,25
      IF(IDIST(NXX).EQ.RVEF) GO TO 251
250   CONTINUE
251   IDXERN=NXX
      DO 252 NXX=1,25
      IF(IDIST(NXX).EQ.STOV) GO TO 253
252   CONTINUE
253   IDXSTD=NXX
15    I=I+1
      IF(I-64) 20,20,130
20    J1=J(1,I)
      CALL UNMIX(7,0,J1,IND)
      IF(IND) 30,15,30
30    JX=J(10,I)
```

```
      CALL UNMIX(7,0,JX,INM)
      MIX=NPCS*INM+NPCL*(100-INM)
      FMIX=FLOAT(MIX)*.01
      MNARC=IFIX(FLOAT(ICON)/((FLOAT(ITA)+FLOAT(IPCT)*FMIX)*
     C.01))
      JX=J(11,I)
      CALL PACK(0,JX,MNARC,1023)
      J(11,I)=JX
      JX=J(12,I)
      IND=IFIX(MNARC*RP09)
      CALL PACK(0,JX,IND,1023)
      J(12,I)=JX
      JX=J(13,I)
      MNARI=ICON/IFIX(FLOAT(ITA+IRN6*IRN7*IPCT)*.01)
      CALL RANDU(IRN2,IXX,RAN)
      IRN2=IXX
      MNARI=IFIX(FLOAT(MNARI)*(.5+RAN))
      CALL PACK(0,JX,MNARI,1023)
      J(13,I)=JX
      JX=J(21,I)
      IND=MNARI/3
      CALL PACK(0,JX,IND,1023)
      J(21,I)=JX
      J11=J(11,I)
      CALL UNMIX(14,10,J11,ISHPNO)
      KTEMP1=J(9,I)
      NSHPTS=IFIX(F(ISHPNO,21))
      ISTOC=IFIX(MNARC*RP09)
      ISTOI=MNARI/3
      RM=FLOAT(MNARC)
      STDR=FLOAT(ISTOC)
      PM=(FLOAT(KTEMP1))/(F(ISHPNO,21)+1.)
      KTEMP=J(15,I)
      STDP=F(IDXSTD,ISHPNO+1)*FLOAT(KTEMP1)
      CALL NORPRB(PM,STDP,RM,STDR,IDXERN,PROBC)
      IPRT=IFIX((PROBC+(1.-PROBC)*(FLOAT(IFIX(PM/RM))+.6))*(
     CNSHPTS+1))
      RM=FLOAT(MNARI)
      STDR=FLOAT(ISTOI)
      PM=(FLOAT(KTEMP))/F(ISHPNO,21)
      STDP=F(IDXSTD,ISHPNO+21)*FLOAT(KTEMP)
      CALL NORPRB(PM,STDP,RM,STDR,IDXERN,PROBI)
      IPRT=IPRT+IFIX((PROBI+(1.-PROBI)*(FLOAT(IFIX(PM/RM))+.
     C6))*        (NSHPTS))
C
      J(20,I)=IPRT
```

```
      IF(J(14,I).LE.2*IPRT) J(14,I)=2*IPRT
      J27=J(27,I)
      CALL PACK(0,J27,8344,8344)
      J(27,I)=J27
301   JX=J(17,I)
      CALL UNMIX(11,1,JX,MMS)
      LDTIM=IFIX(RP08*(MINBLK*MMS*MMCT))/ICON
      IF(J(14,I)-IFIX(RP07)*(LDTIM+J(20,I))) 375,375,305
305   IF(MMS-MTOTAL/40) 307,310,310
307   J12=J(12,I)
      CALL UNMIX(15,10,J12,IND)
      IF(IND-30) 310,375,375
310   NSUC=0
      KX=5
315   KX=KX+1
      IF(KX-7) 320,320,340
320   JX=J(KX,I)
      CALL UNMIX(7,0,JX,IND)
      IF(IND) 330,330,325
325   NSUC=NSUC+1
330   CALL UNMIX(14,7,JX,IND)
      IF(IND) 315,315,335
335   NSUC=NSUC+1
      GO TO 315
340   JX=J(22,I)
      CALL UNMIX(2,1,JX,IND)
      IF(IND) 370,370,345
345   DO 347 KX=1,10
347   IVALOV(KX)=0
      CALL OVFLMG(1,I,2,IVALOV,10)
      DO 360 KX=1,10
      IF(IVALOV(KX)) 355,360,355
355   NSUC=NSUC+1
360   CONTINUE
370   IF(NSUC-NIOS-NCPUS) 371,372,372
371   KTEMP1=NSUC
      GO TO 373
372   KTEMP1=NIOS+NCPUS
373   KTEMP2=J(14,I)
      KTEMP3=MMS
      JX=J(27,I)
      CALL UNMIX(11,10,JX,IND)
      IF(IND.EQ.1) KTEMP3=3*KTEMP3
      IF((FLOAT(IRN8)/100.)*(FLOAT(KTEMP3)/FLOAT(MTOTAL))
     C              -(FLOAT(LDTIM)*FLOAT(KTEMP1))/FLOAT(KTEMP2
     C)) 375,380,380
```

```
375   JX=J(27,I)
      CALL PACK(0,JX,512,512)
      J(27,I)=JX
380   MX=1
35    MX=MX+1
      IF(MX-5) 40,40,70
40    IPRED=J(MX,I)
      IF(IPRED) 35,35,45
45    IF(J(8,IPRED).EQ.0) GO TO 15
      IF(ICLK-J(8,IPRED)-2*J(14,IPRED)) 35,15,15
70    J22=J(22,I)
      CALL UNMIX(1,0,J22,IND)
      IF(IND) 75,100,75
75    DO 77 MX=1,10
77    IVALOV(MX)=0
      CALL OVFLMG(1,I,1,IVALOV,10)
      WRITE(6,1380)I
1380  FORMAT(' HLS CALLED OVFLMG TO CHECK FOR PREDS OF JOB',
     C I8,'75')
      KX=0
80    KX=KX+1
      IF(KX-10) 85,85,100
85    IF(IVALOV(KX)) 90,100,90
90    IPRED=IVALOV(KX)
      IF(J(8,IPRED).EQ.0) GO TO 15
      IF(ICLK-J(8,IPRED)-2*J(14,IPRED)) 80,15,15
100   J27=J(27,I)
      CALL UNMIX(9,8,J27,IND)
      IF(IND) 105,110,105
105   NPRD=J(14,I)
      GO TO 115
110   ISTD=J(14,I)/3
      TEMP1=FLOAT(ISTD)
      KTEMP2=J(14,I)
      TEMP2=FLOAT(KTEMP2)
      CALL RANDN(IRN3,TEMP1,TEMP2,RAN)
      NPRD=IABS(IFIX(RAN))
      IF(FLOAT(NPRD).GT.TEMP2+3.*TEMP1) NPRD=NPRD/2
115   J(25,I)=J(8,I)+NPRD
      IF(ICLK+2*J(20,I).GT.J(25,I)) J(25,I)=ICLK+2*J(20,I)
      CALL UNMIX(10,9,J27,IND)
      IF(IND.EQ.0) GO TO 116
      IF(J(8,I).EQ.0) GO TO 116
      LCTIM=0
      GO TO 117
116   J17=J(17,I)
```

```
      CALL UNMIX(11,0,J17,MMS)
      LDTIM=IFIX(RP08*(MINBLK*MMS*MMCT))/ICON
117   IND=IFIX(RP06*(J(20,I)+LDTIM))
      IF(J(25,I)-ICLK-IND) 120,120,125
120   CONTINUE
1300  FORMAT(' HLS WILL NOW CALL CEC FOR INSERTION OF JOB',I
     C4/          ' INTO ICEC, NOWRUN=',I2,' 120')
     C
      CALL CEC(1,I)
      GO TO 128
125   NTT=J(25,I)-IND
      WRITE(6,1320) I,NOWRUN,NTT
1320  FORMAT(' HLS WILL NOW CALL FEC FOR INSERTION OF JOB',I
     C4/          ' INTO IFEC, NOWRUN=',I2,' NTT=',I6,' 125'
     C)
      CALL FEC(1,NTT,0,I)
128   IF(NOWRUN) 15,15,129
129   NOWRUN=0
      GO TO 1160
130   RETURN
C TO HERE FOR INITIAL ASSIGNMENT OF PROCESSORS AFTER NEW SCH
CEDULE
500   I=ICECS
      IF(I) 520,520,525
520   WRITE(6,523)
523   FORMAT(' CURRENT EVENTS CHAIN EMPTY ON INITIAL CALL TO
     C HLS 500')
      RETURN
525   NFCPU=0
      DO 530 K=1,NCPUS
530   IF(ICPU(K).EQ.0) NFCPU=NFCPU+1
545   NFIOP=0
      DO 550 K=1,NIOS
550   IF(IPROS(K).EQ.0) NFIOP=NFIOP+1
565   JINDX=ICEC(2,I)
      J27=J(27,JINDX)
      CALL UNMIX(14,13,J27,LOAD)
      IF(LOAD) 575,575,570
570   IF(IPASS(5)-IQ2) 575,575,650
575   IND=0
      CALL PACK(3,IND,J27,1053)
      IF(IND.EQ.5) GO TO 640
      IF(IND.EQ.9) GO TO 645
      IF(IND.EQ.1029) GO TO 590
      IF(IND.EQ.1033) GO TO 620
      GO TO 650
```

```
590   IF(NFCPU-3) 650,595,595
595   CALL CKCPU(0,JINDX)
      NFCPU=NFCPU-1
      CALL CKCPU(0,JINDX)
      NFCPU=NFCPU-1
600   CALL CKCPU(0,JINDX)
      NFCPU=NFCPU-1
      CALL PACK(0,J27,2,134)
      J(27,JINDX)=J27
605   INDC=ICEC(4,I)-(ICEC(4,I)/2)*2
      IF(INDC.EQ.0) ICEC(4,I)=ICEC(4,I)+1
      IPASS(3)=IPASS(3)+1
      IF(LOAD.EQ.0) IPASS(4)=IPASS(4)+1
      IF(NFCPU) 650,610,650
610   IF(NFIOP) 650,655,650
620   IF(NFIOP-3) 650,625,625
625   CALL CKIOS(0,JINDX)
      NFIOP=NFIOP-1
      CALL CKIOS(0,JINDX)
      NFIOP=NFIOP-1
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+2
630   CALL CKIOS(0,JINDX)
      NFIOP=NFIOP-1
      MASK=2186
      IF(LOAD.EQ.1) MASK=2184
      CALL PACK(0,J27,2178,MASK)
      J(27,JINDX)=J27
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+1
      GO TO 605
635   IF(IND-2) 640,645,640
640   IF(NFCPU.GT.0) GO TO 600
      IF(NFIOP) 655,655,650
645   IF(NFIOP.GT.0) GO TO 630
      IF(NFCPU) 655,655,650
650   I=ICEC(5,I)
      IF(I) 655,655,365
653   FORMAT(' HLS HAS ASSIGNED CPU NO.',I3,' TO JOB',I3)
654   FORMAT(' HLS HAS ASSIGNED IOP NO.',I3,' TO JOB',I3)
655   DO 670 IX=1,NCPUS
      IF(ICPU(IX).NE.0) WRITE(6,653) IX,ICPU(IX)
670   CONTINUE
675   DO 680 IX=1,NIOS
      IF(IPROS(IX).NE.0) WRITE(6,654) IX,IPROS(IX)
680   CONTINUE
      RETURN
C  TO HERE FOR JOB COMPLETION
```

```
1000  CONTINUE
1340  FORMAT(' HLS WILL NOW CALL CEC FOR REMOVAL OF JOB',I4/
     C                ' FROM ICEC FOLLOWING JOB COMPLETION, 1000
     C')
      CALL CEC(3,JOB)
      J27=J(27,JOB)
      CALL UNMIX(7,6,J27,IND)
      IF(IND.EQ.0) IPASS(3)=IPASS(3)-1
      IPASS(4)=IPASS(4)-1
      J(8,JOB)=ICLK
      J(18,JOB)=0
      J(26,JOB)=0
      J10=J(10,JOB)
      CALL PACK(0,J10,0,32640)
      J(10,JOB)=J10
      IF(J(2,JOB)) 1019,1004,1019
1004  CALL UNMIX(9,8,J27,IND)
      IF(IND) 1010,1005,1010
1005  ISTD=J(14,JOB)/3
      TEMP1=FLOAT(ISTD)
      KTEMP2=J(14,JOB)
      TEMP2=FLOAT(KTEMP2)
      CALL RANDN(IRN3,TEMP1,TEMP2,RAN)
      NPRD=IABS(IFIX(RAN))
      GO TO 1011
1010  NPRD=J(14,JOB)
1011  IF(J(8,JOB)-J(25,JOB)) 1012,1013,1013
1012  J(25,JOB)=J(25,JOB)+NPRD
      GO TO 1015
1013  J(25,JOB)=J(8,JOB)+NPRD
1015  IF(ICLK+2*J(20,JOB).GT.J(25,JOB)) J(25,JOB)=ICLK+2*J(2
     C0,JOB)
      CALL UNMIX(10,9,J27,IND)
      IF(IND.EQ.0) GO TO 1016
      IF(J(8,I).EQ.0) GO TO 1016
      LDTIM=0
      GO TO 1017
1016  J17=J(17,JOB)
      CALL UNMIX(11,0,J17,MMS)
      LDTIM=IFIX(RP08*(MINBLK*MMS*MMCT))/ICON
1017  NTT=J(25,JOB)-IFIX(RP06*(J(20,JOB)+LDTIM))
1360  FORMAT(' HLS WILL NOW CALL FEC FOR INSERTION OF JOB',I
     C4/              ' INTO IFEC AFTER JOB COMPLETION, NTT=',I6
     C,' 1015')
      CALL FEC(1,NTT,0,JOB)
1019  J27=J(27,JOB)
```

```
      CALL UNMIX(10,9,J27,IND)
      IF(IND) 1080,1020,1080
1020  J1=J(1,JOB)
      CALL PACK(0,J27,16,16)
      J(27,JOB)=J27
      CALL UNMIX(15,7,J1,IND)
      IF(IND) 1025,1030,1025
1025  CONTINUE
1026  FORMAT(' HLS WILL NOW CALL MEMRLS TO RELEASE BLOCK',I4
     C,/           ' FROM JOB',I4)
     C
      CALL MEMRLS(JOB,IND)
      CALL PACK(7,J1,0,255)
      J(1,JOB)=J1
1030  J19=J(19,JOB)
      CALL UNMIX(8,0,J19,IND)
      IF(IND) 1035,1040,1035
1035  CONTINUE
      CALL MEMRLS(JOB,IND)
      CALL PACK(0,J19,0,255)
      J(19,JOB)=J19
1040  J17=J(17,JOB)
      CALL UNMIX(15,11,J17,INDU)
      CALL UNMIX(12,8,J19,INDL)
      IND=0
      CALL PACK(0,IND,INDL,15)
      CALL PACK(4,IND,INDU,15)
      IF(IND) 1045,1050,1045
1045  CONTINUE
      CALL MEMRLS(JOB,IND)
      CALL PACK(8,J19,0,15)
      J(19,JOB)=J19
      CALL PACK(11,J17,0,15)
      J(17,JOB)=J17
1050  J22=J(22,JOB)
      CALL UNMIX(8,7,J22,IND)
      IF(IND) 1055,1080,1055
1055  DO 1057 KX=1,10
1057  IVALOV(KX)=0
      CALL OVFLMG(1,JOB,8,IVALOV,10)
      KX=0
1060  KX=KX+1
      IF(KX-10) 1065,1065,1080
1065  IF(IVALOV(KX)) 1070,1080,1070
1070  IND=IVALOV(KX)
      WRITE(6,1026) IND,JOB
```

```
      CALL MEMRLS(JOB,IND)
      IVALOV(1)=KX-1
      CALL OVFLMG(3,JOB,8,IVALOV,1)
      GO TO 1060
1080  CONTINUE
C  WILL NOW UPDATE PREDECESSOR FIELDS OF SUCCESSORS OF JOB
1090  JJX=5
      IRETRN=0
      IFLAG=1
1093  IFLAG=-IFLAG
      IF(IFLAG) 1095,1105,1105
1095  JJX=JJX+1
      IF(JJX-7) 1100,1100,1175
1100  JWRD=J(JJX,JOB)
      CALL UNMIX(7,0,JWRD,IDXSUC)
      GO TO 1107
1105  CALL UNMIX(14,7,JWRD,IDXSUC)
1107  IF(IDXSUC) 1110,1093,1110
1110  NJ27=J(27,IDXSUC)
      CALL UNMIX(13,12,NJ27,IND)
      IF(IND.EQ.1) GO TO 1160
      CALL UNMIX(1,0,NJ27,IND)
      IF(IND.EQ.1) GO TO 1160
      KX=1
1113  KX=KX+1
      IF(KX-5) 1115,1115,1130
1115  IF(J(KX,IDXSUC)) 1113,1113,1120
1120  IF(J(KX,IDXSUC)-JOB) 1125,1113,1125
1125  IPRED=J(KX,IDXSUC)
      IF(J(8,IPRED).EQ.0) GO TO 1160
      IF(ICLK-J(8,IPRED)-2*J(14,IPRED)) 1113,1160,1160
1130  J22=J(22,IDXSUC)
      CALL UNMIX(1,0,J22,ISPILL)
      IF(ISPILL) 1135,1170,1135
1135  DO 1137 LX=1,10
1137  IVALOV(LX) =0
      CALL OVFLMG(1,IDXSUC,1,IVALOV,10)
      KX=0
1140  KX=KX+1
      IF(KX-10) 1145,1145,1170
1145  IF(IVALOV(KX)) 1147,1170,1147
1147  IF(IVALOV(KX)-JOB) 1150,1140,1150
1150  IPRED=IVALOV(KX)
      IF(J(8,IPRED).EQ.0) GO TO 1160
      IF(ICLK-J(8,IPRED)-2*J(14,IPRED)) 1140,1160,1160
1160  IF(JJX-7) 1093,1093,1165
```

```
1165  IF(IRETRN) 1205,1205,1190
1170  I=IDXSUC
      NOWRUN=1
C  WILL NOW BRANCH TO 100 FOR CALCULATION OF TARGET TIME
C  FOR JOB IDXSUC AND INSERTION INTO ICEC OR IFEC
      GO TO 100
1175  J22=J(22,JOB)
      CALL UNMIX(2,1,J22,IRETRN)
      IF(IRETRN) 1205,1205,1180
1180  DO 1181 MX=1,10
1181  ISUCOV(MX)=0
1185  CALL OVFLMG(1,JOB,2,ISUCOV,10)
      MX=0
1190  MX=MX+1
      IF(MX-10) 1195,1195,1205
1195  IF(ISUCOV(MX)) 1200,1190,1200
1200  IDXSUC=ISUCOV(MX)
      GO TO 1110
1205  RETURN
      END
```

```
      SUBROUTINE NORPRB(PM,STDP,RM,STDR,IDXERN,PROB)
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      PROB=0.0
      IF(PM-3.*STDP-RM+3.*STDR) 5,15,12
5     IF(PM+3.*STDP-RM+3.*STDR) 6,6,700
6     PROB=1.0
      RETURN
700   IF(RM-3.*STDR-PM) 710,710,720
710   I=IFIX(10.*((RM-3.*STDR-PM)/STDP)-.5)
      GO TO 7
720   I=IFIX(10.*((RM-3.*STDR-PM)/STDP)+.5)
7     IF(I) 8,9,10
8     PROB=F(IDXERN,-I)
      GO TO 15
9     PROB=.5
      GO TO 15
10    PROB=1.-F(IDXERN,I)
      GO TO 15
12    IF(PM-3.*STDP-RM-3.*STDR) 15,13,13
13     PROB=0.0
      RETURN
15    BL=AMAX1(RM-3.*STDR,PM-3.*STDP)
      BU=AMIN1(RM+3.*STDR,PM+3.*STDP)
2     BLRM=(BL-RM)/STDR
      BURM=(BU-RM)/STDR
      BLPM=(BL-PM)/STDP
      BUPM=(BU-PM)/STDP
      DXP=10.*BLPM
      DXR=10.*BLRM
      IF(STDR-STDP) 23,23,25
23    CNP=1.
      CNR=STDP/STDR
      IF(DXP) 231,231,232
231   DXP=FLOAT(IFIX(DXP-.5))
      GO TO 30
232   DXP=FLOAT(IFIX(DXP+.5))
      GO TO 30
25    CNR=1.
      CNP=STDR/STDP
      IF(DXR) 251,251,252
251   DXR=FLOAT(IFIX(DXR-.5))
      GO TO 30
252   DXR=FLOAT(IFIX(DXR+.5))
30    IDXPL=IFIX(DXP)
      IDXPU=IFIX(DXP+CNP)
```

```
      IF(INDXPU.GT.40) GO TO 165
      INDXRL=IFIX(DXR)
      INDXRU=IFIX(DXR+CNR)
      IF(INDXRU.GT.40) GO TO 165
      IF(STDR-STDP) 35,35,40
35    DIFL=ABS(DXR-INDXRL)
      DIFU=ABS(DXR+CNR-INDXRU)
      GO TO 45
40    DIFL=ABS(DXP-INDXPL)
      DIFU=ABS(DXP+CNP-INDXPU)
45    IF(INDXPL) 55,46,75
C  INDXPL IS 0, INDXPU IS POSITIVE OR 0
46    IF(INDXPU) 47,47,48
47    X1=0.0
      IF(DIFU.GE..1) X1=X1+DIFU*(.5-F(IDXERN,1))
      GO TO 50
48    X1=.5-F(IDXERN,INDXPU)
      IF(CNP.EQ.1.) GO TO 95
      IF(DIFU.GE..1) X1=X1+DIFU*(F(IDXERN,INDXPU)-F(IDXERN,I
     CNDXPU+1))
5     IF(DIFL-.1) 95,51,51
51    IF(DXP-INDXPL) 52,52,53
52    X1=X1+DIFL*(.5-F(IDXERN,1))
      GO TO 95
53    X1=X1-DIFL*(.5-F(IDXERN,1))
      GO TO 95
55    IF(INDXPU) 60,65,70
C  INDXPL AND INDXPU ARE BOTH NEGATIVE
6     X1=F(IDXERN,-INDXPU)-F(IDXERN,-INDXPL)
      IF(CNP.EQ.1.) GO TO 95
      IF(DIFU.GE..1) X1=X1-DIFU*(F(IDXERN,-INDXPU)-F(IDXERN,
     C-INDXPU+1))
      IF(DIFL.GE..1) X1=X1+DIFL*(F(IDXERN,-INDXPL)-F(IDXERN,
     C-INDXPL+1))
      GO TO 95
C  INDXPL IS NEGATIVE, INDXPU IS 0
65    X1=.5-F(IDXERN,-INDXPL)
      IF(CNP.EQ.1.) GO TO 95
      IF(DIFU-.1) 69,66,66
66    IF(DXP+CNP-INDXPU) 67,67,68
67    X1=X1-DIFU*(.5-F(IDXERN,1))
      GO TO 69
68    X1=X1+DIFU*(.5-F(IDXERN,1))
69    IF(DIFL.GE..1) X1=X1+DIFL*(F(IDXERN,-INDXPL)-F(IDXERN,
     C-INDXPL+1))
      GO TO 95
```

```
C  INDXPL IS NEGATIVE, INDXPU IS POSITIVE
7      X1=1.-F(IDXERN,-INDXPL)-F(IDXERN,INDXPU)
       IF(CNP.EQ.1.) GO TO 95
       IF(DIFL.GE..1) X1=X1+DIFL*(F(IDXERN,-INDXPL)-F(IDXERN,
      C-INDXPL+1))
       IF(DIFU.GE..1) X1=X1+DIFU*(F(IDXERN,INDXPU)-F(IDXERN,I
      CNDXPU+1))
       GO TO 95
C  BOTH INDXPL AND INDXPU ARE POSITIVE
75     X1=F(IDXERN,INDXPL)-F(IDXERN,INDXPU)
       IF(CNP.EQ.1.) GO TO 95
       IF(DIFL.GE..1) X1=X1-DIFL*(F(IDXERN,INDXPL)-F(IDXERN,I
      CNDXPL+1))
       IF(DIFU.GE..1) X1=X1+DIFU*(F(IDXERN,INDXPU)-F(IDXERN,I
      CNDXPU+1))
95     IF(INDXRL) 105,96,115
C  INDXPL IS 0, INDXPU IS POSITIVE OR 0
96     IF(INDXRU) 97,97,98
97     X2=.5
       IF(DIFU.GE..1) X2=X2-(DIFU*(.5-F(IDXERN,1)))*.5
       GO TO 100
98     X2=(.5+F(IDXERN,INDXRU))*.5
       IF(CNR.EQ.1.) GO TO 145
       IF(DIFU.GE..1) X2=X2-(DIFU*(F(IDXERN,INDXRU)-F(IDXERN,
      CINDXRU+1     )))*.5
      C
1      IF(DIFL-.1) 145,101,101
1 1    IF(DXR-INDXRL) 102,102,103
1 2    X2=X2+(DIFL*(.5-F(IDXERN,1)))*.5
       GO TO 145
1 3    X2=X2-(DIFL*(.5-F(IDXERN,1)))*.5
       GO TO 145
1 5    IF(INDXRU) 110,106,112
C  INDXRL IS NEGATIVE, INDXRU IS 0
1 6    X2=(1.5-F(IDXERN,-INDXRL))*.5
       IF(CNR.EQ.1.) GO TO 145
       IF(DIFL.GE..1) X2=X2+(DIFL*(F(IDXERN,-INDXRL)-F(IDXERN
      C,-INDXRL+1   )))*.5
      C
       IF(DXR+CNR-INDXRU) 107,107,108
1 7    X2=X2+(DIFU*(.5-F(IDXERN,1)))*.5
       GO TO 145
1 8    X2=X2-(DIFU*(.5-F(IDXERN,1)))*.5
       GO TO 145
C  WITH INDXRU AND INDXRL ARE NEGATIVE
11     X2=1.-(F(IDXERN,-INDXRL)+F(IDXERN,-INDXRU))*.5
```

```
      IF(CNR.EQ.1.) GO TO 145
      IF(DIFL.GE..1) X2=X2+(DIFL*(F(IDXERN,-INDXRL)-F(IDXERN
     C,-INDXRL+1  )))*.5
     C
      IF(DIFU.GE..1) X2=X2+(DIFU*(F(IDXERN,-INDXRU)-F(IDXERN
     C,-INDXRU+1  )))*.5
     C
      GO TO 145
C  INDXRL IS NEGATIVE,INDXRU IS POSITIVE
112   X2=(1.-F(IDXERN,-INDXRL)+F(IDXERN,INDXRU))*.5
      IF(CNR.EQ.1.) GO TO 145
      IF(DIFL.GE..1) X2=X2+(DIFL*(F(IDXERN,-INDXRL)-F(IDXERN
     C,-INDXRL+1  )))*.5
     C
      IF(DIFU.GE..1) X2=X2-(DIFU*(F(IDXERN,INDXRU)-F(IDXERN,
     CINDXRU+1    )))*.5
     C
      GO TO 145
C  BOTH INDXRL AND INDXRU ARE POSITIVE
115   X2=(F(IDXERN,INDXRL)+F(IDXERN,INDXRU))*.5
      IF(CNR.EQ.1.) GO TO 145
      IF(DIFL.GE..1) X2=X2-(DIFL*(F(IDXERN,INDXRL)-F(IDXERN,
     CINDXRL+1    )))*.5
     C
      IF(DIFU.GE..1) X2=X2-(DIFU*(F(IDXERN,INDXRU)-F(IDXERN,
     CINDXRU+1    )))*.5
     C
145   X=X1*X2
      IF(X.GT..00C0031) PROB=PROB+X
      DXP=DXP+CNP
      DXR=DXR+CNR
      IF(CNR-CNP) 160,150,150
150   IF(DXR-10.*BURM) 30,30,155
155   RETURN
160   IF(DXP-10.*BUPM) 30,30,165
165   RETURN
      END
```

```
      SUBROUTINE LLS(KEY,JOB)
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      COMMON/BLK12/ICECSZ,ICCN,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMCDS,NPCL,NPCS,         NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
C
      COMMON/BLK13/RP01,RP02,RP03,RP04,RP05,RP06,RP07,RP08,R
     CP09,          RP10,RP11
C
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(24),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     CCV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,                NREQ(4
     C ,24),NTP(24),NTR(24),NUCA(24)
C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,          ISAVE,IVALCV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL,ICLK
      DIMENSION IHOLD(6)
      INTEGER*2 IHOLD
C        KEY = 1 = I/O INTERRUPT INITIATION
C              2 = I/O INTERRUPT COMPLETION
C              3 = JOB COMPLETION
      FORMAT(' IN LLS,JOB=',I4,' KEY=',I3,'IPASS(1-4),IPROS&
     C(CPU ARE')
   2  FORMAT(4(5X,I7))
   4  FORMAT(16(2X,I3))
   5  FORMAT(3(2X,I3))
      GO TO (1,100,200),KEY
1     CALL SHAPE(JOB,J,NMATNI)
      J27=J(27,JOB)
      CALL PACK(J,J27,128,128)
      J(27,JOB)=J27
      CALL UNMIX(7,6,J27,IN17)
      IF(IN17.EQ.1) GO TO 9
      J1 =J(1 ,JOB)
      CALL UNMIX(15,7,J1 ,INP)
      IF(INP-IP2) 200,2 0,2
2     IF(IPASS(2)-IQ3)3,200,200
3     J27=J(27,JOB)
```

```
      IF(NMATNI-IN1) 10,200,200
9     CALL PACK(0,J27,0,64)
      J(27,JOB)=J27
      GO TO 12
10    CALL PACK(0,J27,32,32)
      J(27,JOB)=J27
12    J27=J(27,JOB)
      CALL UNMIX(11,10,J27,IND11)
      IF(IND11) 20,20,30
20    IF(IPASS(9)) 70,70,55
30    IF(IPASS(9)-3) 70,50,50
50    CALL CKIOS(0,JOB)
      CALL CKIOS(0,JOB)
55    CALL CKIOS(0,JOB)
61    CALL PACK(0,J27,2048,2060)
      J(27,JOB)=J27
600   CONTINUE
601   FORMAT(' EXIT LLS AFT I/O INT.INIT.,JOB=',I4,' IPROS&I
     CCPU ARE')
      RETURN
70    CALL PACK(0,J27,8,8)
      J(27,JOB)=J27
      NP=ICECS
72    IF(ICEC(2,NP)-JOB)74,80,74
74    NP=ICEC(5,NP)
      GO TO 72
80    IACT=ICEC(4,NP)
      CALL PACK(0,IACT,0,3)
      ICEC(4,NP)=IACT
      IPASS(3)=IPASS(3)-1
      GO TO 600
200   J27=J(27,JOB)
C   PRIORITY OF JOB,P,NO. JOBS AWAITING CPU)Q,NO. I/O OPS)N
      CALL UNMIX(11,10,J27,IND11)
      IF(IND11) 205,205,500
205   INDEX=ICECS
210   IF(INDEX) 215,215,220
215   CALL CKCPU(JOB,0)
      GO TO 12
220   NJOB=ICEC(2,INDEX)
      NJ27=J(27,NJOB)
      CALL UNMIX(5,4,NJ27,IND5)
      IF(IND5) 230,230,225
225   INDEX=ICEC(5,INDEX)
      GO TO 210
230   CALL UNMIX(3,2,NJ27,IND3)
```

```
      IF(IND3) 225,225,235
235   CALL UNMIX(2,1,NJ27,IND2)
      IF(IND2) 240,240,245
240   IF(IPASS(4)-NI) 245,225,225
245   CALL UNMIX(11,10,NJ27,IND11)
      IF(IND11) 250,250,300
250   CALL CKCPU(JOB,NJOB)
      IF(IND2.EQ.0) IPASS(4)=IPASS(4)+1
260   IPASS(3)=IPASS(3)+1
      CALL PACK(0,NJ27,2,254)
      J(27,NJOB)=NJ27
      IACT=ICEC(4,INDEX)
      CALL PACK(0,IACT,2,3)
      ICEC(4,INDEX)=IACT
      GO TO 12
300   NFREE=0
      DO 301 KKK=1,NCPUS
301   IF(ICPU(KKK).EQ.0) NFREE=NFREE+1
      IF(NFREE-2) 310,305,305
305   CALL CKCPU(0,NJOB)
      CALL CKCPU(0,NJOB)
      GO TO 252
310   CALL LAST3(NJOB,0,N01,N02,N03,JOBPT,JOB)
315   IF(NFREE) 350,350,320
320   IF(N01) 225,225,325
325   CALL CKCPU(0,NJOB)
      GO TO 360
350   IF(N02) 225,225,355
355   JOBB=ICEC(2,N02)
      CALL CKCPU(JOBB,NJOB)
      J27B=J(27,JOBB)
      CALL PACK(0,J27B,68,68)
      J(27,JOBB)=J27B
      IACT=ICEC(4,N02)
      CALL PACK(0,IACT,0,3)
      ICEC(4,N02)=IACT
      IPASS(3)=IPASS(3)-1
360   JOBA=ICEC(2,N01)
      CALL CKCPU(JOBA,NJOB)
      J27A=J(27,JOBA)
      CALL PACK(0,J27A,68,68)
      J(27,JOBA)=J27A
      IACT=ICEC(4,N01)
      CALL PACK(0,IACT,0,3)
      ICEC(4,N01)=IACT
      IPASS(3)=IPASS(3)-1
```

```
      GO TO 250
C  JOB IS A TMR JOB, 3 CPUS RELEASED
500   INDEX=ICECS
      DO 505 KKK=1,6
505   IHOLD(KKK)=0
510   IF(INDEX) 555,555,515
515   NJOB=ICEC(2,INDEX)
      NJ27=J(27,NJOB)
      CALL UNMIX(5,4,NJ27,IND5)
      IF(IND5) 520,520,550
520   CALL UNMIX(3,2,NJ27,IND3)
      IF(IND3) 550,550,525
525   CALL UNMIX(2,1,NJ27,IND2)
      IF(IND2) 530,530,535
530   IF(IPASS(4)+IHOLD(4)-NI) 532,550,550
532   IHOLD(4)=IHOLD(4)+1
535   CALL UNMIX(11,10,NJ27,IND11)
      IF(IND11) 540,540,545
540   IHOLD(5)=IHOLD(5)+1
      IHOLD(IHOLD(5))=INDEX
      IF(IHOLD(5)-3) 550,565,565
545   CALL CKCPU(JOB,NJOB)
      CALL CKCPU(JOB,NJOB)
      IHOLD(5)=1
      IHOLD(1)=INDEX
      GO TO 565
550   INDEX=ICEC(5,INDEX)
      GO TO 510
555   IF(IHOLD(5)-1) 556,558,560
556   CALL CKCPU(JOB,0)
558   CALL CKCPU(JOB,0)
560   CALL CKCPU(JOB,0)
565   IHOLD(6)=IHOLD(6)+1
      IF(IHOLD(6)-IHOLD(5)) 570,570,12
570   INDEX=IHOLD(IHOLD(6))
      NJOB=ICEC(2,INDEX)
      NJ27=J(27,NJOB)
      CALL CKCPU(JOB,NJOB)
      CALL UNMIX(2,1,NJ27,IND2)
      IF(IND2.EQ.0) IPASS(4)=IPASS(4)+1
      IPASS(3)=IPASS(3)+1
      CALL PACK(0,NJ27,3,255)
      J(27,NJOB)=NJ27
      IACT=ICEC(4,INDEX)
      CALL PACK(0,IACT,2,3)
      ICEC(4,INDEX)=IACT
```

```
      GO TO 565
C  I/O INTERRUPT COMPLETION-COMING FROM PROGRAM EXECUTOR
1000  J27=J(27,JOB)
1100  FORMAT(' LLS WILL NOW CALL SHAPE TO UPDATE NIP FOR JOB
     C',I4,        ' 1000')
      C
      CALL SHAPE(JOB,1,NMATNI)
      DO 1005 KKK=1,6
1005  IHOLD(KKK)=0
      CALL CKIOS(JOB,0)
      CALL UNMIX(14,13,J27,LOAD)
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)-1
      CALL UNMIX(11,10,J27,IND11)
      IF(IND11) 1010,1010,1015
1010  NFREE=0
      DO 1012 KKK=1,NIOS
1012  IF(IPROS(KKK).EQ.0) NFREE=NFREE+1
      IF(NFREE-2) 1020,1200,1020
1015  CALL CKIOS(JOB,0)
      CALL CKIOS(JOB,0)
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)-2
      NFREE=3
1020  INDEX=ICECS
1025  IF(INDEX) 1060,1060,1030
1030  NJOB=ICEC(2,INDEX)
      NJ27=J(27,NJOB)
      CALL UNMIX(5,3,NJ27,IND54)
      IF(IND54-1) 1035,1037,1035
1035  INDEX=ICEC(5,INDEX)
      GO TO 1025
1037  CALL UNMIX(14,13,NJ27,LOAD)
      IF(LOAD) 1040,1040,1038
1038  IF(IPASS(5)-IQ2) 1040,1040,1035
1040  CALL UNMIX(11,10,NJ27,IND11)
      IF(IND11) 1050,1050,1045
1045  IF(NFREE.LT.3) GO TO 1035
      CALL CKIOS(0,NJOB)
      CALL CKIOS(0,NJOB)
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+2
      IHOLD(5)=1
      IHOLD(1)=INDEX
      GO TO 1060
1050  IHOLD(5)=IHOLD(5)+1
      IHOLD(IHOLD(5))=INDEX
      IF(IHOLD(5)-3) 1055,1060,1060
1055  IF(NFREE-1) 1060,1060,1035
```

```
1060   IHOLD(6)=IHOLD(6)+1
       IF(IHOLD(6)-IHOLD(5)) 1065,1065,1200
1065   INDEX=IHOLD(IHOLD(6))
       NJOB=ICEC(2,INDEX)
       NJ27=J(27,NJOB)
       CALL CKIOS(0,NJOB)
       IPASS(3)=IPASS(3)+1
       CALL UNMIX(14,13,NJ27,LOAD)
       IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+1
       CALL PACK(0,NJ27,2048,2056)
       J(27,NJOB)=NJ27
       IACT=ICEC(4,INDEX)
       CALL PACK(0,IACT,2,3)
       ICEC(4,INDEX)=IACT
       GO TO 1060
C  RESET JOB'S STATUS, RESET IPROS IF NECESSARY
C  LOOK FOR CPU(S) FOR JOB, GET OUT
1200   J27=J(27,JOB)
       CALL PACK(0,J27,0,2176)
       J(27,JOB)=J27
       CALL UNMIX(6,5,J27,IND6)
       IF(IND6) 1210,1210,1205
1205   CALL PACK(0,J27,0,32)
       J(27,JOB)=J27
       GO TO 1600
1210   NFREE=0
       DO 1212 KKK=1,NCPUS
1212   IF(ICPU(KKK).EQ.0) NFREE=NFREE+1
       CALL UNMIX(11,10,J27,IND11)
       IF(IND11) 1285,1285,1215
1215   IF(NFREE-3) 1220,1295,1295
1220   NEED=3-NFREE
       CALL LAST3(JOB,0,NO1,NO2,NO3,JPNT,JOB)
       IF(NEED-2) 1260,1255,1225
1225    IF(NO3) 1265,1265,1230
1230   NJOB=ICEC(2,NO3)
       CALL CKCPU(NJOB,JOB)
       NJ27=J(27,NJOB)
       CALL PACK(0,NJ27,68,68)
       J(27,NJOB)=NJ27
       IACT=ICEC(4,NO3)
       CALL PACK(0,IACT,0,3)
       ICEC(4,NO3)=IACT
       IPASS(3)=IPASS(3)-1
1235   NJOB=ICEC(2,NO2)
       CALL CKCPU(NJOB,JOB)
```

```
      NJ27=J(27,NJOB)
      CALL PACK(0,NJ27,68,68)
      J(27,NJOB)=NJ27
      IACT=ICEC(4,NJ2)
      CALL PACK(0,IACT,0,3)
      ICEC(4,NO2)=IACT
      IPASS(3)=IPASS(3)-1
1240  NJOB=ICEC(2,NO1)
1245  CALL CKCPU(NJOB,JOB)
      NJ27=J(27,NJOB)
      CALL PACK(0,NJ27,68,68)
      J(27,NJOB)=NJ27
      IACT=ICEC(4,NO1)
      CALL PACK(0,IACT,0,3)
      ICEC(4,NO1)=IACT
      IPASS(3)=IPASS(3)-1
125   CALL UNMIX(2,1,J27,IND2)
      IF(IND2.EQ.1) GO TO 1251
      IPASS(4)=IPASS(4)+1
1251  CALL PACK(0,J27,2,8198)
      J(27,JOB)=J27
      GO TO 1600
1255  IF(NO2) 1265,1265,1257
1257  CALL CKCPU(0,JOB)
      GO TO 1235
1260  IF(NO1) 1265,1265,1280
1265  CALL LAST3(JOB,1,NO1,NO2,NO3,JPNT,JOB)
      IF(N 1) 1275,1275,1270
1270  NJOB=ICEC(2,NO1)
      CALL CKCPU(NJOB,JOB)
      CALL CKCPU(NJOB,JOB)
      GO TO 1245
1275  CALL PACK(0,J27,4,8196)
      J(27,JOB)=J27
      IACT=ICEC(4,JPNT)
      CALL PACK(0,IACT,0,3)
      ICEC(4,JPNT)=IACT
      IPASS(3)=IPASS(3)-1
      GO TO 1600
1280  CALL CKCPU(0,JOB)
      CALL CKCPU(0,JOB)
      GO TO 1240
1285  IF(NFREE) 1290,1290,1300
1290  CALL LAST3(JOB,0,NO1,NO2,NO3,JPNT,JOB)
      IF(N 1) 1275,1275,1240
1295  CALL CKCPU(0,JOB)
```

```
      CALL CKCPU(0,JOB)
1300  CALL CKCPU(0,JOB)
      GO TO 1250
1600  CONTINUE
1605  FORMAT(' EXIT LLS AFT IOO CMPLT,JOB=',I4,' IPROS&ICPU
     CARE')
      RETURN
C  JOB COMPLETION
2000  J27=J(27,JOB)
      CALL UNMIX(11,10,J27,IND11)
      IF(IND11)2002,2002,4000
C  THE JOB IS SIMPLEX
2002  CALL UNMIX(8,7,J27,IND8)
      IF(IND8)2005,2005,3000
C  JOB IS CPU MODE
2005  NJP=ICECS
      GO TO 2010
2008  NJP=ICEC(5,NJP)
2010  IF(NJP) 2850,2850,2012
2012  NJOB=ICEC(2,NJP)
      NJ27=J(27,NJOB)
      CALL UNMIX(5,4,NJ27,IND5)
      IF(IND5) 2008,2013,2008
2013  CALL UNMIX(3,2,NJ27,IND3)
      IF(IND3)2008,2008,2014
C  NJOB IS AWAITING CPU
2014  CALL UNMIX(11,10,NJ27,IND11)
      IF(IND11) 2016,2016,2100
C  NJOB IS SIMPLEX
2016  CALL CKCPU(JOB,NJOB)
      GO TO 2875
C  NJOB WAS TMR
2100  NE=0
      DO 2104 I=1,NCPUS
      IF(ICPU(I))2104,2102,2104
2102  NE=NE+1
2104  CONTINUE
      IF(NE-2)2200,2106,2106
C TWO CPUS ARE EMPTY(R MORE)
2106  CALL CKCPU(0,NJOB)
2108  CALL CKCPU(0,NJOB)
      GO TO 2016
C  LESS THAN 2 CPUS ARE EMPTY
2200  CALL LAST3(NJOB,0,NO1,NO2,NO3,JOBPT,JOB)
      IF(NE)2220,2220,2210
C  ONE CPU IS EMPTY
```

```
2210  IF(NO1)2008,2008,2212
C  NO1 IS BOUNCABLE-CAN FIT JOB IN
C  NOW FIND NAMES OF NO1
2212  JOB1=ICEC(2,NO1)
      CALL CKCPU(JOB1,NJOB)
C  NOW UPDATE JOB1-SET 3 AND 7
      J27=J(27,JOB1)
      CALL UNMIX(3,2,J27,IND3)
      CALL PACK(3,J27,68,68)
      J(27,JOB1)=J27
      IACT=ICEC(4,NO1)
      CALL PACK(3,IACT,0,3)
      ICEC(4,NO1)=IACT
      IPASS(3)=IPASS(3)-1
      IF(NE) 2016,2016,2108
C  NO CPUS ARE EMPTY
2220  IF(NO2)2008,2008,2222
C  TWO JOBS ARE BOUNCABLE
2222  JOB2=ICEC(2,NO2)
      CALL CKCPU(JOB2,NJOB)
      J27=J(27,JOB2)
      CALL PACK(3,J27,68,68)
      J(27,JOB2)=J27
      IACT=ICEC(4,NO2)
      CALL PACK(3,IACT,0,3)
      ICEC(4,NO2)=IACT
      IPASS(3)=IPASS(3)-1
      GO TO 2212
C  UPDATE ICPU WITH ZERO-NOTHING TO REPLACE JOB IN CPUS
2500  CALL CKCPU(JOB,0)
      GO TO 9999
C  UPDATE NJOB
2975  CALL UNMIX(2,1,NJ27,IND2)
      IF(IND2.EQ.0) IPASS(4)=IPASS(4)+1
      CALL PACK(2,NJ27,2,70)
      J(27,NJOB)=NJ27
      IACT=ICEC(4,NJP)
      CALL PACK(2,IACT,2,3)
      ICEC(4,NJP)=IACT
      IPASS(3)=IPASS(3)+1
      GO TO 9999
C  MODE WAS I/O, JOB WAS SIMPLEX
3000  NJP=ICEC5
      GO TO 3004
3002  NJP=ICEC(5,NJP)
3004  IF(NJP) 3006,3006,3029
```

```
C  COULD NOT FIND JOB TO TAKE PROCESSOR
3006  CALL CKIOS(JOB,0)
3007  CALL UNMIX(6,5,J27,IND6)
      IF(IND6) 9999,9999,2005
C   IS NOT END OF CHAIN-NJP POINTS TO NJOB
3020  NJOB=ICEC(2,NJP)
      NJ27=J(27,NJOB)
      CALL UNMIX(5,3,NJ27,IND54)
      IF(IND54-1) 3002,3022,3002
3022  CALL UNMIX(14,13,NJ27,LOAD)
      IF(LOAD) 3024,3024,3023
3023  IF(IPASS(5)-102) 3024,3002,3002
C  NJOB IS WAITING IOP AND NOT WAITING MEMORY
3024  CALL UNMIX(11,10,NJ27,IND11)
      IF(IND11)3026,3026,3100
C  NJOB IS SIMPLEX AND NEEDS I/O PROCESSOR
3026  CALL CKIOS (JOB,NJOB)
      CALL PACK(0,NJ27,2048,2056)
      J(27,NJOB)=NJ27
      IACT=ICEC(4,NJP)
      CALL PACK(0,IACT,2,3)
      ICEC(4,NJP)=IACT
      IPASS(3)=IPASS(3)+1
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+1
      GO TO 3007
C  NJOB IS TMR JOB-CHECK FOR EMPTY I/O PROCESSORS
3100  NEMTY=0
      DO 3104 I=1,NIOS
      IF(IPROS(I))3104,3102,3104
3102  NEMTY=NEMTY+1
3104  CONTINUE
      IF(NEMTY-2)3006,3110,3110
C  WE HAVE 2 (OR MORE-MORE IS BAD) EMPTY I/O PROCESSORS
3110  CALL CKIOS(0,NJOB)
      CALL CKIOS(0,NJOB)
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+2
      GO TO 3026
C  JOB IS TMR JOB
4000  CALL UNMIX(8,7,J27,IND8)
      IF(IND8)4002,4002,4500
C  JOB IS CPU MODE-SEARCH FOR 3 HIGHEST PRIORITY JOBS
C  WHICH WANT CPUS-IF ONE IS TMR, GIVE IT TO HIM OTHERWISE
C  GIVE TO 3 SIMPLEX-INITIATION IRRELEVANT
4002  NJP=ICECS
      DO 4003 KKK=1,6
4003  IHOLD(KKK)=0
```

```
      NRDY=0
      GO TO 4006
4004  NJP=ICEC(5,NJP)
4006  IF(NJP) 4300,4300,4007
4007  NJOB=ICEC(2,NJP)
      NJ27=J(27,NJOB)
      CALL UNMIX(5,4,NJ27,IND5)
      IF(IND5) 4008,4008,4004
4008  CALL UNMIX(3,2,NJ27,IND3)
      IF(IND3) 4004,4004,4012
C  WE NOW HAVE AN ACCEPTABLE JOB-IT WANTS A CPU
4012  CALL UNMIX(11,10,NJ27,IND11)
      IF(IND11)4014,4014,4100
C  NJOB IS SIMPLEX
4014  NRDY=NRDY+1
      IHOLD(NRDY)=NJP
      IF(NRDY-3)4004,4016,4016
C  WE HAVE FOUND 3 SIMPLEX JOBS WANTING A CPU-FIND NAMES
C  UPDATE, PUT IN CPU
4016  DO 4030 I=1,NRDY
      NJOB=ICEC(2,IHOLD(I))
      NJ27=J(27,NJOB)
      CALL UNMIX(2,1,NJ27,IND2)
      IF(IND2.EQ.0) IPASS(4)=IPASS(4)+1
      CALL PACK(0,NJ27,2,254)
      J(27,NJOB)=NJ27
      CALL CKCPU(JOB,NJOB)
      IPASS(3)=IPASS(3)+1
      IACT=ICEC(4,IHOLD(I))
      CALL PACK(0,IACT,2,3)
4030  ICEC(4,IHOLD(I))=IACT
      GO TO 9999
C  FOUND ACCEPTABLE TMR JOB-NAME NJOB,INDEX NJP ON ICEC
4100  CALL CKCPU(JOB,NJOB)
      CALL CKCPU(JOB,NJOB)
      CALL CKCPU(JOB,NJOB)
      IACT=ICEC(4,NJP)
      CALL PACK(0,IACT,2,3)
      ICEC(4,NJP)=IACT
      CALL UNMIX(2,1,NJ27,IND2)
      IF(IND2) 4105,4105,4110
4105  IPASS(4)=IPASS(4)+1
4110  IPASS(3)=IPASS(3)+1
      CALL PACK(0,NJ27,2,254)
      J(27,NJOB)=NJ27
      GO TO 9999
```

```
C  FOUND END OF CHAIN BEFORE TMR OR 3 JOBS
4300  IF(NRDY-1)4302,4306,4308
C NO JOBS FOUND
4302  CALL CKCPU(JOB,0)
      CALL CKCPU(JOB,0)
      CALL CKCPU(JOB,0)
      GO TO 9999
C  ONE JOB FOUND
4306  CALL CKCPU(JOB,0)
C  FOUND 2 SIMPLEX JOBS WAITING CPU
4308  CALL CKCPU(JOB,0)
      GO TO 4016
C  JOB IS TMR AND RELEASING 3 I/O PROCESSORS
4500  NJP=ICECS
      DO 4501 KKK=1,6
4501  IHOLD(KKK)=0
      NRDY=0
      GO TO 4510
4505  NJP=ICEC(5,NJP)
4510  IF(NJP) 4600,4600,4515
4515  NJOB=ICEC(2,NJP)
      NJ27=J(27,NJOB)
      CALL UNMIX(5,3,NJ27,IND54)
      IF(IND54-1) 4505,4517,4505
4517  CALL UNMIX(14,13,NJ27,LOAD)
      IF(LOAD) 4520,4520,4518
4518  IF(IPASS(5)-IQ2) 4520,4505,4505
C  NJOB IS WAITING IOP AND NOT WAITING MEMORY
4520  CALL UNMIX(11,10,NJ27,IND11)
      IF(IND11)4525,4550,4525
C  NJOB WAS TMR
4525  CALL CKIOS (JOB,NJOB)
      CALL CKIOS (JOB,NJOB)
      CALL CKIOS (JOB,NJOB)
      IACT=ICEC(4,NJP)
      CALL PACK(0,IACT,2,3)
      ICEC(4,NJP)=IACT
      IPASS(3)=IPASS(3)+1
      NJ27=J(27,NJOB)
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+1
      CALL PACK(0,NJ27,2048,2056)
      J(27,NJOB)=NJ27
4530  CALL UNMIX(6,5,J27,IND6)
      IF(IND6)4535,4535,4002
4535  GO TO 9999
C  NJOB WAS SIMPLEX
```

```
4550  NRDY=NRDY+1
      IHOLD(NRDY)=NJP
      IF(NRDY-3)4505,4560,4560
C  HAVE FOUND 3 SIMPLEX JOBS WANTING I/O PROCESSORS
4560  DO 4590 I=1,NRDY
      NJOB=ICEC(2,IHOLD(I))
      NJ27=J(27,NJOB)
      CALL PACK(0,NJ27,2048,2056)
      J(27,NJOB)=NJ27
      IACT=ICEC(4,IHOLD(I))
      CALL PACK(0,IACT,2,3)
      ICEC(4,IHOLD(I))=IACT
      IPASS(3)=IPASS(3)+1
      CALL UNMIX(14,13,NJ27,LOAD)
      IF(LOAD.EQ.1) IPASS(5)=IPASS(5)+1
4590  CALL CKIOS(JOB,NJOB)
      GO TO 4530
C  CAME OUT OF CHAIN BEFORE FINDING 3 SIMPLEX JOBS
4600  IF(NRDY-1)4605,4615,4625
C  NO JOBS FOUND
4605  CALL CKIOS(JOB,0)
      CALL CKIOS(JOB,0)
      CALL CKIOS(JOB,0)
      GO TO 4530
C  1 JOB FOUND
4615  CALL CKIOS(JOB,0)
4625  CALL CKIOS(JOB,0)
      GO TO 4560
9999  J27=J(27,JOB)
      CALL PACK(0,J27,0,2303)
      J(27,JOB)=J27
8005  FORMAT(' EXIT LLS AFT CMPLT OF JOB',I4,'IPROS&ICPU ARE
     C')
      RETURN
      END
```

```
      SUBROUTINE PEX(IDXGAU)
C   PEX EXECUTES PROGRAMS BY GENERATING A RANDOM NUMBER
C   OF MEMORY ACCESS REQUESTS FOR EACH ACTIVE JOB.  IT THEN
C   GRANTS A NUMBER OF ACCESSES FROM EACH MEMORY MODULE
C   TO EACH ACTIVE JOB ON THE BASIS OF THE NUMBER
C   REQUESTED, JOB PRIORITY AND NUMBER OF AVAILABLE FROM
C   EACH MODULE.  IT ALSO KEEPS UP WITH THE EXECUTION
C   POINT OF EACH PROGRAM AND A PREDICTION OF THE
C   NUMBER OF AVAILABLE ACCESSES FROM EACH MEMORY MODULE
      COMMON/BLK3/ IDIST(25),F(25,41),NDIST
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      COMMON/BLK12/ICECSZ,ICON,IN1,IFECSZ,IPCT,IP1,IP2,IC1,I
     CC2,IC3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,          NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
C
      COMMON/BLK13/RP01,RP02,RP03,RP04,RP05,RP06,RP07,RP08,R
     CP09,          RP10,RP11
C
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(24),        IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,                NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,            ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      COMMON/BLK17/ ISTCNT,ISTAT(6,400),IUTL(6,200)
      INTEGER*2 ISTAT,IUTL
      COMMON/BLK18/ AN,CUU,PLU,BWU,SPU,BCUU,BPUU,BBWU,BSPU,A
     CCUU,APUU,    ABWU,ASPU,BQSIZ,AQSIZ
C
      INTEGER*2 J,JOVFL,ICLK
      DIMENSION MSPACE(24)
      INTEGER*2 ICON1,MODSIZ,MSPACE,JMSTOT,IACCS,INP,HFIX
      MODSIZ=MTOTAL/NMODS
      ICON1=ICON/MMCT
      CALL UNMIX(2,1,IS,IWRTB)
      IND1=ICLK-IPASS(6)
      IND=IND1-(IND1/2)*2
      IF(IND.EQ.0.AND.IND1.NE.0) CALL CEC(4,0)
```

```
      IND=ICLK-(ICLK/10)*10
      IF(IND.NE.0) GO TO 4
      WRITE(6,15  ) ICLK,(IPASS(KKK),KKK=1,10)
15    FORMAT(' IN PEX-TIME,IPASS(1-10) ARE',11(1X,I4))
4     IPASS(3)=0
      NTAL=0
      I=ICFCS
      IF(I) 5,5,7
5     WRITE(6,6)
6     FORMAT(' ICFC WAS EMPTY UPON ENTERING PEX')
      RETURN
7     DO 1  IY=1,NMODS
      NT (IY)=0
      NTP(IY)=0
      DO 1  IZ=1,ICECSZ
1     [illegible]C(IZ,IY)=0
      GO TO [illegible]
      IF(I) [illegible],[illegible],[illegible]
      IC4=ICEC(4,I)
      CALL [illegible]X([illegible],  ,IC4,IND)
      IF(I[illegible]-1) [illegible] ,45,43
      [illegible]=IC[illegible](  ,I)
      [illegible]
      I[illegible](  ,I)=(ICEC(4,I)-IND)+1
      J[illegible]=IC[illegible]C(  ,I)
      IPASS(3)=IPASS(3)+1
      J[illegible]=J(1,J[illegible]A )
      IF(  [illegible]  .GT.[illegible]. ) GO TO 111
      DO [illegible] IY=1,[illegible]ODS
      MSPACE(IY)=0
      CALL [illegible]TY(15,7,J[illegible],MBLK)
      [illegible]=[illegible]
      IF( [illegible]LK) [illegible] ,[illegible] ,55
      [illegible]ST=MALC(MBLK,3)/MODSIZ+1
      [illegible]=(MALC(MBLK,3)+MALC(MBLK,5)-1)/MODSIZ+1
      IF(MODE[illegible]-M[illegible]ST-1) [illegible],[illegible],56
      IST=[illegible]ST+1
      IN[illegible]=[illegible]-1
      DO [illegible]7 KKX=IST,I[illegible]
7     MSPACE(KKX)=MSPACE(KKX)+MODSIZ
      MSPACE(MODED)=MSPACE(MODED)+MALC(MBLK,3)+MALC(MBLK,5)
                 -(MODED-1)*MODSIZ

      [illegible]=MALC(MBLK,[illegible])+MALC(MBLK,5)
      [illegible]=MODST*MODSIZ
      IF([illegible]-K[illegible]) [illegible] ,[illegible] ,[illegible]
```

ORIGINAL PAGE IS OF POOR QUALITY

```
590   MSPACE(MODST)=MSPACE(MODST)+MALC(MBLK,5)
      GO TO 592
591   MSPACE(MODST)=MSPACE(MODST)+KTEMP2-MALC(MBLK,3)
592   GO TO (60,70,80,92),KCNT
60    J19=J(19,JNAM)
      KCNT=2
      CALL UNMIX(8,0,J19,MBLK)
      IF(MBLK.NE.0) GO TO 55
70    J17=J(17,JNAM)
      KCNT=3
      CALL UNMIX(15,11,J17,MBLKU)
      CALL UNMIX(12,8,J19,MBLKL)
      MBLK=MBLKU*16+MBLKL
      IF(MBLK.NE.0) GO TO 55
80    J22=J(22,JNAM)
      KCNT=4
      CALL UNMIX(8,7,J22,MSPIL)
      IF(MSPIL-1) 85,85,105
85    DO 86 IX=1,10
86    IVALOV(IX)=0
      CALL OVFLMS(0,JNAM,8,IVALOV,10)
      IX=1
90    MBLK=IVALOV(IX)
      IF(MBLK.NE.0) GO TO 55
92    IX=IX+1
      IF(IX-10) 90,90,97
97    WRITE(6,100)JNAM
100   FORMAT(36H MORE THAN 10 BLOCKS ASSIGNED TO JOB,I7)
105   JMSTOT=0
      DO 110 IY=1,NMODS
      JMSTOT=JMSTOT+MSPACE(IY)
110   CONTINUE
      MTAL=MTAL+JMSTOT
111   CALL RANDU(IRN1,IY,RAN)
      IRN1=IY
      IF(RAN-.5) 114,112,112
112   INDEX=IFIX(80.0*(RAN-.5))+1
      VALUE=F(IDXGAU,INDEX)
      GO TO 115
114   INDEX=IFIX(80.0*(.5-RAN))+1
      VALUE=-F(IDXGAU,INDEX)
115   J27=J(27,JNAM)
      CALL UNMIX(8,7,J27,MODE)
      IF(MODE)130,120,130
120   J11=J(11,JNAM)
      CALL UNMIX(15,0,J11,INDM)
```

```
      J12=J(12,JNAM)
      CALL UNMIX(10,0,J12,INDS)
      STD=FLOAT(INDS)
      AM=FLOAT(INDM)
      IACCS=IABS(IFIX(VALUE*STD+AM))
      GO TO 140
13    J27=J(27,JNAM)
      CALL UNMIX(14,13,J27,LOAD)
      IF(LOAD) 132,132,131
131   J17=J(17,JNAM)
      CALL UNMIX(11,0,J17,MMS)
      IACCS=(ICON1*MMS)/MODSIZ
      GO TO 140
132   J13=J(13,JNAM)
      CALL UNMIX(10,0,J13,INDM)
      J21=J(21,JNAM)
      CALL UNMIX(10,2,J21,INDS)
      STD=FLOAT(INDS)
      AM=FLOAT(INDM)
      IACCS=IABS(IFIX(VALUE*STD+AM))
14    IDIFF=J(26,JNAM)-J(18,JNAM)
      IF(IACCS-IDIFF) 142,144,143
142   IF(IDIFF-IACCS-INDS) 143,143,144
143   IACCS=IDIFF
144   J1 =J(1 ,JNAM)
      CALL UNMIX(15,7,J1 ,IND)
      INP=IND
      IF(NP1.GE.2.) GO TO 710
      J27=J(27,JNAM)
      CALL UNMIX(11,1 ,J27,IND11)
      IF(IND11) 145,15 ,145
145   IACCS=3*IACCS
15    JX=1
155   IF(MSPACE(JX)) 157,157,160
157   NREQ(I,JX)=
      GO TO 170
16    IF(JMSTOT.EQ. ) GO TO 162
      NR (I,JX)=(MSPACE(JX)*IACCS+JMSTOT-1)/JMSTOT
      GO TO 163
162   WRITE(6,15 ) ICLK,I,(ICEC(KKK,I),KKK=1,5)
16    FORMAT(' P X,TIME =',I5,'****ERROR****',6(2X,I6))
163   NT (JX)=NTR(JX)+NREQ(I,JX)
      JTP(JX)=JTP(JX)+INP
17    JX=JX+1
      IF(J -JMODS) 155,155,4
7     J27=J(27,JNAM)
```

```
      CALL UNMIX(14,13,J27,LOAD)
      IF(LOAD) 710,710,705
705   ITOTA=J(26,JNAM)
      GO TO 715
710   ITOTA=J(0,JNAM)+J(15,JNAM)
715   J17=J(17,JNAM)
      CALL UNMIX(11,0,J17,MMS)
      ISTRT=(MMS*J(18,JNAM))/ITOTA
      CALL UNMIX(11,1?,J27,IND)
      NTAL=NTAL+MMS+2*IND*MMS
      IF(IND) 72?,72?,725
72?   NMBRU=1
      GO TO 730
725   NMBRU=3
73    NMBR=
735   NMBR=NMBR+1
      IF(NMBR.GT.NMBRU) GO TO 4?
      LENGTH=?
      IKEEP=
7?    IF(NMBRU.GT.1) GO TO 82?
74?   IKEEP=IKEEP+1
      GO TO (745,75 ,755,76J),IKEEP
745   J1=J(1,JNAM)
      CALL UNMIX(15,7,J1,INDX6)
      GO TO 77J
75    J1?=J(19,JNAM)
      CALL UNMIX(6, ,J19,INDX6)
      GO TO 77
755   J19=J(19,JNAM)
      J17=J(17,JNAM)
      CALL UNMIX(15,11,J17,INDXBU)
      CALL UNMIX(12,8,J19,INDXBL)
      INDX6=INDXBU*16+INDXBL
      GO TO 77?
76    J22=J(22,JNAM)
      CALL UNMIX(6,7,J22,IND6)
      IF(IND6.LE.) GO TO 775
      ISKIP=
      GO TO 765
765   ISKIP=ISKIP+1
768   IVALOV(1)=ISKIP
      CALL OVFLWS(1,JNAM,8,IVALOV,1)
      INDX6=IVALOV(1)
      IF(INDX6.LE.0) GO TO 77?
      GO TO 7P
77    IF(INDX6.GT.0) GO TO 785
```

```
///     WRITE (6,776)
        GO TO 4
776     FORMAT(' **** ERROR **** MEMORY BLOCK INDEX ERROR REX,
       ,776')
        IF(NERR.EQ.1) GO TO 779
        CALL INDEX(15,13,INDXB,IND)
        IF(IND.EQ.INDXB) GO TO 765
        INDXB=INDXB-IND*8192
        IRET=1
        GO TO 77
765     IF(LENGTH+MALC(INDXB,5).GE.ISTRT) GO TO 79
        LENGTH=LENGTH+MALC(INDXB,5)
        IF(NERR.NE.1) GO TO 84
        IF(IRET-4) 743,765,769
79      IADDR=MALC(INDXB,3)+ISTRT-LENGTH
        IDX=IADDR/MDSIZ+1
        MAP(I,MODIDX)=IACCS
        MAP(MODIDX)=MAP(MODIDX)+IACCS
        MAP(MODIDX)=MAP(MODIDX)+IND
        GO TO 78
        GO TO (745,75,743),NMSP
        IF(IRET) 76,74,765
        JX=JX+1
        IF(JX-NMOD) 145,145,26
        IF(IBMCTR(JX)-IBMS) 2 ,193,19
        MAS(JX)=(MAS(JX)*(IRN5-1)+IRN5/2)/IRN5
        IF( BMCTR(JX)-ICC+1) 195,21,21
        MAS(JX)=MAS(JX)+(IRN5-IBMCTR(JX))/IRN5
        GO TO 21
        IBMCTR(JX)=IBMCTR(JX)+1
        MAS(JX)=(MAS(JX)*(IBMCTR(JX)-1))/IRN5
                +((IRN5-IBMCTR(JX))*(CON1*MAS(JX))/(IRN5*M
       1)
        GO TO 193
        I(JX)=
        I(JX)=
        IF(IT(JX)) 163,163,214
        IF(I) 193,193,22
        IF(ICC(4,I)-(ICC(4,I)/2)*2) 25 ,253,223
        JX=ICC(2,I)
        IF(MAP(I,JX)) 25 ,25 ,225
        IF(MAP(JX)-ICC+1) 23 ,25 ,23
        JX=I(I ,JX+1)
        CALL IY(15,7,JX,IRET)
```

```
      INP=IND
      IF(NREQ(I,JX)-(INP*ICON1)/NTP(JX)) 24?,240,235
235   MTP1(JX)=MTP1(JX)+INP
      GO TO 25?
24?   NUCA(JX)=NUCA(JX)+(INP*ICON1)/NTP(JX)-NREQ(I,JX)
25?   I=ICEC(5,I)
      GO TO 215
26?   IF(IWRTB.EQ.0) GO TO 263
      WRITE(6,9?')
9'    FORMAT(' NAA AND MAS FOLLOW')
      WRITE(6,9?5) (NAA(IX),IX=1,NMODS)
      WRITE(6,9?5) (MAS(IX),IX=1,NMODS)
9?5   FORMAT(24(1X,I4))
263   NTAG=
      NTREQ=
      CCC=?.?
      PCC=?.
      I=ICECS
265   IF(I) 455,455,27?
27?   IF(ICEC(4,I)-(ICEC(4,I)/2)*2)45?,450,271
271   JNAM=ICEC(?,I)
      NREQI=
      NAGI=
      J??=J(1?,JNAM)
      CALL UNMIX(15,7,J??,IND)
      INP=IND
      JX=
27?   JX=JX+1
2?    IF(JX-NMODS) 285,285,33?
285   IF(NREQ(I,JX)) 275,275,29?
29?   NREQI=NREQI+NREQ(I,JX)
      IF(NTR(JX)-ICON1) 31?,31?,295
295   IF(NREQ(I,JX)-(INP*ICON1)/NTP(JX)) 31?,31?,30?
3?    NAG=(INP*ICON1)/NTP(JX)+(INP*NUCA(JX))/MTP1(JX)
3?5   IF(NAG-NREQ(I,JX)) 315,315,31?
31?   NAG=NREQ(I,JX)
315   NAGI=NAGI+NAG
      GO TO 275
33?   NTAG=NTAG+NTAGI
      NTREQ=NTREQ+NREQI
      J27=J(27,JNAM)
      CALL UNMIX(11,1?,J27,IND11)
      CALL UNMIX(6,7,J27,MODE)
      IDIFF=J(26,JNAM)-J(18,JNAM)
      IF(IND11.EQ.1) IDIFF=3*IDIFF
      IF(NAGI.GE.IDIFF-1) NAGI=IDIFF
```

```
      AUU=1.0
      IF(NREQI.GT.0) AUU=FLOAT(NAGI)/FLOAT(NREQI)
      IF(MODE) 345,345,335
335   IF(IND11) 355,355,340
340   PUU=PUU+3.0*AUU
      GO TO 370
345   IF(IND11) 360,360,350
350   CUU=CUU+3.0*AUU
      GO TO 370
355   PUU=PUU+AUU
      GO TO 365
360   CUU=CUU+AUU
365   J(18,JNAM)=J(18,JNAM)+NAGI
      GO TO 375
370   J(18,JNAM)=J(18,JNAM)+(NAGI+2)/3
375   CONTINUE
      FORMAT(' COMPLETION COUNTDOWN FOR JOB',I3,' =',I6)
      J(16,JNAM)=J(16,JNAM)+1
      IACT=IOEC(4,I)
      J27=J(27,JNAM)
      CALL UNMIX(14,13,J27,LOAD)
      IF(LOAD.EQ.1) GO TO 380
      IF(J(18,JNAM)-J(9,JNAM)-J(15,JNAM)) 380,400,400
380   IF(J(18,JNAM)-J(26,JNAM)) 450,385,385
385   IF(MODE) 395,390,395
390   CALL PACK( ,IACT,4,4)
      IOEC(4,I)=IACT
      GO TO 450
395   CALL PACK( ,IACT,8,8)
      IOEC(4,I)=IACT
      IF(LOAD.EQ.0) GO TO 450
      J(26,JNAM)=0
      J(16,JNAM)=-1
      J(27,JNAM)=J27
      GO TO 40
400   CALL PACK( ,IACT,16,16)
      IOEC(4,I)=IACT
405   I=IOEC(5,I)
      GO TO 265
410   I=IOEC5
      GO TO 457
415   I=IOEC(5,I)
417   IF(I) 420,420,460
460   IACT=IOEC(4,I)
      CALL UNMIX(5,2,IACT,IND)
      IF(IND) 456,456,465
```

```
465   J27=J(27,ICEC(2,I))
      JNAM=ICEC(2,I)
      CALL PACK(J27,J27,12)
      J(27,JNAM)=J27
      CALL PACK(J27,IACT,1,31)
      ICEC(4,I)=IACT
      IF(IND-2) 470,475,480
470   CONTINUE
      FORMAT(' PEX NOW CALLS LLS FOR I/O INTRPT. INIT. FOR J
     OB',I4)
      CALL LLS(1,JNAM)
      GO TO 456
475   CONTINUE
      FORMAT(' PEX NOW CALLS LLS FOR I/O CMPTL. FOR JOB',I4)
      CALL LLS(2,JNAM)
      GO TO 456
480   ISTCNT=ISTCNT+1
      ISTAT(1,ISTCNT)=JNAM
      ISTAT(2,ISTCNT)=ICEC(3,I)
      ISTAT(3,ISTCNT)=ICLK
      J27=J(27,JNAM)
      CALL UNMIX(1,9,J27,IAD)
      IF(IND.EQ.1) GO TO 491
      IF(J(8,JNAM).EQ.0) GO TO 481
      ISTAT(4,ISTCNT)=J(20,JNAM)
      GO TO 482
481   J17=J(17,JNAM)
      CALL UNMIX(11,0,J17,MMS)
      J13=J(13,JNAM)
      CALL UNMIX(16,0,J13,MNARI)
      ISTAT(4,ISTCNT)=J(20,JNAM)+(1024*MMS)/MNARI
482   ISTAT(5,ISTCNT)=J(25,JNAM)
      ISTAT(6,ISTCNT)=J(16,JNAM)
      FORMAT(/,' COMPLETION OF JOB',I4,' ICEC ENTRY TIME'/
     1       ' PRESENT TIME,ESTIMATED MIN PERIODS,TARGE
     CT TIME, AND ACTUAL'/    ' PROCESSING PERIODS FOLLOW'/,
     2(9X,I8))
      J(16,JNAM)=0
      J(25,JNAM)=0
      I=ICEC(5,I)
      CALL MLS(3,JNAM)
      J27=J(27,JNAM)
      CALL UNMIX(7,6,J27,IPRE)
      IF(IPRE) 485,485,484
485   CALL PACK(J27,J27,23,3)
      J(27,JNAM)=J27
```

```
      GO TO 457
465   CONTINUE
      FORMAT(' PEX WILL NOW CALL LLS FOR COMPLETION OF JOB',
     .I4)
      CALL LLS(3,JNAM)
      GO TO 457
49    CCU=CUU/FLOAT(NCPUS)
      PCU=PUU/FLOAT(NIOS)
      BWU=FLOAT(NTAG)/FLOAT(NMODS*ICCN1)
      SPU=FLOAT(MTAL)/FLOAT(MTOTAL)
595   FORMAT(/,' CUU,PUU,BWU,SPU ARE',4(4X,F7.3))
      BCUU=(9.*BCUU+CCU)/10.
      BPUU=(9.*BPUU+PCU)/10.
      BBWU=(9.*BBWU+BWU)/10.
      BSPU=(9.*BSPU+SPU)/10.
      ITEMP=IPASS(1)
      BQSIZ=(9.*BQSIZ+ITEMP)*.1
      AN=AN+1.
      ACUU=(AN*ACUU+CCU)/AN
      APCU=(AN*APUU+PCU)/AN
      ABWU=(AN*ABWU+BWU)/AN
      ASPU=(AN*ASPU+SPU)/AN
      AQSIZ=(AN*AQSIZ+ITEMP)/AN
      N=AN
      FORMAT(' CUR. AV. UTLS ARE',4(3X,F8.4))
      IDX=ICLK/20
      IF(ICLK-INDX*20 ) 598,596,598
596   IUTL(1,INDX)=ICLK
      IUTL(2,INDX)=IFIX(100.*ACUU)
      IUTL(3,INDX)=IFIX(100.*APUU)
      IUTL(4,INDX)=IFIX(100.*ABWU)
      IUTL(5,INDX)=IFIX(100.*ASPU)
      IUTL(6,INDX)=IFIX(10.*AQSIZ)
      IUTL(7,INDX+20)=IFIX(100.*BCUU)
      IUTL(8,INDX+20)=IFIX(100.*BPUU)
      IUTL(9,INDX+20)=IFIX(100.*BBWU)
      IUTL(10,INDX+20)=IFIX(100.*BSPU)
      IUTL(11,INDX+20)=IFIX(10.*BQSIZ)
598   RETURN
      END
```

```
      SUBROUTINE NFMAL(KEY,JOB,NSPACE)
C  NFMAL IS THE NON FEEDBACK MEMORY ALLOCATOR
C  JOB IS THE NAME OF THE JOB FOR WHICH SPACE IS TO BE ALLOC
CATED
C  NSPACE IS THE AMOUNT OF SPACE TO BE ALLOCATED
C  IF NSPACE.NE.0 THEN NSPACE IS THE AMOUNT TO BE ALLOCATED
C  IF NSPACE.EQ.0 THE AMOUNT IS CONTAINED IN THE J TABLE
C  IF KEY=0 THEN ALLOCATE MEMORY TO JOB
C  IF KEY =1 THEN ALLOCATE MEMORY TO ALL JOBS IN
C  ICEC THAT ARE WAITING FOR MEMORY
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      INTEGER*2 ICLK
      COMMON/BLK12/ICECSZ,ICCN,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,          NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
     C
      COMMON/BLK13/ RANAI
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,               NREQ(4
     C,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,           ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL
      DIMENSION ITKOF(3,3),NTKOF(3,3),L(3)
      INTEGER*2 ITKOF,NTKOF,L
      CALL UNMIX(1,0,IS,IWRTA)
      WRITE(6,1520) ICLK,NJWM,NFB(1),IPASS(10)
1520  FORMAT(' IN NFMAL-TIME,NJWM,NFB,&AMT. FREE ARE',4(1X,I
     C4))
      MODSIZ=MTOTAL/NMODS
      IF(KEY) 1010,1050,1010
1010  KX=ICECS
1012  IF(KX) 460,460,1015
1015  JOB=ICEC(2,KX)
      J27=J(27,JOB)
      CALL UNMIX(5,4,J27,IND)
1020  IF(IND) 1050,1025,1050
```

```
1025  KX=ICEC(5,KX)
1027  FORMAT(' NFMAL NOW SCANS ICEC FOR NEXT JOB WAITING MEM
     CORY')
      GO TO 1012
1050  K1=0
      JJJ=MAVLS
      DO 1 NC=1,3
      L(NC)=0
      DO 1 MC=1,3
      ITKOF(NC,MC)=0
1     CONTINUE
      I=1
      IF(NSPACE)2,4,2
2     MMS=NSPACE
      GO TO 10
4     J22=J(22,JOB)
      CALL UNMIX(7,6,J22,IND7)
      IF(IND7) 8,8,6
6     CALL OVFLMG(0,JOB,7,IVALOV(1),1)
      MMS=IVALOV(1)
      GO TO 10
8     J17=J(17,JOB)
      CALL UNMIX(11,0,J17,MMS)
10    IF(NFB(I)) 15,15,30
15    IF(IWRTA.EQ.0) GO TO 21
      WRITE(6,20) JOB,I
20    FORMAT(' NFMAL CANNOT FILL MEMORY REQUEST FOR JOB',I4,
     C              ' ON PASS NO',I1)
     C
21    IF(NFB(1)) 460,460,22
22    IF(KEY) 460,460,1025
30    DO 31 NC=1,3
      DO 31 MC=1,3
31    NTKOF(NC,MC)=0
      IF(NFB(I)-9) 40,35,35
35    KTEMP1=NFB(I)
      KNO=IFIX(FLOAT(KTEMP1)/2.7183+.99999)
      GO TO 80
40    IF(NFB(I)-8) 50,45,45
45    KNO=4
      GO TO 80
50    IF(NFB(I)-5) 60,55,55
55    KNO=3
      GO TO 80
60    IF(NFB(I)-3) 70,65,65
65    KNO=2
```

```
      GO TO 80
70    KNO=1
80    K=1
81    FORMAT(' NFMAL WILL NOW TRY TO FIND A BLOCK FOR JOB',I
     C3,' I=',I3)
      IBST1=0
      IBST2=0
85    IF(MAVL(JJJ,5)-MMS) 120,2000,2000
90    IF(IBST1) 100,95,100
95    IBST1=MAVL(JJJ,5)
      IBST2=JJJ
      DO 96 NC=1,3
      DO 96 MC=1,3
96    ITKOF(NC,MC)=NTKOF(NC,MC)
100   IF(IBST1-MAVL(JJJ,5)) 117,117,105
105   DO 107 NC=1,3
      DO 107 MC=1,3
107   ITKOF(NC,MC)=NTKOF(NC,MC)
      IF(K-KNO) 115,115,110
110   L(I)=JJJ
      GO TO 155
115   IBST1=MAVL(JJJ,5)
      IBST2=JJJ
117   DO 118 NC=1,3
      DO 118 MC=1,3
118   NTKOF(NC,MC)=0
120   JJJ=MAVL(JJJ,1)
      IF(JJJ) 125,125,121
121   IF(I-2) 124,123,122
122   IF(JJJ.EQ.L(2)) GO TO 120
123   IF(JJJ.EQ.L(1)) GO TO 120
124   K=K+1
      GO TO 85
125   IF(IBST1) 130,15,130
130   L(I)=IBST2
      JJJ=IBST2
      GO TO 155
2000  CONTINUE
2001  FORMAT(' NFMAL,2000,START,LENGTH,JOB,I',4(3X,I6))
2002  IF(I-2) 90,3001,2005
2005  IF(MAVL(JJJ,3)-MAVL(L(1),3)) 2010,2020,2020
2010  IF(MAVL(L(1),3)-MAVL(L(2),3)) 2030,2015,2015
2015  IF(MAVL(JJJ,3)-MAVL(L(2),3))2035,2040,2040
2020  IF(MAVL(JJJ,3)-MAVL(L(2),3)) 2045,2025,2025
2025  IF(MAVL(L(1),3)-MAVL(L(2),3)) 2050,2055,2055
2030  IRX=L(2)
```

```
      IRXT=2
      IMX=L(1)
      IMXT=1
      GO TO 2037
2035  IRX=L(1)
      IRXT=1
      IMX=L(2)
      IMXT=2
2037  ILX=JJJ
      ILXT=3
      GO TO 2060
2040  IRX=L(1)
      IRXT=1
      ILX=L(2)
      ILXT=2
      GO TO 2047
2045  IRX=L(2)
      IRXT=2
      ILX=L(1)
      ILXT=1
2047  IMX=JJJ
      IMXT=3
      GO TO 2060
2050  ILX=L(1)
      ILXT=1
      IMX=L(2)
      IMXT=2
      GO TO 2057
2055  ILX=L(2)
      ILXT=2
      IMX=L(1)
      IMXT=1
2057  IRX=JJJ
      IRXT=3
      GO TO 2060
3001  IF(MAVL(JJJ,3)-MAVL(L(1),3))3005,3010,3010
3005  IRX=L(1)
      IRXT=1
      IMX=JJJ
      IMXT=2
      GO TO 2060
3010  IRX=JJJ
      IRXT=2
      IMX=L(1)
      IMXT=1
2060  IRREF=MAVL(IRX,3)+MAVL(IRX,5)
```

```
2065  IMREF=MAVL(IMX,3)+MAVL(IMX,5)
      IF(IMREF/MODSIZ-(IRREF-MMS)/MODSIZ) 2075,2070,2070
2070  IMREF=((IRREF-MMS)/MODSIZ)*MODSIZ
2075  IF(MAVL(IMX,3)-IMREF+MMS) 2077,2077,117
2077  IF(IMX-IRX) 2080,2087,2080
2080  IF(IMREF-(MAVL(IMX,3)+MAVL(IMX,5))) 2085,2090,2090
2085  NTKOF(IMXT,1)=1
      NTKOF(IMXT,2)=IMREF
      NTKOF(IMXT,3)=MAVL(IMX,3)+MAVL(IMX,5)-IMREF
      GO TO 2090
2087  IF(IMREF-IRREF+MMS) 2088,2090,2090
2088  NTKOF(IMXT,1)=1
      NTKOF(IMXT,2)=IMREF
      NTKOF(IMXT,3)=IRREF-MMS-IMREF
2090  IF(I-2) 2140,2140,2095
2095  ILREF=MAVL(ILX,3)+MAVL(ILX,5)
      IF(ILREF/MODSIZ-(IMREF-MMS)/MODSIZ) 2105,2100,2100
2100  ILREF=((IMREF-MMS)/MODSIZ)*MODSIZ
2105  IF(MAVL(ILX,3)-ILREF+MMS) 2110,2110,117
2110  IF(ILX-IMX) 2115,2125,2115
2115  IF(ILREF-(MAVL(ILX,3)+MAVL(ILX,5))) 2120,2140,2140
2120  NTKOF(ILXT,3)=MAVL(ILX,3)+MAVL(ILX,5)-ILREF
      GO TO 2135
2125  IF(ILREF-IMREF+MMS) 2130,2140,2140
2130  NTKOF(ILXT,3)=IMREF-MMS-ILREF
2135  NTKOF(ILXT,1)=1
      NTKOF(ILXT,2)=ILREF
2140  GO TO 90
155   J27=J(27,JOB)
      CALL UNMIX(11,10,J27,IND)
      IF(IND) 220,220,160
160   IF(I-3) 162,208,208
162   MODNM(I)=(MAVL(L(I),3)+MAVL(L(I),5)-MMS-ITKOF(I,3))/MO
     CDSIZ
      IF(MODNM(I)) 192,192,164
164   IF(MODNM(I)*MODSIZ-MMS-MAVL(L(I),3)) 192,168,168
168   IF(I-1) 169,169,1681
1681  IF(MAVL(L(I),3)-MAVL(L(I-1),3)) 169,1682,1682
1682  IF((MAVL(L(I-1),3)+MMS)/MODSIZ-(MODNM(I)*MODSIZ-MMS)/M
     CODSIZ)      1683,192,192
     C
1683  IF((MAVL(L(I-1),3)+MAVL(L(I-1),5))/MODSIZ-MODNM(I)-(MM
     CS/MODSIZ    +1)) 169,169,1684
     C
1684  ITKOF(I-1,1)=1
      ITKOF(I-1,2)=(MODNM(I)-(MMS/MODSIZ+1))*MODSIZ
```

```
      ITKOF(I-1,3)=MAVL(L(I-1),3)+MAVL(L(I-1),5)-ITKOF(I-1,2
     C)
169   L(I+1)=L(I)
      IF(MAVL(L(I),3)+MAVL(L(I),5)-MMS-ITKOF(I,3)-MODNM(I)*M
     CODSIZ)       172,172,170
     C
170   ITKOF(I+1,1)=1
      ITKOF(I+1,2)=MODNM(I)*MODSIZ
      ITKOF(I+1,3)=MAVL(L(I),3)+MAVL(L(I),5)-MMS-ITKOF(I,3)
     C              -MODNM(I)*MODSIZ
     C
172   I=I+1
      IF(I-3) 178,208,208
178   MODNM(I)=(MODNM(I-1)*MODSIZ-MMS)/MODSIZ
      IF(MODNM(I)) 192,192,180
180   IF(MODNM(I)*MODSIZ-MMS-MAVL(L(I),3)) 192,182,182
182   L(I+1)=L(I)
      IF(MODNM(I-1)*MODSIZ-MMS-MODNM(I)*MODSIZ) 208,208,190
190   ITKOF(I+1,1)=1
      ITKOF(I+1,2)=MODNM(I)*MODSIZ
      ITKOF(I+1,3)=MODNM(I-1)*MODSIZ-MMS-MODNM(I)*MODSIZ
      GO TO 208
192   I=I+1
      NFB(I)=NFB(1)-(I-1)
195   JJJ=MAVLS
      GO TO 197
196   JJJ=MAVL(JJJ,1)
197   IF(JJJ.LE.0) GO TO 15
      IF(I-2) 198,199,198
198   IF(JJJ.EQ.L(2)) GO TO 196
199   IF(JJJ.EQ.L(1)) GO TO 196
      GO TO 10
208   I=1
      KKX=0
210   IF(ITKOF(I,1)) 212,220,212
212   KKX=KKX+1
      IF(KKX-128) 214,214,213
213   WRITE(6,1510)
1510  FORMAT(' NO MORE EMPTIES IN MAVL')
      GO TO 460
214   IF(MAVL(KKX,5)) 212,216,212
216   IF(MAVL(L(I),1)+1) 218,219,218
218   MAVL(MAVL(L(I),1),2)=KKX
219   MAVL(KKX,1)=MAVL(L(I),1)
      MAVL(L(I),1)=KKX
      MAVL(KKX,2)=L(I)
```

```
      MAVL(KKX,3)=ITKOF(I,2)
      MAVL(KKX,4)=MAVL(L(I),4)
      MAVL(KKX,5)=ITKOF(I,3)
      MAVL(L(I),5)=MAVL(L(I),5)-ITKOF(I,3)
      NFB(1)=NFB(1)+1
220   IF(I.NE.1) GO TO 221
      NJWM=NJWM-1
      IPASS(11)=IPASS(11)-1
221   INDXB=L(I)
      INDXB=L(I)
      WRITE(6,1500) JOB,MMS,INDXB,MAVL(INDXB,3),MAVL(INDXB,5
     C),I
1500  FORMAT(' NFMAL TO MASGN',6(3X,I5))
440   CALL MASGN(1,JOB,INDXB,MMS)
445   IF(L(3)) 450,455,450
450   I=I+1
      IF(I-3) 210,210,455
455   J27=J(27,JOB)
      CALL PACK(0,J27,0,16)
      J(27,JOB)=J27
      IF(KEY) 1025,460,1025
460   WRITE(6,1550) ICLK,NJWM,NFB(1),IPASS(10)
1550  FORMAT(' OUT NFMAL-TIME,NJWM,NFB &AMT. FREE ARE',4(1X,
     CI4))
      IF(IWRTA.EQ.0) GO TO 495
      WRITE(6,465)
465   FORMAT(' MAVL FOLLOWS',/)
466   FORMAT(6(3X,I5))
467   FORMAT(' MALC FOLLOWS',/)
      KKK=MAVLS
470   IF(KKK) 480,480,475
475   WRITE(6,466) KKK,(MAVL(KKK,JX),JX=1,5)
      KKK=MAVL(KKK,1)
      GO TO 470
480   WRITE(6,467)
      KKK=MALCS
485   IF(KKK) 495,495,490
490   WRITE(6,466) KKK,(MALC(KKK,JX),JX=1,5)
      KKK=MALC(KKK,1)
      GO TO 485
495   RETURN
      END
```

```
      SUBROUTINE MAPREF
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      INTEGER*2 ICLK
      COMMON/BLK12/ICECSZ,ICON,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,     IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,        NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,        MMAPF,MMSPF
     C
      COMMON/BLK13/RP01,RP02,RP03,RP04,RP05,RP06,RP07,RP08,R
     CP09,        RP10,RP11
     C
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),      IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),     MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),        NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,            NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,          ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,     NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL
      DIMENSION L(3)
C   TO SET UP ISAVE LIST OF JOBS IN ORDER OF PREFERENCE
C   JOBS TO BE ORDERED PASSED IN ARRAY NAML(K)
C   PROGRAM FINDS JOBS TO ALLOCATE
      DIMENSION ITKOF(3,3),NTKOF(3,3),KPTMR(3,2)
      INTEGER*2 MODSIZ,IRREF,IMREF,ILREF,IRX,IRXT,IMX,IMXT,I
     CLX,ILXT,    ITKOF,NTKOF,KPTMR,MODST,MODED,L
     C
      CALL UNMIX(1,0,IS,IWRTA)
      WRITE(6,1580) ICLK,NJWM,NFB(1),IPASS(10)
1580  FORMAT(' IN MAPREF-TIME,NJWM,NFB,&AMT.FREE ARE',4(1X,I
     C4))
      MODSIZ=MTOTAL/NMODS
      IX=0
      DO 10 KX=1,ICECSZ
      NAML(KX)=0
      JNAM=ICEC(2,KX)
      IF(JNAM) 3,10,3
3     J27=J(27,JNAM)
      CALL UNMIX(5,4,J27,IND)
      IF(IND) 5,10,5
```

```
5       IX=IX+1
        NAML(IX)=JNAM
10      CONTINUE
        NJWM=IX
        NJPW=IX
        IF(NJPW) 12,12,15
12      WRITE(6,13)
13      FORMAT(' NO JOBS WAITING MEMORY AT THIS CALL TO MAPREF
       C')
        RETURN
15      DO 20 KX=1,128
        NBLK(KX)=0
20      CONTINUE
        KX=MAVLS
        IF(KX) 25,25,35
25      WRITE(6,30)
30      FORMAT(' NO AVAILABLE BLOCKS IN MAVL')
        RETURN
35      IX=0
40      IX=IX+1
55      NBLK(IX)=KX
        KX=MAVL(KX,1)
        IF(KX) 60,60,40
60      NFB(1)=IX
        NBPF=IX
65      DO 70 KX=1,3
        DO 68 IX=1,2
        ISAVE(KX,IX)=0
        ITKOF(KX,IX)=0
        KPTMR(KX,IX)=0
68      CONTINUE
        ISAVE(KX,3)=0
        ITKOF(KX,3)=0
70      CONTINUE
        K=0
80      K=K+1
        IF(K-NJPW) 85,85,180
85      IF(NAML(K)) 80,80,90
90      I=0
75      I=I+1
        J17=J(17,NAML(K))
        CALL UNMIX(11,0,J17,MMS)
        M=0
95      DO 97 KX=1,3
        DO 97 IX=1,3
97      NTKOF(KX,IX)=0
```

C-5

```
98      M=M+1
        IF(M-NBPF) 105,105,100
100     IF(I-1) 80,80,180
105     IF(NBLK(M)) 98,98,110
C  SEE IF JOB WILL FIT INTO PRESENT BLOCK IN NBLK(M)
110     KTEMP1=MAVL(NBLK(M),5)
        IF(KTEMP1.LE.0) GO TO 98
        DSM=FLOAT(MMS)/FLOAT(KTEMP1)
        IF(DSM-1.) 2000,2000,98
2000    JJJ=NBLK(M)
        L(1)=ISAVE(1,2)
        L(2)=ISAVE(2,2)
        L(3)=ISAVE(3,2)
2001    FORMAT(' MAPREF 2001',5(4X,I4))
2002    IF(I-2) 115,3001,2005
2005    IF(MAVL(JJJ,3)-MAVL(L(1),3)) 2010,2020,2020
2010    IF(MAVL(L(1),3)-MAVL(L(2),3)) 2030,2015,2015
2015    IF(MAVL(JJJ,3)-MAVL(L(2),3))2035,2040,2040
2020    IF(MAVL(JJJ,3)-MAVL(L(2),3)) 2045,2025,2025
2025    IF(MAVL(L(1),3)-MAVL(L(2),3)) 2050,2055,2055
2030    IRX=L(2)
        IRXT=2
        IMX=L(1)
        IMXT=1
        GO TO 2037
2035    IRX=L(1)
        IRXT=1
        IMX=L(2)
        IMXT=2
2037    ILX=JJJ
        ILXT=3
        GO TO 2060
2040    IRX=L(1)
        IRXT=1
        ILX=L(2)
        ILXT=2
        GO TO 2047
2045    IRX=L(2)
        IRXT=2
        ILX=L(1)
        ILXT=1
2047    IMX=JJJ
        IMXT=3
        GO TO 2060
2050    ILX=L(1)
        ILXT=1
```

```
      IMX=L(2)
      IMXT=2
      GO TO 2057
2055  ILX=L(2)
      ILXT=2
      IMX=L(1)
      IMXT=1
2057  IRX=JJJ
      IRXT=3
      GO TO 2060
3001  IF(MAVL(JJJ,3)-MAVL(L(1),3))3005,3010,3010
3005  IRX=L(1)
      IRXT=1
      IMX=JJJ
      IMXT=2
      GO TO 2060
3010  IRX=JJJ
      IRXT=2
      IMX=L(1)
      IMXT=1
      GO TO 2060
2060  IRREF=MAVL(IRX,3)+MAVL(IRX,5)
2065  IMREF=MAVL(IMX,3)+MAVL(IMX,5)
      IF(IMREF/MODSIZ-(IRREF-MMS)/MODSIZ) 2075,2070,2070
2070  IMREF=((IRREF-MMS)/MODSIZ)*MODSIZ
2075  IF(MAVL(IMX,3)-IMREF+MMS) 2077,2077,95
2077  IF(IMX-IRX) 2080,2087,2080
2080  IF(IMREF-(MAVL(IMX,3)+MAVL(IMX,5))) 2085,2090,2090
2085  NTKOF(IMXT,1)=1
      NTKOF(IMXT,2)=IMREF
      NTKOF(IMXT,3)=MAVL(IMX,3)+MAVL(IMX,5)-IMREF
      GO TO 2090
2087  IF(IMREF-IRREF+MMS) 2088,2090,2090
2088  NTKOF(IMXT,1)=1
      NTKOF(IMXT,2)=IMREF
      NTKOF(IMXT,3)=IRREF-MMS-IMREF
2090  IF(I-2) 2140,2140,2095
2095  ILREF=MAVL(ILX,3)+MAVL(ILX,5)
      IF(ILREF/MODSIZ-(IMREF-MMS)/MODSIZ) 2105,2100,2100
2100  ILREF=((IMREF-MMS)/MODSIZ)*MODSIZ
2105  IF(MAVL(ILX,3)-ILREF+MMS) 2110,2110,95
2110  IF(ILX-IMX) 2115,2125,2115
2115  IF(ILREF-(MAVL(ILX,3)+MAVL(ILX,5))) 2120,2140,2140
2120  NTKOF(ILXT,3)=MAVL(ILX,3)+MAVL(ILX,5)-ILREF
      GO TO 2135
2125  IF(ILREF-IMREF+MMS) 2130,2140,2140
```

```
2130  NTKOF(ILXT,3)=IMREF-MMS-ILREF
2135  NTKOF(ILXT,1)=1
      NTKOF(ILXT,2)=ILREF
2140  CONTINUE
      GO TO 115
C  JOB WILL FIT INTO MEMORY BLOCK
115   MODST=(MAVL(NBLK(M),3)+MAVL(NBLK(M),5)-MMS-NTKOF(I,3))
     C/MODSIZ+1
      MODED=(MAVL(NBLK(M),3)+MAVL(NBLK(M),5)-NTKOF(I,3)-1)/M
     CODSIZ+1
      J11=J(11,NAML(K))
      CALL UNMIX(10,0,J11,MNAR)
      MUD=0
      IF(MODED-MODST-1) 130,125,120
120   IST=MODST+1
      IED=MODED-1
      DO 122 LX=IST,IED
122   MUD=MUD+NAA(LX)
125   ILMS=MAVL(NBLK(M),3)+MAVL(NBLK(M),5)-(MODED-1)*MODSIZ
      MUD=MUD+(NAA(MODED)*ILMS)/MAS(MODED)
130   IF(MAVL(NBLK(M),3)+MAVL(NBLK(M),5)-MODST*MODSIZ)
     C              131,131,132
     C
131   IFMS=MAVL(NBLK(M),5)
      GO TO 133
132   IFMS=MODST*MODSIZ-MAVL(NBLK(M),3)
133   MUD=MUD+(NAA(MODST)*IFMS)/MAS(MODST)
135   TEMP=FLOAT(MMSPF)*DSM
      DUM=FLOAT(MUD)/FLOAT(MNAR)
      IF(DUM-1.) 140,140,145
140   TEMP2=FLOAT(MMAPF)*DUM
      GO TO 150
145   TEMP2=FLOAT(MMAPF)/DUM
150   ITEMP=IFIX(TEMP+TEMP2+.5)
      IF(ISAVE(I,1)) 165,165,162
162   J1 =J(10,NAML(K))
      CALL UNMIX(15,7,J10,KTEMP1)
      J10=J(10,ISAVE(I,1))
      CALL UNMIX(15,7,J10,KTEMP2)
      IF(ITEMP+IFIX((KTEMP1-KTEMP2)*RP02)-ISAVE(I,3)) 95,95,
     C165
165   KPTMR(I,1)=K
      ISAVE(I,1)=NAML(K)
168   KPTMR(I,2)=M
      ISAVE(I,2)=NBLK(M)
      ISAVE(I,3)=ITEMP
```

```
      DO 172 KX=1,3
170   ITKOF(KX,1)=NTKOF(KX,1)
      ITKOF(KX,2)=NTKOF(KX,2)
      ITKOF(KX,3)=NTKOF(KX,3)
172    CONTINUE
      GO TO 95
180   IF(ISAVE(I,2)) 185,210,185
185   J17=J(17,ISAVE(1,1))
      CALL UNMIX(11,0,J17,MMS)
      J27=J(27,ISAVE(1,1))
      CALL UNMIX(11,10,J27,IND)
      IF(IND) 240,190,240
190   IF(MAVL(ISAVE(I,2),5)-MMS) 195,195,200
195   IF(KPTMR(I,2)-NBPF) 196,197,197
196   NBLK(KPTMR(I,2))=NBLK(NBPF)
197   NBLK(NBPF)=0
      NBPF=NBPF-1
200   IF(KPTMR(I,1)-NJPW) 201,202,202
201   NAML(KPTMR(I,1))=NAML(NJPW)
202   NAML(NJPW)=0
      NJPW=NJPW-1
      JINDX=ISAVE(1,1)
      MINDX=ISAVE(1,2)
      IF(IWRTA.EQ.0) GO TO 207
      WRITE(6,205) JINDX,MMS,MINDX,MAVL(MINDX,3),MAVL(MINDX,
     C5)
205    FORMAT(' MAPREF 205 CALLS MASGN',5(4X,I4))
207   CALL MASGN(1,JINDX,MINDX,MMS)
      J(23,JINDX)=ISAVE(1,3)
      NJWM=NJWM-1
      GO TO 65
1500   FORMAT(' TIME=',I6,' MAPREF FINDS NO BLK FOR JOB',I4,'
     C PASS=',I3)
210   IF(ISAVE(1,1)) 400,400,211
211   IF(I-1) 400,400,212
212   WRITE(6,1500) ICLK,NAML(K),I
215   IF(KPTMR(I,1)-NJPW) 216,217,217
216   NAML(KPTMR(1,1))=NAML(NJPW)
217   NAML(NJPW)=0
      NJPW=NJPW-1
      DO 220 KX=1,3
      IF(KPTMR(KX,2))218,220,218
218   NBLK(KPTMR(KX,2))=ISAVE(KX,2)
220   CONTINUE
      GO TO 65
240   L(I)=ISAVE(I,2)
```

```
      MODNM(I)=(MAVL(L(I),3)+MAVL(L(I),5)-MMS-ITKOF(I,3))/MO
     CDSIZ
      IF(I-2) 260,242,300
242   L(1)=ISAVE(1,2)
245   IF(MAVL(L(2),3)-MAVL(L(1),3))260,250,250
250   IF((MAVL(L(1),3)+MMS)/MODSIZ-(MODNM(I)*MODSIZ-MMS)/MOD
     CSIZ)          251,255,255
     C
251   IF(MAVL(L(I),3)-MODNM(I)*MODSIZ+MMS) 252,252,255
252   IF((MAVL(L(1),3)+MAVL(L(1),5))/MODSIZ-(MODNM(I)*MODSIZ
     C-MMS)/MOD    SIZ) 260,253,253
     C
253   IF(MAVL(L(1),3)+MAVL(L(1),5) -((MODNM(I)*MODSIZ-MMS)/M
     CODSIZ)        *MODSIZ) 265,265,254
     C
254   ITKOF(1,1)=1
      ITKOF(1,2)=((MODNM(I)*MODSIZ-MMS)/MODSIZ)*MODSIZ
      ITKOF(1,3)=MAVL(L(1),3)+MAVL(L(1),5)-ITKOF(1,2)
      GO TO 265
255   DO 258 KX=1,3
      IF(ISAVE(KX,2)) 256,258,256
256   NBLK(KPTMR(KX,2))=0
258   CONTINUE
      K=KPTMR(1,1)
      GO TO 75
260   IF(MAVL(L(I),3)-MODNM(I)*MODSIZ+MMS) 265,265,255
265   IF(MODNM(I)*MODSIZ-MAVL(L(I),3)-MAVL(L(I),5)+MMS+ITKOF
     C(I,3))        266,270,270
     C
266   ITKOF(I+1,1)=1
      ITKOF(I+1,2)=MODNM(I)*MODSIZ
      ITKOF(I+1,3)=MAVL(L(I),3)+MAVL(L(I),5)-MMS-ITKOF(I,3)-
     CMODNM(I)*     MODSIZ
     C
270   ISAVE(I+1,1)=ISAVE(I,1)
      ISAVE(I+1,2)=ISAVE(I,2)
      ISAVE(I+2,3)=ISAVE(I,3)
      KPTMR(I+1,1)=KPTMR(I,1)
      KPTMR(I+1,2)=KPTMR(I,2)
      I=I+1
      IF(I-3) 275,300,300
275   MODNM(I)=(MODNM(I-1)*MODSIZ-MMS)/MODSIZ
      IF(MAVL(ISAVE(I,2),3)-MODNM(I)*MODSIZ+MMS) 285,285,255
285   IF(MODNM(I)*MODSIZ-MODNM(I-1)*MODSIZ+MMS) 290,295,295
290   ITKOF(I+1,1)=1
      ITKOF(I+1,2)=MODNM(I)*MODSIZ
```

```
      ITKOF(I+1,3)=MODNM(I-1)*MODSIZ-MMS-MODNM(I)*MODSIZ
295   ISAVE(I+1,1)=ISAVE(I,1)
      ISAVE(I+1,2)=ISAVE(I,2)
      ISAVE(I+1,3)=ISAVE(I,3)
      KPTMR(I+1,1)=KPTMR(I,1)
      KPTMR(I+1,2)=KPTMR(I,2)
300   DO 310 KX=1,3
      IF(KPTMR(KX,2)) 305,310,305
305   NBLK(KPTMR(KX,2))=ISAVE(KX,2)
310   CONTINUE
      I=3
320   I=I+1
      IF(I-3) 325,325,322
322   IF(KPTMR(1,1)-NJPW) 323,324,324
323   NAML(KPTMR(1,1))=NAML(NJPW)
324   NAML(NJPW)=0
      NJPW=NJPW-1
      GO TO 65
325   KKX=0
      IF(ITKOF(I,1)) 330,370,330
330   KKX=KKX+1
      IF(KKX-128) 334,334,332
332   WRITE(6,333)
333   FORMAT(' NO MORE EMPTIES IN MAVL')
      RETURN
334   IF(MAVL(KKX,5)) 330,335,330
335   MAVL(KKX,1)=MAVL(ISAVE(I,2),1)
      IF(MAVL(KKX,1)) 345,345,340
340   MAVL(MAVL(KKX,1),2)=KKX
345   MAVL(ISAVE(I,2),1)=KKX
      MAVL(KKX,2)=ISAVE(I,2)
      MAVL(KKX,3)=ITKOF(I,2)
      MAVL(KKX,4)=MAVL(ISAVE(I,2),4)
      MAVL(KKX,5)=ITKOF(I,3)
      MAVL(ISAVE(I,2),5)=MAVL(ISAVE(I,2),5)-ITKOF(I,3)
      NFB(1)=NFB(1)+1
350   IX=NBPF+1
      IF(IX-128) 360,360,355
355   WRITE(6,357)
357   FORMAT(' NO MORE EMPTIES IN NBLK')
      RETURN
360   NBLK(IX)=KKX
      NBPF=NBPF+1
370   IF(MAVL(ISAVE(I,2),5)-MMS) 375,375,380
375   IF(KPTMR(I,2)-NBPF) 376,378,378
376   NBLK(KPTMR(I,2))=NBLK(NBPF)
```

```
      DO 377 KKK=1,3
377   IF(NBPF.EQ.KPTMR(KKK,2)) KPTMR(KKK,2)=KPTMR(1,2)
378   NBLK(NBPF)=0
      NBPF=NBPF-1
380   IF(I-1) 390,385,390
385   J(23,ISAVE(1,1))=ISAVE(1,3)
      NJWM=NJWM-1
390   JINDX=ISAVE(I,1)
      MINDX=ISAVE(I,2)
      IF(IWRTA.EQ.0) GO TO 397
      WRITE(6,395) JINDX,MMS,MINDX,MAVL(MINDX,3),MAVL(MINDX,
     C5),I
395   FORMAT(' MAPREF 395 CALLS MASGN',6(3X,I4))
397   CALL MASGN(1,JINDX,MINDX,MMS)
      GO TO 320
400   WRITE(6,405) ICLK,NJWM,NFB(1),IPASS(10)
405   FORMAT(' OUT MAPREF-TIME,NJWM,NFB &AMT. FREE ARE',4(1X
     C,I4))
      IF(IWRTA.EQ.0) GO TO 432
      WRITE(6,410)
410   FORMAT(' MAVL FOLLOWS',//)
415   FORMAT(6(3X,I5))
      KKK=MAVLS
418   IF(KKK) 422,422,420
420   WRITE(6,415) KKK,(MAVL(KKK,JX),JX=1,5)
      KKK=MAVL(KKK,1)
      GO TO 418
422   WRITE(6,425)
425   FORMAT(' MALC FOLLOWS',//)
      KKK=MALCS
427   IF(KKK) 432,432,430
430   WRITE(6,415) KKK,(MALC(KKK,JX),JX=1,5)
      KKK=MALC(KKK,1)
      GO TO 427
432   RETURN
      END
```

```
      SUBROUTINE MASGN(KEY,INDXJ,INDXB,ISPACE)
C  MASGN ASSIGNS MEMORY TO JOBS AFTER MEMORY PREFERENCE FACT
CORS HAVE
C  BEEN CALCULATED
C  IF KEY=1 THEN ASSIGN THE AMOUNT ISPACE TO JOB INDXJ FROM
CBLOCK
C  INDXB
C  IF KEY =0 THEN ASSIGN ALL JOBS IN ISAVE MEMORY FROM ASSOC
CIATED
C  BLOCKS
      COMMON/BLK7/J(27,64)
      COMMON/BLK8/ ICLK
      INTEGER*2 ICLK
      COMMON/BLK12/ICECSZ,ICON,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,     IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,         NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,         MMAPF,MMSPF
     C
      COMMON/BLK13/ RANAI
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),         NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,              NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,           ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL
      CALL UNMIX(1,0,IS,IWRTA)
      MODSIZ=MTOTAL/NMODS
      IF(KEY-1) 5,2,5
2      JOB=INDXJ
      IBLKN=INDXB
      NMS=ISPACE
      KEMPT=1
      GO TO 40
5     I=1
      KEMPT=1
1     IF(ISAVE(I,1)) 25,15,25
15    WRITE(6,20)
20    FORMAT(31H ALL ASSIGNMENTS HAVE BEEN MADE)
      GO TO 300
```

```
25      JOB=ISAVE(I,1)
        IBLKN=ISAVE(I,2)
        J22=J(22,JOB)
        CALL UNMIX(7,6,J22,IND7)
        IF(IND7) 30,35,30
30      IVALOV(1)=0
        CALL OVFLMG(1,JOB,7,IVALOV,1)
        MMS=IVALOV(1)
        GO TO 40
35      J17=J(17,JOB)
        CALL UNMIX(11,0,J17,MMS)
40      IF(KEMPT-255) 55,55,45
45      WRITE(6,50)
50      FORMAT(29H NO MORE EMPTY BLOCKS IN MALC)
        GO TO 300
55      IF(MALC(KEMPT,5)) 60,65,60
60      KEMPT=KEMPT+1
        GO TO 40
65      KOCC1=0
        KOCC=MALCS
70      IF(KOCC) 75,75,100
C   TO HERE IF NEW BLOCK GOES AT END OF MALC CHAIN
75      IF(KOCC1) 85,85,80
C   TO HERE IF MALC CHAIN WAS NOT EMPTY
80      MALC(KOCC1,1)=KEMPT
        MALC(KEMPT,2)=KOCC1
        GO TO 90
C   TO HERE IF MALC CHAIN WAS EMPTY
85      MALCS=KEMPT
        MALC(KEMPT,2)=-1
90      MALC(KEMPT,1)=-1
        GO TO 135
100     IF(MALC(KOCC,3)-MAVL(IBLKN,3)) 105,105,110
105     KOCC1=KOCC
        KOCC=MALC(KOCC,1)
        GO TO 70
110     IF(KOCC1) 120,115,120
C   ALLOCATED BLOCK WITH INDEX KOCC IS PRESENTLY AT HEAD OF M
CALC CHAIN
115     MALC(KEMPT,2)=-1
        MALCS=KEMPT
        GO TO 125
C   NEW BLOCK GOES BETWEEN BLOCKS WITH INDEXES KOCC1 AND KOCC
120     MALC(KEMPT,2)=KOCC1
        MALC(KOCC1,1)=KEMPT
125     MALC(KEMPT,1)=KOCC
```

```
      MALC(KOCC,2)=KEMPT
135   IF(MAVL(IBLKN,5)-MMS) 140,140,210
C  TO HERE IF ALL OF BLOCK WITH INDEX IBLKN IS TO BE ALLOCAT
CED
140   IF(MAVL(IBLKN,2)) 145,145,153
145   MAVLS=MAVL(IBLKN,1)
      GO TO 155
153   MAVL(MAVL(IBLKN,2),1)=MAVL(IBLKN,1)
155   IF(MAVL(IBLKN,1)+1) 160,165,160
160   MAVL(MAVL(IBLKN,1),2)=MAVL(IBLKN,2)
165   MALC(KEMPT,3)=MAVL(IBLKN,3)
      MALC(KEMPT,4)=MAVL(IBLKN,1)
      MALC(KEMPT,5)=MAVL(IBLKN,5)
      IPASS(10)=IPASS(10)-MAVL(IBLKN,5)
      KO=KOCC1
170   IF(KO) 200,200,175
175   IF(MALC(KO,4)-IBLKN) 200,180,200
180   MALC(KO,4)=MAVL(IBLKN,1)
      KO=MALC(KO,2)
      GO TO 170
200   NFB(1)=NFB(1)-1
      DO 205 IX=1,5
      MAVL(IBLKN,IX)=0
205   CONTINUE
      GO TO 310
210   MALC(KEMPT,3)=MAVL(IBLKN,3)+MAVL(IBLKN,5)-MMS
      MAVL(IBLKN,5)=MAVL(IBLKN,5)-MMS
      MALC(KEMPT,4)=MAVL(IBLKN,1)
      MALC(KEMPT,5)=MMS
      MAVL(IBLKN,4)=KEMPT
      NAB=NAB+1
      IPASS(10)=IPASS(10)-MMS
      GO TO 310
215   IVALOV(1)=KEMPT
      J1=J(1,JOB)
      CALL UNMIX(15,7,J1,IBK)
      IF(IBK) 225,220,225
220   CALL PACK(7,J1,KEMPT,255)
      J(1,JOB)=J1
      GO TO 270
225   J19=J(19,JOB)
      CALL UNMIX(8,0,J19,IBK)
      IF(IBK) 235,230,235
230   CALL PACK(0,J19,KEMPT,255)
      J(19,JOB)=J19
      GO TO 270
```

```
235     J17=J(17,JOB)
        CALL UNMIX(15,11,J17,IBKU)
        CALL UNMIX(12,8,J19,IBKL)
        IBK=IBKL
        CALL PACK(4,IBK,IBKU,15)
        IF(IBK) 245,240,245
240     CALL PACK(7,J17,KEMPT,240)
        J(17,JOB)=J17
        CALL PACK(8,J19,KEMPT,15)
        J(19,JOB)=J19
        GO TO 270
245     J22=J(22,JOB)
        CALL UNMIX(8,7,J22,IND8)
        IF(IND8) 260,250,260
250     CALL PACK(0,J22,128,128)
        J(22,JOB)=J22
260     CALL OVFLMG(2,JOB,8,IVALOV,1)
270     J27=J(27,JOB)
        CALL PACK(0,J27,0,16)
        J(27,JOB)=J27
        IF(IWRTA.EQ.0) GO TO 280
        WRITE(6,1500) ICLK,JOB,KEMPT,MALC(KEMPT,3),MALC(KEMPT,
       C5)
1500    FORMAT(' MASGN,TIME,JOB,MALCBLK,ST,LG,',5(2X,I5))
280     IF(KEY) 285,285,300
285     I=I+1
        GO TO 10
300     RETURN
315     MODST=(MALC(KEMPT,3)+MALC(KEMPT,5)-MMS)/MODSIZ+1
        MODED=(MALC(KEMPT,3)+MALC(KEMPT,5)-1)/MODSIZ+1
        J11=J(11,JOB)
        CALL UNMIX(10,0,J11,MNAR)
         IF(MODED-MODST-1) 330,325,315
315     IST=MODST+1
        IED=MODED-1
        KKX=IST
        DO 320 KKX=IST,IED
        NA(KKX)=NA(KKX)+1
        MAS(KKX)=MAS(KKX)-MODSIZ
        IBWCTR(KKX)=1
320     CONTINUE
325     MEDAMT=MALC(KEMPT,3)+MALC(KEMPT,5)-(MODED-1)*MODSIZ
        NA(MODED)=NA(MODED)+1
        MAS(MODED)=MAS(MODED)-MEDAMT
        NAA(MODED)=MAX0(0,NAA(MODED)-(MEDAMT*MNAR)/MMS)
        IBWCTR(MODED)=1
```

```
330   IF(MALC(KEMPT,3)+MALC(KEMPT,5)-MODST*MODSIZ) 340,335,3
     C35
335   MSTAMT=MODST*MODSIZ-MALC(KEMPT,3)
      GO TO 345
340   MSTAMT=MALC(KEMPT,5)
345   NA(MODST)=NA(MODST)+1
      MAS(MODST)=MAS(MODST)-MSTAMT
      NAA(MODST)=MAXO(0,NAA(MODST)-(MSTAMT*MNAR)/MMS)
      IBWCTR(MODST)=1
      GO TO 215
      END
```

```
      SUBROUTINE MEMRLS(JOB,JBLK)
C  MEMRLS RELEASES THE BLOCK NO 'JBLK' FROM JOB 'JOB'.
      COMMON/BLK7/J(27,64)
      COMMON/BLK12/ICECSZ,ICON,IN1,IFECSZ,IPCT,IP1,IP2,IQ1,I
     CQ2,IQ3,      IR,IS,ITA,MINBLK,MMCT,MMST,MTOTAL,NI,NIOS,
     CNMODS,NPCL,NPCS,         NCPUS,IFEDBK,IRN1,IRN2,IRN3,IR
     CN4,IRN5,IRN6,IRN7,IRN8,IRN9,          MMAPF,MMSPF
     C
      COMMON/BLK13/ RANAI
      COMMON/BLK16/IBWCTR(24),ICEC(5,40),ICECS,ICPU(10),IPAS
     CS(20),       IFECS,IFEC(40,4),IPROS(16),ISAVE(3,3),IVAL
     COV(10),MALC(256,5),      MALCS,MAS(24),MAVL(128,5),MAVL
     CS,MODNM(24),MTP1(24),NA(24),          NAA(24),NAB,NAG,NA
     CML(40),NBLK(128),NFB(3),NJWM,NSCHED,               NREQ(4
     C0,24),NTP(24),NTR(24),NUCA(24)
     C
      INTEGER*2 IBWCTR,ICEC,ICECS,ICPU,IPASS,IFECS,IFEC,IPRO
     CS,           ISAVE,IVALOV,MALC,MALCS,MAS,MAVL,MAVLS,MOD
     CNM,MTP1,NA,NAA,NAB,      NAG,NAML,NBLK,NFB,NJWM,NSCHED,
     CNREQ,NTP,NTR,NUCA
      INTEGER*2 J,JOVFL
      CALL UNMIX(1,0,IS,IWRTA)
      MODSIZ=MTOTAL/NMODS
      MODST=MALC(JBLK,3)/MODSIZ+1
      MODED=(MALC(JBLK,3)+MALC(JBLK,5)-1)/MODSIZ+1
      J11=J(11,JOB)
      CALL UNMIX(10,0,J11,MNAR)
      MMS=MALC(JBLK,5)
      IF(MODED-MODST-1) 6,4,2
2     IST=MODST+1
      IED=MODED-1
      DO 3 KKX=IST,IED
      NA(KKX)=NA(KKX)-1
      MAS(KKX)=MODSIZ
      NAA(KKX)=ICON/MMCT
      IBWCTR(KKX)=1
3     CONTINUE
4     NA(MODED)=NA(MODED)-1
      MEDAMT=MALC(JBLK,3)+MALC(JBLK,5)-(MODED-1)*MODSIZ
      MAS(MODED)=MAS(MODED)+MEDAMT
      NAA(MODED)=MIN0(ICON/MMCT,NAA(MODED)+(MNAR*MEDAMT)/MMS
     C)
      IBWCTR(MODED)=1
6     KTEMP=MALC(JBLK,5)
      MSTAMT=MIN0(KTEMP,MODST*MODSIZ-MALC(JBLK,3))
```

```
      MAS(MODST)=MAS(MODST)+MSTAMT
      NA(MODST)=NA(MODST)+1
      NAA(MODST)=MINO(ICON/MMCT,NAA(MODST)+(MNAR*MSTAMT)/MMS
     C)
      IBWCTR(MODST)=1
      M1=0
      M=MAVLS
10    IF(M) 15,15,70
15    IF(MAVLS) 20,20,90
20    MAVL(1,1)=-1
      MAVL(1,2)=-1
      MAVL(1,3)=MALC(JBLK,3)
      MAVL(1,4)=MALC(JBLK,1)
      MAVL(1,5)=MALC(JBLK,5)
      MAVLS=1
      NENT=1
      I2=MALC(JBLK,2)
23    IF(I2) 27,27,25
25    MALC(I2,4)=1
      I2=MALC(I2,2)
      GO TO 23
27    I2=MALC(JBLK,2)
30    IF(I2) 32,32,33
32    MALCS=MALC(JBLK,1)
      GO TO 35
33    MALC(I2,1)=MALC(JBLK,1)
35    I2=MALC(JBLK,1)
      IF(I2) 44,44,40
40    MALC(I2,2)=MALC(JBLK,2)
44    NFB(1)=NFB(1)+1
45    NAB=NAB-1
      IPASS(10)=IPASS(10)+MALC(JBLK,5)
      IF(IWRTA.EQ.0) GO TO 48
      WRITE(6,47) JOB,JBLK,MALC(JBLK,3),MALC(JBLK,5),NENT,MA
     CVL(NENT,3), MAVL(NENT,5)
     C
47    FORMAT(' MEMRLS--JOB,ALCBLK,STRT,LG.,AVLBLK,STRT,LG.',
     C7(1X,I4))
48    DO 50 I=1,5
      MALC(JBLK,I)=0
50    CONTINUE
      GO TO 500
70    IF(MAVL(M,3)-MALC(JBLK,3)) 75,75,80
75    M1=M
      M=MAVL(M,1)
      GO TO 10
```

```
80      IF(MALC(JBLK,3)+MALC(JBLK,5)-MAVL(M,3))90,85,90
85      ICASE=1
        NENT=M
        GO TO 95
90      ICASE=0
95      IF(M.LE.0) GO TO 100
        IF(MAVL(M,2)) 220,220,100
100     IF(M1.LE.0) GO TO 110
        IF(MAVL(M1,3)+MAVL(M1,5)-MALC(JBLK,3)) 110,105,110
105     ICASE=ICASE+2
        NENT=M1
110     IF(ICASE-3) 170,115,170
115     MAVL(M1,1)=MAVL(M,1)
        MAVL(M1,5)=MAVL(M1,5)+MALC(JBLK,5)+MAVL(M,5)
        MAVL(M1,4)=MALC(JBLK,1)
        IF(MAVL(M,1).GT.0) MAVL(MAVL(M,1),2)=M1
        DO 120 JX=1,5
120     MAVL(M,JX)=0
135     NFB(1)=NFB(1)-1
        GO TO 1751
170     IF(ICASE-2)220,175,220
175     MAVL(M1,4)=MALC(JBLK,1)
        MAVL(M1,5)=MAVL(M1,5)+MALC(JBLK,5)
1751    IF(MALC(JBLK,2)) 176,176,177
176     MALCS=MALC(JBLK,1)
        GO TO 178
177     MALC(MALC(JBLK,2),1)=MALC(JBLK,1)
178     IF(MALC(JBLK,1).GT.0) MALC(MALC(JBLK,1),2)=MALC(JBLK,2
      C)
        GO TO 45
220     IF(ICASE) 290,290,225
225     MAVL(M,3)=MALC(JBLK,3)
        MAVL(M,5)=MAVL(M,5)+MALC(JBLK,5)
        IF(MAVL(M,2)) 232,232,227
227     IF(MAVL(MAVL(M,2),4)-JBLK) 232,230,232
230     MAVL(MAVL(M,2),4)=MALC(JBLK,1)
232     GO TO 1751
290     IQ=1
295     IF(MAVL(IQ,5))300,320,300
300     IQ=IQ+1
        IF(IQ-128) 295,295,305
305     WRITE(6,310)JBLK,JOB
310     FORMAT(47H MEMRLS FOUND NO EMPTIES IN MAVL WHEN RELEAS
      CING ,I7,     4HFROM,I7)
      C
        GO TO 500
```

```
320   NENT=IQ
      IF(MALC(JBLK,1).GT.0) MALC(MALC(JBLK,1),2)=MALC(JBLK,2
     C)
      IF(MALC(JBLK,2)) 325,325,330
325   MALCS=MALC(JBLK,1)
      GO TO 335
330   MALC(MALC(JBLK,2),1)=MALC(JBLK,1)
335   IF(M1) 364,364,362
362   MAVL(M1,1)=IQ
      MAVL(IQ,2)=M1
      GO TO 365
364   MAVL(IQ,2)=-1
      MAVLS=IQ
365   IF(M.GT.0) MAVL(M,2)=IQ
      MAVL(IQ,1)=M
      MAVL(IQ,3)=MALC(JBLK,3)
      MAVL(IQ,4)=MALC(JBLK,1)
      MAVL(IQ,5)=MALC(JBLK,5)
      K=MALC(JBLK,2)
380   IF(K) 44,44,385
385   IF(MALC(K,4)-M) 44,390,44
390   MALC(K,4)=IQ
      K=MALC(K,2)
      GO TO 380
500   RETURN
      END
```

```
      SUBROUTINE OVFLMG(KEY,JOB,IFLD,IVAL,NNT)
C  OVFLMG HANDLES OVERFLOW FROM VARIOUS FIELDS OF THE J-TABL
CF
C  KEY = 1 INDICATES RETRIEVE.  THE FIRST IVAL(1) VALUES
C  FOR JOB'S FIELD IFLD ARE SKIPPED AND RETRIEVAL BEGINS
C  AFTER THIS AND CONTINUES FOR THE NEXT NNT VALUES OR UNTIL
C  THE VALUES UNDER IFLD ARE EXAUSTED WHICHEVER COMES FIRST
C  KEY = 2 INDICATES PLACE.  THE FIRST NNT VALUES IN
C  IVAL(1 THRU NNT) ARE PLACED IN JOB'S OVERFLOW AREA
C  UNDER IFLD.
C  KEY = 3 INDICATES REMOVE.  THE FIRST IVAL(1) VALUES
C  UNDER JOB'S IFLD FIELD ARE SKIPPED AND REMOVAL
C  BEGINS AFTER THIS AND CONTINUES FOR THE NEXT NNT
C  VALUES OR UNTIL THE VALUES UNDER IFLD ARE EXAUSTED
C  WHICHEVER COMES FIRST.
      COMMON/BLK7/J(27,64)
      COMMON/BLK9/JOVFL(32,32)
      DIMENSION IVAL(10)
      INTEGER*2 J,HFIX,JOVFL
      GO TO (10,200,500),KEY
C  TO HERE FOR RETRIEVAL OF NNT VALUES
1     NSKIP=IVAL(1)
      IVAL(1)=0
      K=0
      JADD=J(24,JOB)
15    IF(JADD) 70,70,20
2     NPNTB=0
      IPNT=1
25    IFHD=JOVFL(JADD,IPNT)
      CALL UNMIX(15,10,IFHD,LFLD)
      IF(LFLD-IFLD) 30,40,30
3     NPNTB=IPNT
      CALL UNMIX(10,5,IFHD,IPNT)
      IF(IPNT) 35,35,25
35    JADD=JOVFL(JADD,32)
      GO TO 15
4     CALL UNMIX(5,0,IFHD,NENTRY)
      I=0
45    I=I+1
      IF(I-NENTRY) 50,50,30
5     IF(NSKIP) 60,60,55
55    NSKIP=NSKIP-1
      GO TO 45
6     K=K+1
      IF(K-NNT) 65,65,70
```

```
      IVAL(K)=JOVFL(JADD,IPNT+I)
      IF(K.GE.NNT) GO TO 70
      GO TO 45
   70 RETURN
C  TO HERE FOR PLACING THE VALUES FROM IVAL(1 THRU NNT)
  200 K=1
      LASTB=0
      JADD=J(24,JOB)
  205 IF(JADD) 290,290,210
  210 IPNT=1
  212 IFHD=JOVFL(JADD,IPNT)
      CALL UNMIX(15,10,IFHD,LFLD)
      CALL UNMIX(10,5,IFHD,NPNTF)
      CALL UNMIX(5,6,IFHD,NENTRY)
      IF(LFLD-IFLD) 215,350,215
  215 IF(NPNTF) 225,225,220
  220 NPNTB=IPNT
      IPNT=NPNTF
      GO TO 212
  225 NPNTF=32
      IF(IPNT+NENTRY+6-31) 235,230,230
  230 LASTB=JADD
      JADD=JOVFL(JADD,32)
      GO TO 205
  235 NPNTB=IPNT
      IPNT=IPNT+NENTRY+3
      IFULL=JOVFL(JADD,NPNTB)
      CALL PACK(5,IFULL,IPNT,31)
      JOVFL(JADD,NPNTB)=IFULL
  240 JOVFL(JADD,IPNT)=IFLD*1.24
  245 I=0
  250 I=I+1
      IF(IPNT+I-NPNTF) 260,255,255
  255 IF(NPNTF-32) 220,230,220
  260 JOVFL(JADD,IPNT+I)=IVAL(K)
      JOVFL(JADD,IPNT)=JOVFL(JADD,IPNT)+1
      K=K+1
      IF(K-NNT) 220,220,290
  290 JADD=0
  295 JADD=JADD+1
      IF(JADD-32) 310,310,300
  300 WRITE(6,305) K,NNT,IFLD,JOB
  305 FORMAT(' JOVFLW RAN OUT OF ROOM WHEN TRYING TO PLACE',
     1/,          ' VALUE NO.',I4,' FROM A TOTAL OF',I4,' VA
     2LUES',/,                ' INTO FIELD',I4,' FOR JOB',I4
     3)
```

```
      RETURN
315   IF(JOVFL(JADD,1)) 295,315,295
315   IF(LASTB) 320,320,317
317   JOVFL(LASTB,32)=JADD
      GO TO 325
32    J(24,JOB)=JADD
      IFULL=J(22,JOB)
      CALL PACK(0,IFULL,IFLD,IFLD)
      J(22,JOB)=IFULL
325   IPNT=1
      NPNTF=32
      GO TO 240
C  FOUND A FIELD WITH NAME IFLD
35.   IF(NPNTF) 360,360,355
355   IF(IPNT+NENTRY+1-NPNTF) 245,220,220
36.   IF(IPNT+NENTRY-31) 230,365,365
365   NPNTF=32
      GO TO 245
39.   IFULL=J(22,JOB)
      CALL PACK(1,IFULL,IFLD,IFLD)
      J(22,JOB)=IFULL
      RETURN
C  TO HERE FOR REMOVAL OF THE FIRST NNT ENTRIES AFTER IVAL(1
C)
5.    LASTB=0
      JADD=J(24,JOB)
      K=1
      ISKIP=IVAL(1)
505   IF(JADD) 510,510,520
51.   KK=K-1
      WRITE(6,515) KK,NNT,IFLD,JOB
515   FORMAT(' OVFLMG CAN FIND ONLY',I4,' OF THE',I4,/,
     C       ' VALUES FROM FIELD',I4,' OF JOB',I4)
     C
      GO TO 595
52.   NPNTB=0
      IPNT=1
525   IFWD=JOVFL(JADD,IPNT)
      CALL UTMIX(15,1,IFWD,LFLD)
      IF(LFLD-IFLD) 530,540,530
53    NPNTB=IPNT
      CALL UTMIX(15,5,IFWD,IPNT)
      IF(IPNT) 535,535,525
535   LASTB=JADD
      JADD=JOVFL(JADD,32)
      GO TO 505
```

```
54      CALL UNMIX(5,0,IFHD,NENTRY)
        IF(ISKIP) 560,560,545
545     IF(ISKIP-NENTRY) 555,550,550
550     ISKIP=ISKIP-NENTRY
        GO TO 530
555     I=NENTRY-ISKIP
        L=I+1
        ISKIP=0
        GO TO 565
56      I=
        L=1
565     I=I+1
        IF(I-NENTRY) 580,580,570
57      IF(L-1) 530,575,530
575     ITEMP=IPNT
        GO TO 600
580     JOVFL(JADD,IPNT+I)=0
        JOVFL(JADD,IPNT)=JOVFL(JADD,IPNT)-1
        K=K+1
        IF(K-NNT) 565,565,585
585     IF(I-NENTRY) 590,570,570
590     I=I+1
        IF(I-NENTRY) 595,595,695
595     JOVFL(JADD,IPNT+L)=JOVFL(JADD,IPNT+I)
        JOVFL(JADD,IPNT+I)=0
        L=L+1
        GO TO 590
6       CALL UNMIX(10,5,IFHD,NPNTF)
        IF(NPNTF) 650,650,605
C   TO HERE TO RESTACK REMAINDER OF BLOCK
605     IDIFF=NPNTF-IPNT
61      IFHL=JOVFL(JADD,NPNTF)
        CALL UNMIX(5,0,IFHD,NENTRY)
        JOVFL(JADD,IPNT)=JOVFL(JADD,NPNTF)
        JOVFL(JADD,NPNTF)=0
        I=
615     I=I+1
        IF(I-NENTRY) 620,620,625
62      JOVFL(JADD,IPNT+I)=JOVFL(JADD,NPNTF+I)
        JOVFL(JADD,NPNTF+I)=0
        GO TO 615
625     CALL UNMIX(10,5,IFHD,NPNTF)
        IF(NPNTF) 635,635,63
63      JOVFL(JADD,IPNT)=JOVFL(JADD,IPNT)-IDIFF*32
        IPNT=NPNTF-IDIFF
        GO TO 61
```

```
635     IPNT=ITEMP
        IF(K-NNT) 530,530,695
650     IF(NPNTB) 660,660,655
655     IFULL=JOVFL(JADD,NPNTB)
        CALL PACK(5,IFULL,0,31)
        JOVFL(JADD,NPNTB)=IFULL
        IF(K-NNT) 535,535,695
C  THE PRESENT BLOCK IS DESTROYED
660     ITEMP=JADD
        IF(LASTB) 670,670,665
C  RELINK AROUND PRESENT DESTROYED BLOCK
665     JOVFL(LASTB,32)=JOVFL(JADD,32)
        GO TO 680
670     IF(JOVFL(JADD,32)) 685,685,675
C  RESET JOB'S FIRST BLOCK POINTER IN J-TABLE
675     J(24,JOB)=JOVFL(JADD,32)
680     JADD=JOVFL(JADD,32)
        JOVFL(ITEMP,32)=0
        JOVFL(ITEMP,IPNT)=0
        IF(K-NNT) 505,505,695
C  JOB'S OVERFLOW AREA IS DESTROYED
685     J(24,JOB)=0
        J(25,JOB)=0
        JOVFL(JADD,IPNT)=0
695     RETURN
        END
```

```
      PROGRAM FOR SOLVING ACCESS CONFLICT EQUATION


      DIMENSION R(15),P(15),A(225)
      DOUBLE PRECISION R,A,P,PIVIT,TOP,BOT,X,Y,Z,ZZ,CMR
      DOUBLE PRECISION PZRO,PNU
    1 READ(5,5) NSTEP
    5 FORMAT(I5)
      ICNT=0
   10 ICNT=ICNT+1
      IF(ICNT-NSTEP) 20,20,15
   15 GO TO 2[illegible]
   20 READ(5,5) N
      WRITE(6,25) N
   25 FORMAT(' THE NEXT SOLUTION IS FOR',I3,' CONTENDING PRO
     CCS')
      M=N-1
      Z=[illegible].DO
      Y=.25DO/N
   26 Z=Z+Y
      IF(N.EQ.2.AND.Z.GT.1./N) GO TO 10
      IF(Z.GT.3.1/N) GO TO 10
      ZZ=1.0DO-Z
      K=
   30 K=K+1
      IF(K.GE.N) GO TO 150
      LUP=MIN0(N-1,K+1)
      I=-1
   37 I=I+1
      IF(I.GT.LUP) GO TO 100
      X=1.0DO
      IBIG=MAX0(N-K-1,K+1-I)+1
      ILTL=MIN0(N-K-1,K+1-I)
      TOP=1.0DO
      BOT=1.0DO
    4 IF(IBIG.GT.N-1) GO TO 5[illegible]
      IF(TOP.GE.1.0D12) GO TO 5[illegible]
      TOP=TOP*IBIG
      IBIG=IBIG+1
      GO TO 4[illegible]
    5 IF(ILTL.LE.1) GO TO 55
      IF(BOT.GE.1.0D12) GO TO 55
      BOT=BOT*ILTL
      ILTL=ILTL-1
      GO TO 5[illegible]
```

```
55      X=X*(TOP/BOT)
        TOP=1.0D0
        BOT=1.0D0
        IF(IBIG.LE.N-I) GO TO 40
        IF(ILTL.GE.2) GO TO 50
        GO TO 70
7       X=(X*ZZ**(N-K-1))*Z**(K+1-I)
75      IF(I.GT.0) GO TO 80
        PIVIT=X
        DO 76 JJ=1,M
76      P(JJ)=-X
        GO TO 37
8       P(I)=P(I)+X
        GO TO 37
1       CM =P(K)
        I=
1.5     I=I+1
        IF(I.GT.N-1) GO TO 110
        P(I)=P(I)/(1.0D0 -CMR)
        GO TO 1
11      R(K)=-PIVIT/(1.0D0-CMR)
        P(K)=-1.0D0
        I=
115     I=I+1
        IF(I.GT.N-1) GO TO 3
        A(K+(I-1)*(N-1))=P(I)
        GO TO 115
15      IER=
        EPS=.0000015
        CALL SOLVER(R,A,M,1,EPS,IER)
        WRITE(6,16 ) N,Z,IER
16      FORMAT(/' RETURN FROM SOLVER N,Z,IER ARE',2X,I3,2X,F5.
       2,2X,I3)
        WRITE(6,17 ) N,Z
17      FORMAT(/' THE SOLUTION ARRAY FOR N =',I3,' AND Z = ',F
       5.2,' IS'/)
        WRITE(6,175) (R(I),I=1,M)
175     FORMAT(4(2X,D14.8))
        SD= 0.
        DO 18  I=1,M
18      SD=SD+FLOAT(I)/FLOAT(N)*R(I)
        SPEED=1.-SD
        WRITE(6,185) N,Z,SD,SPEED
185     FORMAT(/' FOR N = ',I3,' AND Z = ',F5.2,' SLOW DOWN DU
       E',/,        ' TO CONTENTION IS  ',F7.5,'.  EACH PROCES
       SOR IS EXPECTED',/,     ' TO PROCEED AT  ',F7.5,' NORM
```

```
     CAL SPEED.'//)
      PZRO=0.0D0
      DO 190 I=1,M
190   PZRO=PZRO+R(I)
      PZRO=1.D0-PZRO
      PNU=(ZZ**N)*PZRO
      WRITE(6,195) PNU
195   FORMAT(' EXPECTED BANDWIDTH NONUTILIZATION = ',D14.8)
      GO TO 26
200   STOP
      END
```

```
      SUBROUTINE SOLVER(R,A,M,N,EPS,IER)
      DIMENSION A(1),R(1)
      DOUBLE PRECISION R,A,PIV,TB,TOL,PIVI
      IF(M) 23,23,1
C     SEARCH FOR GREATEST ELEMENT IN MATRIX A
1     IER=0
      PIV=0.0D0
      MM=M*M
      NM=N*M
      DO 3 L=1,MM
      TB=DABS(A(L))
      IF(TB-PIV) 3,3,2
2     PIV=TB
      I=L
3      CONTINUE
      TOL=EPS*PIV
C     A(I) IS PIVOT ELEMENT.  PIV CONTAINS ABS VALUE OF A(I).
C     START ELIMINATION LOOP
      LST=1
      DO 17 K=1,M
C     TEST ON SINGULARITY
      IF(PIV) 23,23,4
4     IF(IER) 7,5,7
5     IF(PIV-TOL) 6,6,7
6      IER=K-1
7     PIVI=1.0D0/A(I)
      J=(I-1)/M
      I=I-J*M-K
      J=J+1-K
C     I+K IS ROW INDEX, J+K COLUMN-INDEX OF PIVOT ELEMENT
C     PIVOT ROW REDUCTION AND ROW INTERCHANGE IN RIGHT HAND SI
CDE R
      DO 8 L=K,NM,M
      LL=L+I
      TB=PIVI*R(LL)
      R(LL)=R(L)
8      R(L)=TB
C     IS ELIMINATION TERMINATED
      IF(K-M) 9,18,18
C     COLUMN INTERCHANGE IN MATRIX A
9     LEND=LST+M-K
      IF(J) 12,12,10
10    II=J*M
      DO 11 L=LST,LEND
      TB=A(L)
```

```
      LL=L+II
      A(L)=A(LL)
11    A(LL)=TB
C   ROW INTERCHANGE AND PIVOT ROW REDUCTION IN MATRIX A
12    DO 13 L=LST,MM,M
      LL=L+I
      TB=PIVI*A(LL)
      A(LL)=A(L)
13    A(L)=TB
C   SAVE COLUMN INTERCHANGE INFORMATION
      A(LST)=J
C   ELEMENT REDUCTION AND NEXT PIVOT SEARCH
      PIV=0.0D0
      LST=LST+1
      J=0
      DO 16 II=LST,LEND
      PIVI=-A(II)
      IST=II+M
      J=J+1
      DO 15 L=IST,MM,M
      LL=L-J
      A(L)=A(L)+PIVI*A(LL)
      TB=DABS(A(L))
      IF(TB-PIV) 15,15,14
14    PIV=TB
      I=L
15    CONTINUE
      DO 16 L=K,NM,M
      LL=L+J
16    R(LL)=R(LL)+PIVI*R(L)
17    LST=LST+M
C   END OF ELIMINATION LOOP
C   BACK SUBSTITUTION AND BACK INTERCHANGE
18    IF(M-1) 23,22,19
19    IST=MM+M
      LST=M+1
      DO 21 I=2,M
      II=LST-I
      IST=IST-LST
      L=IST-M
      L=A(L)+.5D)
      DO 21 J=II,NM,M
      TB=R(J)
      LL=J
      DO 20 K=IST,MM,M
      LL=LL+1
```

```
20      TB=TB-A(K)*R(LL)
        K=J+L
        R(J)=R(K)
21      R(K)=TB
22      RETURN
C     ERROR RETURN
23      IER=-1
        RETURN
        END
```

```
      PROGRAM FOR DETERMINING SLOW DOWN AND
      BANDWIDTH UTILIZATION UNDER LOCALIZED
      BANDWIDTH LIMITATION AND MISMATCH


      DIMENSION ISLOW(40),IUTL(40),BW(16),IBWR(40)
      READ(5,10) NRUNS,NSEED
10    FORMAT(2(2X,I10))
      WRITE(6,10) NRUNS,NSEED
      IX=NSEED
      NR=0
15    NR=NR+1
      IF(NR.GT.NRUNS) GO TO 300
      READ(5,20) NPROS,NSAMP,W
      NPRO=NPROS-1
20    FORMAT(2(2X,I10),7X,F5.3)
      BW(1)=0.0
      TOTAL=(FLOAT(NPROS)*.5)*FLOAT(NPRO)
      I=1
30    I=I+1
      IF(I.GT.NPROS) GO TO 35
      BW(I)=((I-1)*W)/TOTAL
      GO TO 30
35    UMEAN=0.0
      SMEAN=0.0
      RMEAN=0.0
      USTD=0.0
      SSTD=0.0
      RSTD=0.0
      DO 37 I=1,40
      ISLOW(I)=0
      IBWR(I)=0
37    IUTL(I)=0
      NS=0
40    NS=NS+1
      IF(NS.LE.NSAMP) GO TO 70
      UMEAN=UMEAN/NSAMP
      SMEAN=SMEAN/NSAMP
      RMEAN=RMEAN/NSAMP
      USTD=SQRT(USTD/NSAMP-UMEAN**2)
      SSTD=SQRT(SSTD/NSAMP-SMEAN**2)
      RSTD=SQRT(RSTD/NSAMP-RMEAN**2)
      WRITE(6,45) NPROS,NSAMP,W,UMEAN,USTD,SMEAN,SSTD
45    FORMAT(/,' NPROS,NSAMP,W,UMEAN,USTD,SMEAN,SSTD ARE',/,
                  2(2X,I1 ),5(2X,F7.5))
```

```
C
      WRITE(6,46) RMEAN,RSTD
46    FORMAT(' RMEAN AND RSTD ARE',2(2X,F8.5))
      WRITE(6,50) ISLOW
50    FORMAT(/,20(1X,I5))
      WRITE(6,50) IUTL
      WRITE(6,50) IBWR
      GO TO 15
70    I=1
      BWR=0.0
75    I=I+1
      IF(I.GT.NPROS) GO TO 100
      CALL RANDU(IX,IY,RAN)
      IX=IY
      INDEX=IFIX(RAN*NPRO+.5)+1
      BWR=BWR+BW(INDEX)
      GO TO 75
100   IF(BWR.GT.1) GO TO 110
      BWU=BWR
      SLDN=0.0
      GO TO 120
110   BWU=1.0
      SLDN=(BWR-1.)/BWR
120   INDXS=IFIX(SLDN*39.+.5)+1
      INDXU=IFIX(BWU*39.+.5)+1
      INDXB=IFIX(BWR*39./(NPROS*BW(NPROS))+.5)+1
      ISLOW(INDXS)=ISLOW(INDXS)+1
      IUTL(INDXU)=IUTL(INDXU)+1
      IBWR(INDXB)=IBWR(INDXB)+1
      UMEAN=UMEAN+BWU
      USTD=USTD+BWU**2
      SMEAN=SMEAN+SLDN
      SSTD=SSTD+SLDN**2
      RMEAN=RMEAN+BWR
      RSTD=RSTD+BWR**2
      GO TO 40
      STOP
      END
```

```
      SUBROUTINE RANDU(IX,IY,YFL)
      IY=IX*65539
      IF(IY) 5,6,6
    5 IY=IY+2147483647+1
    6 YFL=IY
      YFL=YFL*.4656613E-9
      RETURN
      END
```

```
      PROGRAM FOR DETERMINING SERVICE FOR PRIORITY
      DRIVEN CONFLICT RESOLUTION SCHEME


      DIMENSION IP(16),IACR1(16),NAEP(16)
      DIMENSION DIFF(40),DIFP(40),DIFB(40)
      INTEGER DIFF,DIFP,DIFB
      READ(5,10) NRUNS,ISEED
10    FORMAT(2(2X,I10))
      NR=0
15    NR=NR+1
      IF(NR.GT.NRUNS) GO TO 200
      READ(5,11) NPROCS,NSTEPS
      WRITE(6,20) NSTEPS,NPROCS,ISEED
20    FORMAT(//,' THE NEXT ',I5,' STEPS IS FOR ',I3,' PROCES
     &SORS',/,     ' ISEED = ',I10)
      C
      WRITE(6,22)
22    FORMAT(' LABEL   L1-2 RATIO OF TP TO PL  IP(1 THRU NPRO
     &CS)')
      M=NPROCS-1
      DO 25 I=1,40
      DIFF(I)=0
      DIFP(I)=0
25    DIFB(I)=0
      IACCEPT=0
      IREJP=0
      IREJB=0
      PMEAN=0.0
      PMEAN1=0.0
      PMEAN2=0.0
      AF=0.0
      AP=0.0
      CSTP=0.0
      TS=0.0
3     NS=NS+1
      IF(NS.GT.NSTEPS) GO TO 150
      DO 35 I=1,NPROCS
      CALL RANDU(ISEED,IX,RAN)
      ISEED=IX
      NAEP(I)=0
35    IP(I)=IFIX(RAN*5.)+1
      DO 4  I=1,NPROCS
      DO 4  J=I,NPROCS
```

```
      IF(IP(I).GE.IP(J)) GO TO 4
      ITEMP=IP(I)
      IP(I)=IP(J)
      IP(J)=ITEMP
4     CONTINUE
      DO 45 I=1,NPROS
45    IACR1(I)=IP(I)
      IFLAG=
      JJ=
5     IBIG=1
      JJ=JJ+1
      J=
55    J=J+1
      IF(J.GT.NPROS) GO TO 6
      IF(IACR1(J).LE.IACR1(IBIG)) GO TO 55
      IBIG=J
      GO TO 55
6     IACR1(IBIG)=
      NAEP(IBIG)=NAEP(IBIG)+1
      DO 65 I=1,NPROS
65    IACR1(I)=IACR1(I)+IP(I)
      IF(IBIG.NE.NPROS) GO TO 5
      IF(IFLAG.EQ.1) GO TO 67
      LBLP=JJ
      IFLAG=1
      JJ=
      GO TO 5
67    ISUM=
      DO 7 I=1,NPROS
7     ISUM=ISUM+IP(I)
      IRSUM=ISUM/IP(NPROS)
      WRITE(6,8) LBLP,JJ,IRSUM,(IP(I),I=1,NPROS)
8     FORMAT(19(1X,I5))
      BRATE=FLOAT(NAEP(1)*IP(NPROS))/FLOAT(IP(1)*NAEP(NPROS)
     ,)
      WRITE(6,82) (NAEP(I),I=1,NPROS)
82    FORMAT(' NO. ACCES BY PROC',16(1X,I5))
      WRITE(6,83) BRATE
83    FORMAT('+',118X,F7.4)
      BMEAN=BMEAN+BRATE
      BSTD=BSTD+BRATE**2
      FF=FLOAT(LBLP-IRSUM)/FLOAT(IRSUM)+.5
      AF=AF+FF**2
      FMEAN=FMEAN+FF
      INDEX=IFIX(FF*4.+.99999)
      IF(INDEX.GT.1.AND.INDEX.LE.4) GO TO 95
```

```
      IRNGF=IRNGF+1
      GO TO 90
85    DIFF(INDEX)=DIFF(INDEX)+1
90    FF=FLOAT(JJ-IRSUM)/FLOAT(IRSUM)+.5
      AP=AP+FF**2
      PMEAN=PMEAN+FF
      INDEX=IFIX(FF*4.0+.99999)
      IF(INDEX.GT.0.AND.INDEX.LE.40) GO TO 95
      IRNGP=IRNGP+1
      GO TO 100
95    DIFP(INDEX)=DIFP(INDEX)+1
100   INDEX=IFIX(BRATE*2.0+.99999)
      IF(INDEX.GT.0.AND.INDEX.LE.40) GO TO 105
      IRNGB=IRNGB+1
      GO TO 90
105   DIFB(INDEX)=DIFB(INDEX)+1
      GO TO 90
150   WRITE(6,155) IRNGF,DIFF
155   FORMAT(/,21(1X,I5))
      WRITE(6,155) IRNGP,DIFP
      WRITE(6,155) IRNGB,DIFB
      FMEAN=FMEAN/NSTEPS
      PMEAN=PMEAN/NSTEPS
      BMEAN=BMEAN/NSTEPS
      FSTD=SQRT(AF/NSTEPS-FMEAN**2)
      PSTD=SQRT(AP/NSTEPS-PMEAN**2)
      BSTD=SQRT(BSTD/NSTEPS-BMEAN**2)
      WRITE(6,165) NPROS,FMEAN,FSTD
165   FORMAT(/' FOR',I3,' PROS, FMEAN W.R.T. SUM(P(I))/P(L)
     CAND',        ' FSTD ARE',2(2X,F8.5))

      WRITE(6,170) NPROS,PMEAN,PSTD
170   FORMAT(' FOR',I3,' PROS, PMEAN W.R.T. SUM(P(I))/P(L) A
     CAND',        ' PSTD ARE',2(2X,F8.5))

      WRITE(6,175) NPROS,BMEAN,BSTD
175   FORMAT(' FOR',I3,' PROS, MEAN AND STANDARD DEVIATION O
     F SERVICE TOHIGHEST PRIORITY',/,' JOB RELATIVE TO LOWE
     ST PRIORITY JOB ARE',2(2X,F8.5))

      GO TO 15
      STOP
      END
```