72-20,009

ZIMMERLI, Dana Wayne, 1942-
SYMBIOTIC COMPUTER SYSTEM MEASUREMENT AND EVALUATION.

Iowa State University, Ph.D., 1972
Computer Science

Symbiotic computer system measurement and evaluation

by

Dana Wayne Zimmerli

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of

The Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subjects: Electrical Engineering
Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Departments

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1972

TABLE OF CONTENTS

LIST OF FIGURES

PART ONE.

THE PROBLEM OF EVALUATION

INTRODUCTION

Computer evaluation began as soon as the second computer was developed. The inevitable comparisons have been made over and over again by persons interested in using computers. Early evaluations were primarily comparisons of the hardware characteristics, but the development of the computer operating system added another facet to the problem. Where simple numeric comparison is sufficient for a hardware comparison, the software is much more complicated.

It is the purpose of this dissertation to present a method for the evaluation of an operating system. This method is a three-step process which is described in some detail. The basis for the evaluation is a set of data which is obtained by direct measurement of an average job stream in the computer installation. This data is a microscopic trace of all of the important events occurring during the time of measurement. This data is then restructured into a job-oriented representation of the job stream. As the data is restructured, the characteristics of the job stream are accumulated and presented for preliminary analysis. The final step of the evaluation is somewhat iterative in nature. The restructured data is used as input to a simulation model of the operating system. The simulation model may then be varied to optimize the performance of the job stream as applied to the model.

This dissertation describes the choice of parameters to be measured, using several references as guides to these choices. The measurability of these parameters is experimentally verified by measurements on an IBM 360/65 system. The resulting data is accumulated and restructured as described above, and some interesting observations are made. Finally, a simulation language is developed which supports the job stream data as input, and provides the necessary features to simulate an operating system.

INTRODUCTION TO EVALUATION

Early computer programmers enjoyed a sometimes enviable rapport with the computer. A detailed knowledge of the computer's characteristics was necessary to produce efficient programs. Many man-hours were necessary to develop a program and patience was a most important quality in the programmer. Often, only numeric codes were available for programming purposes. Debugging was usually done by tracing the program on display lights and programmers worked directly with the machine.

The sole purpose of early operating systems was the reduction of computer idle time. Even a casual observer could easily discern the primary sources of idle time in an environment where the programmer marched into the machine room with his card decks and listings, preparatory to an extended session of playing with the console keyboard. Whenever the program failed and the machine stopped, the programmer scratched his head and tried to figure out his next move. When the job was completed or the time period expired, the transfer of material to the departing programmer and from the incoming programmer caused more idle time.

The initial thrust of the computer operating system was to provide some means of isolating the programmer from the computer. Professional computer operators were hired to perform some of the mechanical operations involved with program loading and execution. These operators certainly cut the amount of time required to transfer from job to job, but this time period still was noticeable.

The next step was to automate these operator functions as much as possible. This automation required rigid control of access to the machine console. Programming aids such as dump routines, loader programs, and symbolic assemblers were necessary to free the programmer from console debugging. These aids also allowed programmers to spend more time on the logic of their programs and less time to the mechanics of coding the program.

The development of the first computer operating system has been attributed to the General Motors Research Laboratories by Steel (30). General Motors developed and used an elementary operating system for their 701 computer and later collaborated with the Los Angeles division of North American Aviation to produce a similar system for a 704 computer. Shortly after the operating system concept became generally known, the idea jumped the boundaries of machine type. Today operating systems are used almost universally in connection with large computers.

Batch processing or jobstacking was the basic idea of early operating systems. Rather than reading each job into the machine independently, a collection of jobs (a batch) is gathered into an input stream together with their respective data. A supervisor program, which is normally kept in the main storage unit, loads the next job from the input file and initiates execution of that job. Upon completion of the job, control must be returned to the supervisor, which then selects the next job in the input stream.

The remaining idle time during program execution was connected with the assignment of input-output(I/O) devices to particular jobs. The complex logistics of these assignments was solved by removing all I/O from the programmer's control. Standard routines that are part of the operating system were added to perform the I/O operations with standardized constraints. Additionally, error recovery was standardized for all programmers.

These changes and requirements allowed the efficiency of the computer to increase by reducing the idle time and delivering more machine cycles to the users. At the same time, changes were occurring in the area of programming. A desire to express programs in a problem-oriented language led to the development of languages which guided researchers into the development of what has become known as a compiler. The compiler allowed programmers to express their programs in a language which is reasonably close to natural language. The compiler is the software device which allows a programming language to be converted to a machine code for execution.

These problem-oriented languages are easy to learn and remember because they are closely related to both the problems to be solved and a natural language (usually English). These languages are also easier than machine or assembler languages because of the macro like properties of each statement and the automatic management of storage that they provide. This ease of use propagated the use of the computer into many new areas.

Unfortunately, dumps of memory at abnormal completion were no longer as useful in error diagnosis. The compiler obscured the executed code because a one-to-one relationship was no longer possible. Error diagnostics were then developed within the compilers to aid the programmer. These diagnostics aided the debugging process by classifying the errors and indicating, where possible, the area of trouble in the program. The diagnostics removed some of the last objections to the restricted access to the computer console. The programmer could no longer gain very much by watching the console. The isolation of programmer and console was thus achieved.

As programming became more sophisticated and complex, routines were developed and shared for common problems. To use these routines in many different applications, the idea of the relocatable loader was developed. Using this concept, programs or parts of programs could then be written in such a manner that they could be loaded into any location in memory and then could be dynamically linked to any other program.

To control and direct the computer, job control languages were developed. The allocation of resources such as primary and secondary storage and I/O devices is controlled by specific statements in the input stream. These resources are then logically or actually connected to the program by the operating system. Common or standard routines are also combined with the program under the direction of the job control statements. Operator action can then be directed by the operating system, and resource allocation can be automatically determined.

Idle time was now observed in the time required to transfer data from I/O devices and secondary storage into or out of the main memory. The necessity of communicating with the programmer required mechanical operations which were slower than the internal operations. This communication time was unused since the program had to wait for the data. Since computer personnel were interested in the over-all efficiency, some form of overlapping the internal operations was soon attempted.

The ability to overlap I/O and computation was made possible by the principle of cycle-stealing in memory devices. Cycle-stealing allows the I/O equipment to obtain memory cycles on demand for fetching or storing data. These memory cycles can often be fitted in during times when the CPU is not accessing memory. This sharing of memory allows the independent action of the I/O devices and the CPU.

Multiprogramming, the concept of executing more than one programming job at a time, was an attempt to utilize a greater proportion of the computer time and memory. Under multiprogramming, separate programs may be written to process the input-output requirements. This transfer of the data (called spooling) onto faster I/O devices such as disc, drum, or tape allowed the programs to make the logical connection for I/O to one of these faster devices. With multiprogramming the overlap of I/O and computation was permitted and greater efficiency resulted.

The concept of a real-time system originated with the defense installations where an immediate response was necessary. A real-time system allows a user direct interaction with the computer for computation purposes. This interaction eliminates the clerical details (both external and internal) of the computer operation. The concept of multiprogramming allows several stations or terminals to interact with a single computer. The coexistence of both real-time programs and normal batch processing is therefore allowed in large machines.

As this computer system evolution occurred, many potentially incorrect assumptions were made. Insufficient analysis may have contributed to some of these incorrect assumptions, but many of them have been made on the basis of intuition alone. As these incorrect assumptions were uncovered, more emphasis was placed on the use of deterministic and probabilistic measurements.

As computer systems have become more complex, evaluation of these systems has become more difficult. Many new elements must be considered as part of the computer system. Since one objective of an operating system is to aid programmers by providing common routines, computing time must be spent to allow generality. Multiprogramming requires large quantities of information about each job. This information is used to define the transfer between jobs and to start and stop each job. Updating and maintaining this job information requires an additional part of the available computer time. In this case, a tradeoff occurs between the capability of the computing system and the manual or semimanual procedures surrounding the computer. As the computing system makes more of the decisions in the scheduling, allocation of resources, and operation of these resources, time is required which is no longer available to the user. Evaluation must therefore consider many more parts than ever before, because these procedures are a part of the considerations in a large system.

In any discussion on computer system evaluation, the characteristics which are to be compared influence the evaluation. Historically, computers were divided into two classes, scientific and commercial. Scientific computers were measured almost exclusively in terms of their arithmetic speed. Certain discrete operations such as add, subtract, multiply, load, and store were considered to be the main activity of scientific operations. The tacit assumption that arithmetic processing was the dominant function for consuming time was the justification for this approach to evaluation. Transferring data into and out of primary storage was assumed to occur only a small fraction of the time.

In the commercial processing field the input-output capabilities were considered the most important factor. Deep commitments to card-processing techniques, where literally tons of data had to be processed led to assessment of systems in terms of their record reading and writing rates. In fact, the first commercial processors were little more than collators and sorters.

Presently, no clear division of computers is possible. Jobs which are structured much like a "commercial" application are now found in the "scientific" computers and vice versa. This blurring of the distinction between scientific and commercial computers has complicated the task of performance evaluation even further. Evaluation techniques must now be made general enough to cover both types of computer. In fact, the two types of computer have merged into only one, the general-purpose or universal computer.

Why Evaluate ?

The nature of the widespread interest in evaluation is difficult to classify. Users need a basis to compare competing proposals, a basis for acceptance testing, and ways of selecting and describing systems tasks. Users must also be able to estimate the running time and costs of new tasks for planning purposes.

A common problem is the determination of a new configuration. Should new devices be installed? Will a faster CPU be utilized? Is more secondary storage necessary? Will data channels, additional I/O devices or anything else add to the cost-effectiveness of the system performance? These questions are among some of the things users would like answered by system evaluation. System programmers and computer designers are faced with the problem of determining the performance of systems under development. During system development, evaluation is an important aid to the system designers both for verification of the performance and for direction in determining new features to be added. Consequently, performance evaluations are extensively used in both software and hardware development. An example of this type of development evaluation may be found in the Multics project described by

Saltzer and Gintell (20). The evaluation system used in this project consists of both hardware and software devices combined. The important hardware features described are a high resolution clock capable of reading accurately to a microsecond, an internal memory cycle counter, and an externally driven I/O channel which permits another computer to access the computer under test. Most of the software evaluation tools in the Multics project are concerned with the measurement of time spent in certain sections of the experimental system, or with the number of times a particular event occurs during program execution.

Another application for system evaluation is in the field of system optimization. Each computer installation has a unique distribution of job characteristics as determined by their users. Proper system parameters must be chosen to optimize the system performance for the user community at each installation. Job scheduling algorithms, page swapping (in a virtual memory), time-slicing, and priority levels are some of the potential areas to be modified during system optimization. These items may be part of the normal system maintenance and may require change due to a change in the users' job characteristics. Optimization of individual programs such as compilers may be contained in this application of evaluation, as well as the gross system characteristics.

The selection and acquisition of new equipment is another area where system evaluation may aid a computer installation. Often evaluation can identify a potential problem or deficiency before it becomes serious. These problems may be related to an equipment deficiency which can be corrected by the addition of more or different devices. These deficiencies may require some additional evaluation or testing to determine the proper actions, but the time may be available due to the foresight given by the evaluation.

Evaluation may also be used to determine the relative merits of several competing philosophies. An example of this form of evaluation is described by Sherman, et al. (27). This paper describes the evaluation of several CPU scheduling philosophies. The evaluation is a simulation which produces a comparison between the scheduling techniques. Theoretical results for other computer processes may also be tested by suitable evaluations.

In summary, evaluation is a desirable activity for computer systems personnel, because it provides better insight into the operation of the system. This insight may then be used to optimize the operation of the system. The requirement for evaluation leads to a desire to provide a systematic method for evaluation. The remainder of this dissertation describes a system which may fulfill this requirement.

CLASSES OF EVALUATION

Evaluation falls into two primary classes according to Drummond (5). These are availability and work capability. Availability expresses how much of the time a system (or part of a system) is or can be used for productive purposes. Work capability is an assessment of a system's ability and efficiency as applied to performing an intended function.

Availability may be defined in absolute terms as the time the computer is on (power applied) minus the portion of that time which is required for maintenance. The reduction of maintenance time therefore increases the availability. Two forms of maintenance are common, scheduled and unscheduled. Scheduled maintenance may be planned ahead and may not be a serious loss of availability, if the time period chosen is during a period of low usage requirements.

Unscheduled maintenance can be very critical because it is unpredictable and may last for an indeterminate length of time. Unscheduled maintenance is directly dependent on the reliability of the entire system. The reliability of the system is dependent on the quality of the components and the construction of the computer. In this area, consideration must be given to the fact that some failures may be transient in nature, so error correction and error re-try schemes may extend the availability of the computer.

Additional methods of decreasing unscheduled maintenance time are dependent on by-passing the failing part or parts. Multiple or redundant parts may be substituted for the unavailable part until a more convenient time period allows the bad part to be replaced. This redundancy is usually only used for highly critical applications and is usually quite costly.

A similar scheme of increasing availability involves modification of the system so that the work which does not require the failing part may still be performed. This allows partial availability to the computer so some work could still be performed.

Work capability is measured by many forms, with the three most popular being job time, throughput, and response time or turnaround time. Job time is a measure of the time it takes to process a particular application. This measure of job time does not commonly account for the external clerical portion (the handling of the input decks and the output listings) of the job. To determine the relative performance of a computer system, a synthetic job has been formulated. This job is described by Buchholz (2) as a "greatly simplified file maintenance procedure". He postulates that it can be "programmed with a modest effort in different languages and on dissimilar machines, so as to be run and timed on each of the systems". This job exercises both the CPU and the major I/O devices of a computer. Naturally the data obtained is valid only for this particular job in the particular environment in which it is executed. In another job in another environment, the results may be considerably different.

Throughput is a generic term which relates in some way to doing the total work of the system, rather than any single job. In a multiprogramming environment, the number of jobs per day may be cited as a measure of throughput. The use of throughput as a relative measure estimates the performance of a computing system when it is measured against some base computing system. Relative system throughput is defined as the ratio of the time of computation for a given load on the base system divided by the time of computation for the same load on the new system. Naturally, the systems to be compared must have similar or equivalent facilities or the comparison will be invalid.

Response time, a term generally associated with real-time systems, is usually measured in absolute terms. In terminal-oriented systems, response time refers to the amount of time that the computing system takes to react to terminal transactions. In other real-time systems, response time can indicate the time needed to identify, load, and execute a critical function. Although no response is necessary, completion of some critical processing may be required. Response time calculations must be well defined within the context of their intended use.

Turnaround time is generally associated with batch processing systems to imply the same relative time period as response time. Turnaround time is usually defined as the time between turning in a job at a station and the time that the results are received. Turnaround time does include the time required for the external clerical handling necessary to execute the job.

Acceptable computer performance must be a mixture of these factors. The programmer is usually most interested in job time and turnaround or response time. The job time is, of course, directly related to the turnaround and response time. The effectiveness of an individual programmer may be partially dependent upon the turnaround time. Job time, on the other hand, is a measure of the cost of an individual job. System programmers and operations personnel are probably more concerned with the throughput, because this is a measure of the number of people the computer is serving. Basically, programmers or users are interested in the factors that affect their jobs, where system programmers and operations personnel are more interested in the total system.

TYPES OF EVALUATION

The three primary types of evaluation are classification, comparison, and time estimation. Classification is probably the most popular form of evaluation, although evaluation of a set of attributes or a single attribute by classification may be misleading. Classification may investigate attributes such as capacity of main storage, storage cycle time, or add time, and attempt to tabulate computers into classes based on these properties. Often vague terms such as small, intermediate, and large systems accompany evaluation by classification.

Comparative evaluations usually designate one system as a base against which all other systems are compared. Like other types of evaluation, comparative evaluation often considers only the CPU and processor storage. The interdependent methods of the instruction mix and the kernel have been developed for comparative evaluation. The mix method assigns a weight to each instruction or group of instructions. The weighted instruction time can be used to compute an average instruction time for comparison purposes. The kernel method examines the central or essential part of the application under study. The most frequently used portions of an application are determined and these portions are programmed in the various instruction sets. Continuing this to programming the entire job stream would allow a comparison of system throughput.

Time estimation involves estimating the time involved for required functions or operations. The comparison then could involve entire jobs and all system components. The time estimate may or may not be the desired end.

Analysis of Evaluation Methods

As Calingaert (3) has shown, the above mentioned types of evaluation have proven inadequate to produce meaningful results in present-day computers. The simplifications and approximations used can cause large discrepancies in the results. Application of these erroneous measures may then incorrectly bias the opinions of users.

The first method of evaluation was the classification of instructions and other absolute data items. This form of evaluation is an over-simplification of the problem and does not consider the additional structure of a viable system. An example of this problem is the comparison of memory times. If only memory times are compared, the amount of information transferred per access may later cloud the comparison. A comparison of the amount of information transmitted per unit time (bandwidth) may be more accurate, but the other factors in the memory may still enhance or diminish the significance of the overall memory speed as a measure of the system.

Instruction time comparison can also be influenced by the other parts of the computer structure. The add instruction is a common instruction for comparison. It must be recognized that no one instruction can adequately describe a computer system, but even if this one instruction is considered interesting, are the word lengths equal in both machines? If the machines have character addressability, what operand length should be chosen and why? The addressing schemes for different machines may vary radically, so what effect will addressing have? These are a few of the problems involved in instruction time comparison.

Calingaert has discussed some of the problems with the instruction mix and kernel methods. The instruction mix technique must be based on a measurement of the execution of several programs through a large number of instructions, and is therefore dependent upon the structure of this original CPU. The coefficient which weights each instruction time supposedly represents the relative frequency of instruction occurrence. As the structure of the CPU varies from the original, the instruction mix becomes less and less applicable. To illustrate this effect, Calingaert cites an experiment performed with a group of experienced system engineers. "Its members were asked to specify the time in microseconds on System/360 Model 40 for the compare class of instructions, given only the fact that the original mix was based on the 7090, where the instructions in that class were CAS and LAS. The ten answers ranged from 11.88 to 30.66 with a mean of 21.5 and a standard deviation of 7.0".

Kernels, like instruction mixes, are not free of disadvantages. The problem of providing equal programming skill for the different CPU's is a practical limitation in both personnel requirements and implementation time. The proper weighting of two or more kernels can also be a problem. Calingaert (3) again cites a difference in performance ratios of one CPU compared against another. Different kernels yielded ratios as high as 9.5 and as low as 3.3. There is strong evidence that all presently identified kernels are atypical and typical kernels may not be definable in the general sense.

The time estimation technique relies heavily on the thorough understanding of the processes involved and requires careful analysis of available data. Verification of the results depends on the subsequent measurements of the transplanted system. Time estimation may be performed by a simulation of the system. In this case, the accuracy of the result is dependent on the knowledge of the designer.

A recent survey of performance evaluation by Lucas (13) rates several techniques for evaluation in terms of three main purposes. These three purposes for evaluation are selection evaluation, performance projection, and performance monitoring. Selection evaluation is the process associated with obtaining new equipment or programs which already exist; Performance projection is that part of the decision activity which preceeds the design and implementation of both new hardware and new software; Performance monitoring is the constant measurement process used to evaluate the performance of a production system. Each of the eight techniques (instruction times, instruction mixes, kernels, models, benchmarks, synthetic programs, simulation, and monitoring) is rated in terms of its suitability to the purposes for performance evaluation. The most satisfactory technique is postulated to be simulation, but simulation has drawbacks in the cost of running the simulation, the validation of the simulation results, and the question of the necessary problem of the level of detail required to produce valid results.

Many additional techniques of evaluation have been described. An excellent bibliography of computer performance analysis techniques has been compiled by Miller (15). The references in this bibliography cover all aspects of performance measurement. Itemized listings of the references included within particular areas of performance evaluation are also included. The bibliography included with the article by Lucas (13) is also comprehensive and current.

EVALUATION DATA

In designing an evaluation, consideration must be given to the data which must be acquired. The choice of the data to be collected greatly influences the evaluation because systems are not the same. One system may be weak in the same area that another system has its strength. If the extremes of the systems are tested, the evaluation loses validity because the environment is no longer typical but must be artificial. Representative information obtained from a complete jobstream is better for the evaluation, but the volume of data makes it difficult to analyze. One way of overcoming this problem is to make use of a benchmark program. Drummond (5) defines a benchmark as a "particular programmed procedure with some associated data chosen in such a way as to impart meaning to the originator of the benchmark". For the scientific problem, matrix inversion is a typical example of a benchmark. Another alternate job to be used as a benchmark is the synthetic job discussed earlier. With these programs, gross measurements of time might be enough upon which to base an evaluation.

Two main classes of data acquisition are common: hardware measurement and software measurement. Software measurement is able to obtain data related to individual jobs and provide probabilistic data to indicate usage distributions.

Certain data which is job-oriented may be obtained only by a software monitor which may be tailored to fit the system. On the other hand, hardware monitors do not easily acquire system related information, but rather describe the hardware utilization of the system. Certain hardware related information such as instruction usage distributions may be gathered most conveniently by a hardware monitor. In addition, hardware monitors can be attached so that the rest of the system is not affected by the measurement.

Software techniques generally intercept the normal flow of execution at particular points where information is desired. The complexity and duration of the interception depends on the information required and the information known at that point. Locating the necessary information may require extensive searching through memory. Intimate knowledge of the system being measured is necessary to obtain the proper information at the proper point. Examples of this technique are given by Stanley and Hertel (29), Stanley (28), and Scherr (22).

Stanley and Hertel (29) present a measurement system for the real-time system used in the Apollo space flights. Their system collects data designed to provide performance measures and to allow testing of the system for the expected loads during an Apollo space flight. This data is collected by a software monitor which records time in terms of an accumulated total time for each function. Data are not in general associated with a particular job since all jobs are equally important, but certain tasks are separately monitored.

Stanley (28) presents a system in a later article which measures certain parameters which are presented in a later part of this dissertation. In this article, the data is produced as a part of the job accounting system used in a real-time system. The operating system was modified to perform this accounting activity by adding computer instructions in those areas where data collection was necessary. This is then a permanent collection device which does interface directly into the system.

Scherr (22) also presents a monitoring system for another real-time system. The definitions used for this real-time system suggest a different set of parameters to be measured, but otherwise the system resembles the job accounting system presented by Stanley.

A second method of software measurement is the "snapshot" approach. At regular intervals, selected portions of the computer memory are dumped to the collection device. By applying statistical methods to these data a set of distributions may be produced which represent the measured data. This sampling of the system produces a probabilistic rather than a deterministic measure of the desired data.

Hardware monitor devices generally sense electrical signals at critical points in the CPU to determine what is happening in the system. These signals must be decoded by the monitor, which may be as complex as a small computer, and may have an interactive or immediate display. Perhaps the most dramatic attribute of the hardware monitor is its ability to obtain data reflecting the occurrence and duration of many events simultaneously. Description of a hardware monitor is given in the paper by Bonner (1). This monitor may be used to measure the activity of the CPU and the I/O channel activity. This information may be used to classify a system as CPU bound or I/O bound and also indicate I/O channel overloading. In addition, this monitor may be used to monitor the time spent within a particular protect key which may be associated with a particular job. Thus, certain important jobs may be monitored.

After data are obtained, a certain amount of analysis is immediately possible. Graphs and charts may be prepared such as those by Scherr (22). These graphs may describe the characteristics of the system and the job stream into the system. Probability densities of program size, processor time, and response time are typical of the useful measurements. A careful examination of these figures may lead to necessary answers. All the other methods of evaluation previously described may be used to extract the maximum possible information from these data.

Required Data for Evaluation

A minimal set of data is necessary to adequately describe the computer system which is being evaluated. This minimal set may vary due to the characteristics of the system being studied but certain parameters should be common to all systems. These parameters must completely describe the significant characteristics of the system, including both batch processing and time-sharing applications.

In an article describing an experimental simulation of System/360, Katz (11) describes a set of parameters which represent each job and each job step. The parameters pertaining to each job as a whole are:

(1) Job identification number.

(2) Time job is submitted.

(3) Station at which job arrives.

(4) Job priority.

(5) Keypunching time.

(6) Number of job steps.

(7) The device class which specifies the input devices that can service this job's input.

Parameters that characterize each step of the job are the following:

(1) The core storage requirement.

(2) The base time for the job step, i.e. the minimal execution time for the step.

(3) The programmer specified time limit for the job step.

(4) The number of data sets belonging to the job step.

(5) Whether the job step requires setup.

Parameters that characterize each data set belonging to each job step are:

(1) The device class whose equipments may be assigned to the data set.

(2) The storage which needs to be allocated for the data set.

(3) The programmer's estimate for the quantity of data in the data set.

(4) The actual quantity of data in the data set.

(5) The variance of the data rate to and from the data set.

(6) Whether the volume assigned to the data set needs to be retained for subsequent job steps.

(7) Whether the volume assigned to the data set is private, i.e., must not be shared by any other data set.

(8) An identifier of that data set, if any, to which this data set has a unit affinity.

(9) Whether the data set is new (was generated during the job step), old (was in existence at the beginning of the job step), or modified (was developed during the job step by modifying an existing data set).

(10) The output class to which the data set belongs (relevant only if the data set constitutes output).

(11) The disposition to be made of this data set. Possible dispositions are: sysout, an output data set; temporary, hold the data set for the duration of the entire job rather than for the current job step only; delete, destroy the data set following the current job step; keep, hold the data set indefinitely - until a subsequent delete.

In this set of parameters very little information is available on the system activity which is also present in all computer systems. A later article by Stanley (28) includes more system information. Stanley's choices of parameters are divided into two classes, job statistics and step statistics. The class of job statistics includes:

(TOTAL COUNTS)

(1) Jobs run.

(2) IPL's (initial program load) necessary.

(3) Abnormally terminated jobs.

(4) Operator accounting messages.

(5) Background utilities.

(6) Concurrent initiators.

(TOTAL TIMES)

(1) CPU time for job stream.

(2) CPU time for system tasks.

(3) CPU time for utilities.

(4) System I/O wait time.

(5) System idle wait time.

(6) Job run time.

(7) Nonjob time.

(8) Sample time.

(AVERAGE TIME)

(1) Job elapsed time.

(2) Job CPU time.

(3) Initiator between job time.

(4) Time to IPL.

(5) Time between IPL's.

The general step statistics measured by Stanley are:

(1) Number of completed steps.

(2) Average steps per job.

(3) Average steps per hour.

(4) Average elapsed time.

(5) Average step CPU time.

The step statistics by step name are:

(1) Average CPU time.

(2) Number in sample.

(3) Percent of step type.

(4) Step to job CPU time.

Unfortunately, not all of these parameters have meaning in all circumstances. These records do have a striking similarity to the parameters now supplied, if desired, by the IBM System/360. A facility called SMF (system management facility) is now being offered as a part of the IBM system. This facility records information which is considered important in the IBM system. Several classes of information are recorded as individual records of variable length. Each record has a standard header section which includes the time of the record in hundredths of seconds, the date, the model number of the computer, a system ID, and the record type. The records which describe the system are:

(1) IPL record.

(2) Initial I/O configuration.

(3) Vary online-offline (the logical removal or addition of I/O devices as directed by the operator).

(4) Scratch or rename a data set.

(5) Direct access volume record.

(6) Error statistics on tape volumes.

(7) Wait time (written every 10 minutes).

Records written by SMF to describe each job are:

(1) Job initiation record.

(2) Step termination record containing: step initiation time; dispatching priority; completion code; program name; regions requested and used; CPU time.

(3) Job termination record containing: job initiation time; number of steps; completion code; job priority; termination indicator; job CPU time.

(4) Data set activity records: data definition names; data set organization information; data set name; count of accesses; device type information.

(5) Output classes and counts.

The SMF records are continously recorded during system operation if the SMF option is chosen. Several levels of use allow a variety of information records, but the overhead required for the SMF processing has been estimated to be less than 3% for the worst case. The data available can be used as a basis for accounting, so it is probable that the option will be chosen at system generation time for accounting purposes. This data is quite similar to the data previously described, except the SMF records include a time field which identifies the time that the record was written.

One more set of information may also be desired. The three sources above do not contain any information for real time or time-sharing applications. The article by Scherr (22) shows some of the relationships in a time-sharing system, and presents some of the results of the measurements. More detail on this subject is available in Scherr's monograph (21) on the same subject. Six distinct states of a time-sharing system are described:

(1) Dead; no program is waiting to run for the user, and no core-image is being saved for the user. This is the normal starting point.

(2) Command wait; a program is waiting to run but it has not yet run for the first time. It must be loaded before execution may begin.

(3) Working; the program is in execution.

(4) Input wait; the program requires a line of input from the terminal.

(5) Output wait; an output buffer is full and terminal output must empty this buffer until space is available for further action.

(6) Dormant; a special state where no action is possible.

These measurements are based on the basic unit of work in a time-sharing system called the interaction. The usual form of interaction is the sequence of events as follows: the user thinks, types input, waits for a response from the system, reads the response, and begins the process again. The user is in one of two states: 1) the user is waiting for the system to execute the program, or 2) the system is waiting for the user. These two states correspond to "working" and "input wait", respectively, so an interaction may be defined as the activity which occurs between two successive exits from either "working" or "input wait".

In this environment the following measurements were made:

(1) "Think" time of the interaction. The terminal or input wait part of the transaction.

(2) Program sizes.

(3) Processor time per interaction.

(4) Interactions per command.

(5) Response time. The working time of the interaction.

(6) The number of concurrent users.

These measurements correspond to some of the measurements in the batch system. The time-sharing system places the greatest importance on response time. This single measurement is the most requested item in a time-sharing evaluation. Other considerations are necessary for other types of time-sharing systems. If a paging system is studied, for example, the paging algorithm needs study. The frequency of fetching a new page is then an important statistic.

A SUFFICIENT SET OF MEASURABLE PARAMETERS

The considerations above have shown some of the parameters which are used in the area of system performance evaluation. A set of these parameters which is all things to all people would be impossible to formulate. A set of parameters which will satisfy most of the requirements should be much easier to assemble. Some parameters are obvious, but perhaps all parameters should be discussed with their uses.

Starting with the job oriented parameters, the first most obvious parameter is job and step CPU time. These two parameters are nearly redundant except for one difference. The job CPU time should include the time required for step initiation processing, and data set allocation. These parameters are useful in determining the CPU time distribution for an installation. Any evaluations of CPU utilization must have information on the distribution of CPU time being used per job.

Real time, wall clock time, or elapsed time is a measure of the time the job resides in memory. The ratio of real time to CPU time can be considered important in the measurement of I/O blocking and buffering. High real time to CPU time ratios indicate a poor buffering factor for I/O. The real time is also important in determining the number of jobs which can run through the system in a given time period.

Required memory space is important in estimating the number of concurrent jobs which may run. If a hierarchy of memories is available, the measurement should be made for each memory type. Information on the amount of memory actually used can also be used to make estimates of optimizing the jobs being run. Users also tend to be interested in this information. Strategies of running jobs of certain maximum sizes at certain times depend on knowing the memory distribution information.

The name of the program being executed may be a valuable piece of information. The distribution of languages being used can point to desirable development projects. Optimization efforts should be directed toward the highest used programs.

The number of steps in a job reveals how many times the job had to get another program and its associated space. In most systems the initiation of a job step is a non-trivial process which involves interpreting the JCL (job control language), loading a program, allocating both secondary and primary storage and other housekeeping. The number of steps, therefore, determines this effect.

Job priority determines the system action on the program in terms of allocating CPU time to the job. Priority levels allow faster system response for high priority jobs. Job priority may also indicate why a particular job required much less time than another.

The job step completion condition indicates the reason for job completion. If abnormal completion occurs, the data should be analyzed differently. If a large number of users get the same completion code, some action may be called for. Either some form of system problem has shown up (usually certain completion codes indicate these system problems), or the users may need education on the causes of this particular code.

Submittal time studies may give some indications of operational changes to be made. Submittals may come in large batches, which may be the most or the least optimal, depending on the environment. Comparisons might also be made on the sizes of jobs at certain times of the day.

Data set information is valuable in evaluating the usage of the I/O devices. This information may be the most difficult of all to obtain, because it is a dynamic measurement of unpredictable actions. The obvious place to obtain this information is in the I/O supervisor of the system. Additional information on the data set, I/O device, and perhaps the time of the action would often be convenient. As a matter of fact, one of the interesting factors about data set activity may be the distribution with time. Certainly, time distributions on terminal devices can provide the measurements needed for time-sharing evaluation.

System oriented measurements must also consider data set activity. The balance or distribution of the system data sets is important in tuning a particular system for better performance. Certain operations on data sets such as catalogs, procedure libraries, job libraries, and other system data sets need monitoring.

One activity which should produce records is the initial program load (IPL). At this time of system initialization, the system is probably inspected to see what is attached and operational. This initial configuration should be recorded, preferably automatically. After IPL, any changes in the configuration should also be noted.

System time measurements would also be very interesting if available. Several time measurements could be mentioned. System wait time could be defined as the time the CPU is idle; It could also be further broken down into times when no work is available and times when the CPU is waiting for I/O completion. Another measure might be the system CPU time. This is very difficult to define, since much of the time the system is doing its work for some user, if that user could be identified. In most cases, the system is not programmed to find out who to charge this time to, because the search would take more time than the operation required.

A set of parameters which is postulated to be sufficient to describe a system is given below.

(JOB PARAMETERS)

(1) CPU time; by step and by total job.

(2) Real time.

(3) Memory space; by step, broken down into types or hierarchies.

(4) Step program name.

(5) Number of steps.

(6) Job priority.

(7) Step completion condition.

(8) Submittal time.

(9) Data set activity.

(SYSTEM PARAMETERS)

(1) Data set activity.

(2) IPL configuration.

(3) System time measurements.

HOW TO USE THE DATA

After data are collected from the system which is to be evaluated or to be used as a base for evaluation, the data must be properly used. Absolute forms of measurement are of some value, but generally the scientific method is preferable. The classical trilogy of hypothesis, experiment, and modification of hypothesis is a desirable form of evaluation. Absolute evaluations of the data should not be ignored, but only experimentation can prove or disprove a particular hypothesis. This experimentation cannot realistically be performed on the production system, so some form of simulation is desirable.

Evaluation of a system may now be viewed as a three step process where first, data are obtained by a measurement process, second, these data are manipulated and summarized, and third, the observations obtained from the first step are used as input to a system simulation. Each step of this process is an evaluation by itself, but the total process provides a direction for optimization and allows testing prior to commitment to a particular system (hardware or software). This series of operations produces data which must be manipulated so that it has meaning to the user. The processing to provide this meaning is described below.

If the measurements described above are the base for the evaluation, several distributions will be of interest. Distributions of CPU time, real time, and total data set activity should be drawn. Correlations between these variables should be checked for relationships. These correlations may indicate device or channel contention or improper management of resources. Memory space could be presented as a bar graph or histogram since discrete values are involved. The number of steps could also be presented as a histogram.

Presentation of the system-oriented parameters may be viewed in more than one way. The minimum detail required would be a set of totals summarizing the amount of CPU time used by the system, and the total I/O activity by unit address. To determine the unit usage, the individual unit totals are sufficient, but to determine a particular data set order on the unit, data set references are necessary. Exceptionally fine detail would even indicate the proper ordering of information within the data set. This extent of detail would be voluminous and difficult to analyze, so summaries are necessary.

One possibly interesting presentation might be a time-data set graph which would present the data set activity as a function of time. Unit requests could be presented as time dependent entities. If jobs could be associated with each request, an I/O activity - time relation could be shown during the course of a job. Many measurements of these data would show any correlation in these variables. Distributions of I/O activity within a job could then be cited for analysis.

The only process which can predict and measure the changes in the system in terms of throughput or turnaround time without implementation of these changes is simulation. Simulation has traditionally been used as a means of prediction. Many references can be cited as support for simulation. Among these, Katz (11), Seaman and Soucy (25), and Nielsen (16) discuss simulation in some detail.

Katz describes a job generator to produce a simulated job stream for system simulation. His job stream is produced by a simulation language program which might be called a simulated programmer. The output of this program is then used as input to a system simulator. Presumably, this system simulation is variable to represent different conditions. The actual language used is Simscript and a macroscopic simulation is produced for the System/360. Extended events are included such as messenger pick-up and delivery. The simulation produces measurements related to turnaround time, throughput, hardware utilization, software utilization, and queueing processes.

Seaman and Soucy describe a simulation which is much more hardware oriented than software. This simulation is produced in an IBM proprietary language which has many features which are easily adaptable to hardware simulation. A discussion of an operating system sub-model is given to show how such a simulation may be written.

Nielsen's work is in the field of time-sharing computers with page structured memories. The language chosen for this simulation was Fortran because of its nearly universal availability. A study of the IBM/360 model 67 time-sharing system is presented with this paper and several different configurations are tested. In this study, as in the previous two, the job stream used to exercise the system was obtained as a series of approximations.

The concept of using a set of measured data for the input to a simulation model is presented by Cheng (4). This attempts to solve the problem of making too many simplifying assumptions. It also removes the problem of approximating the job stream, since the job stream is a part of the input data. This concept of using a set of measurements or jobtrace as the input is also a part of the Advanced Multiprogramming Analysis Procedure (AMAP) as distributed by IBM (6).

As in all simulations, the simulation must be very carefully formulated. Simulation must be carefully controlled to avoid the problems of incorrect results. In simulation more than anywhere else, incorrect answers may go unrecognized.

Simulations often produce information which is not well understood, and cannot be cross-checked. In the case of the trace information, the cross-checking of the simulation may be achieved with the trace data. Of course, one data point for checking is not really conclusive, but some changes should produce predictable results which can also verify the correctness of the simulation.

A simulation of a computer system allows the iterative process of hypothesis experiment, new hypothesis, more easily than any other scheme. A modular simulation of devices should be possible to allow simple substitution of various components. A trace-driven simulation should provide all of the goals of evaluation previously stated, if it is initially properly designed.

Summary - Part One

Examination of evaluation techniques has shown that many of the traditional methods have logical flaws which may invalidate their conclusions. Since evaluation is a valuable tool for people interested in computer performance, additional study must be devoted to the problem. The key to performance evaluation appears to be a thorough understanding of what systems are and how they operate. To better understand systems, more measurements of their characteristics are necessary.

A proposed set of parameters is presented for consideration in evaluation efforts. The first requirement for system evaluation is the measurement of system requests and actions.

Proper analysis of the measurement data is the next step in system evaluation. Many valuable hints may be discovered with no more than this data. Improving the system performance may be based on these measurements.

Finally, predictive information may only be obtained reliably by the simulation of the system. It must be emphasized that the preferred form of simulation should use as much data as is available. For this reason, a trace of the computer activity is suggested as input to the simulation.

In summary, the system suggested by these preliminary studies is composed of three parts. The first phase is a software monitor system to collect microscopic data to describe individual jobs within the jobstream. The second phase is a data manipulation phase which has two purposes: First, to tabulate and summarize the data produced by the first phase; and second, to organize and prepare the data for a simulation model. The third phase is a simulation of that system which is to be tested. The input data is obtained from the first two phases and is sufficiently detailed to provide a complete representation of the jobstream. The simulation may be modified to change the characteristics of the model system and therefore, will allow testing of hypothesis, followed by further modification according to the test results.

It is clear that measurement of computer systems in both laboratory and production environments is likely to increase in importance. In evaluation and prediction, measurement should become an extremely important part of the total picture.

PART TWO.

THE MEASUREMENT OF DATA

INTRODUCTION

To verify the measurability of the parameters considered sufficient for system evaluation, a series of experiments were performed on the IBM/360 model 65 of the Iowa State University Computation Center. This computer is a typical medium-large scale computer with 512K bytes of high-speed core memory, and 1 megabyte of slow-speed core memory. The normal I/O configuration includes one 2303 drum, two sets of eight 2314 disk drives, eight tape drives, two seven-track and six nine-track. The unit record devices include one 2540 card reader-punch unit, one 2501 card reader, and two 1403 line printers. The three basic types of remote terminal devices are two model 2260 character display cathode ray tube terminals with keyboard, one model 2780 remote card reader-printer and fifteen low-speed typewriter-like terminal ports connected to telephone lines.

This hardware is operated under the OS/360 operating system with the MVT option. The slow-speed memory is used in a memory hierarchy for supplemental processor storage. Some of the system tasks and the time-sharing monitors reside in most of this memory, with a small portion of it reserved for users. Jobs are distributed into classes determined by the memory and time requirements of the jobs and these classes are used as the basis for job scheduling.

The availability of the complete system code and documentation allowed the necessary research and study. Even with this availability, several unforeseen difficulties were encountered. Unfortunately, the measurement program is potentially more dangerous than normal programs in terms of its effects on other jobs and the operating system. Debugging is therefore much more difficult and must be restricted to nonproduction time periods on a special arrangement basis.

The monitor was designed to operate as a series of interrupt-driven asynchronous tasks. The measurements were selected on the basis of the projected uses. The most important activities seemed to be the I/O operations and the CPU time required for each job. These activities were chosen because they represent the limits applied to jobs executing in the computer. Generally, the sum of the time required for I/O operations plus the CPU time determines the real time for the job. I/O operations cause time periods during which other tasks may use the CPU and may show contention on a particular channel. These measurements are described below in some detail. The subsequent analysis and simulations of the system show the general utility of this scheme of evaluation.

COLLECTION OF DATA

Obtaining information on the characteristics of a particular job stream requires interaction with the system which is running the job stream. In that respect, the principle of uncertainty is a factor. Any method of measurement will influence the information being measured. Either the influence must be kept as small as possible or the influence must be known and later removed from the evaluation. If information is available without additional measurements, this influence is minimized. If additional measurement is necessary, the influence must be considered.

Some information is supplied by the operating system as a consequence of the SMF operations. These records are written at all times for all jobs and thus do not abnormally affect the normal job stream. This influence is a part of the accounting system so jobs are always influenced by the SMF recording and allowances for this influence are already part of every job.

Complete information on the I/O activity is more difficult to obtain. The measure of activity is determined by the accesses and replies to and from each unit. Accesses to the unit are handled by a supervisor call (SVC). An SVC produces an interrupt in the normal processing of jobs. The reply from the I/O unit also produces an interrupt. In the

IBM/360 series of computers an interrupt is processed by a swapping of the program status word (PSW). The current PSW is stored in a fixed location in memory. Another fixed location contains a new PSW which is then loaded.

A measurement scheme initiated by each of the system interrupts will gather data in an asynchronous manner, and will obtain all the data. This method may then be described as an interrupt-driven process.

Three specific operations are chosen to represent the I/O activity. The first operation is SVC 0. This SVC is the primary entry point to the input/output supervisor from a problem program. This SVC is also known as execute channel program (EXCP). The second operation is an error EXCP. This SVC is called if an I/O operation must be restarted. The third operation is the I/O interrupt produced by the I/O unit. Certain conditions may cause I/O interrupts without corresponding requests.

To obtain information from these sources the system must branch to the data gathering code. Each of the three operations require slightly different procedures. These procedures are shown in Figure 1 as a flow diagram. An initialization procedure is used to overwrite the system entry point information. Since memory protection is part of the computer hardware, and these addresses are in the protected core, a user supplied SVC must be inserted into the system. The ad-

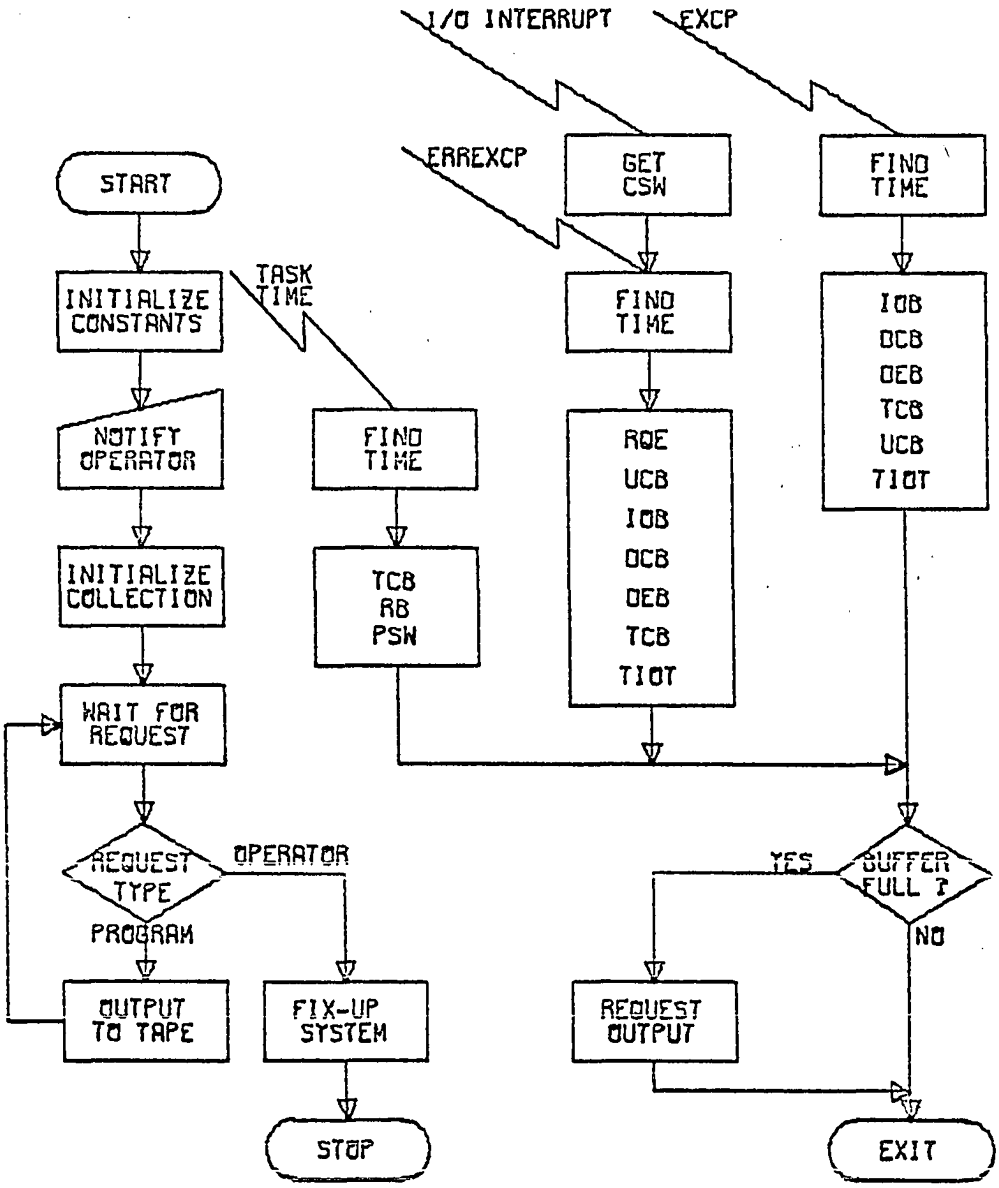


Figure 1. A flow diagram of the collection monitor

dresses needed by the initialization routine are also provided by this SVC. The data collection program then issues a write-to-operator with reply. When the operator is instructed to halt the program, the proper reply is given to the program. This instructs the program to fix-up the system modifications and terminate the measurement.

When a request for an EXCP operation occurs, the system enters a section of code called the first-level interrupt handler (SVC FLIH). The FLIH loads certain important addresses, determines if this SVC is resident or transient, and acts accordingly. In the case of EXCP, a resident routine, an address is loaded as an offset from the beginning of the SVC table (IBMORG). The initialization section has overwritten this address with the address of the EXCP data collection entry point. As a precaution, the registers are stored on entry and reloaded on exit. The starting point for the data to be collected is the address of the input-output block (IOB) which is passed in register one. From this control block other control blocks are located to provide the information required (Figure 2).

An error EXCP is another resident SVC. The data starting point is now the address of a request queue element (RQE). Again addresses may be obtained to locate all of the required information (Figure 3).

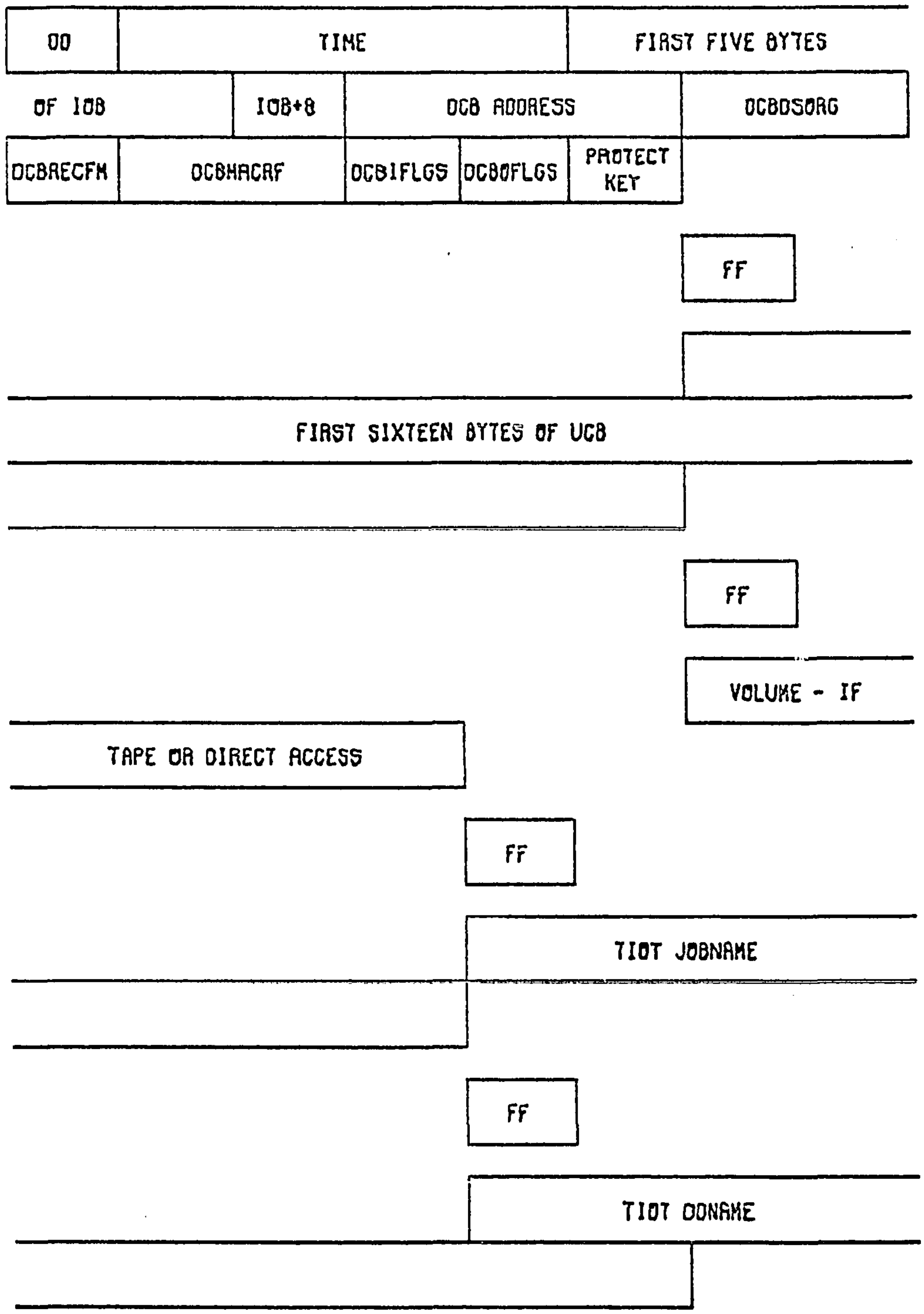


Figure 2. The monitor output record for EXCP

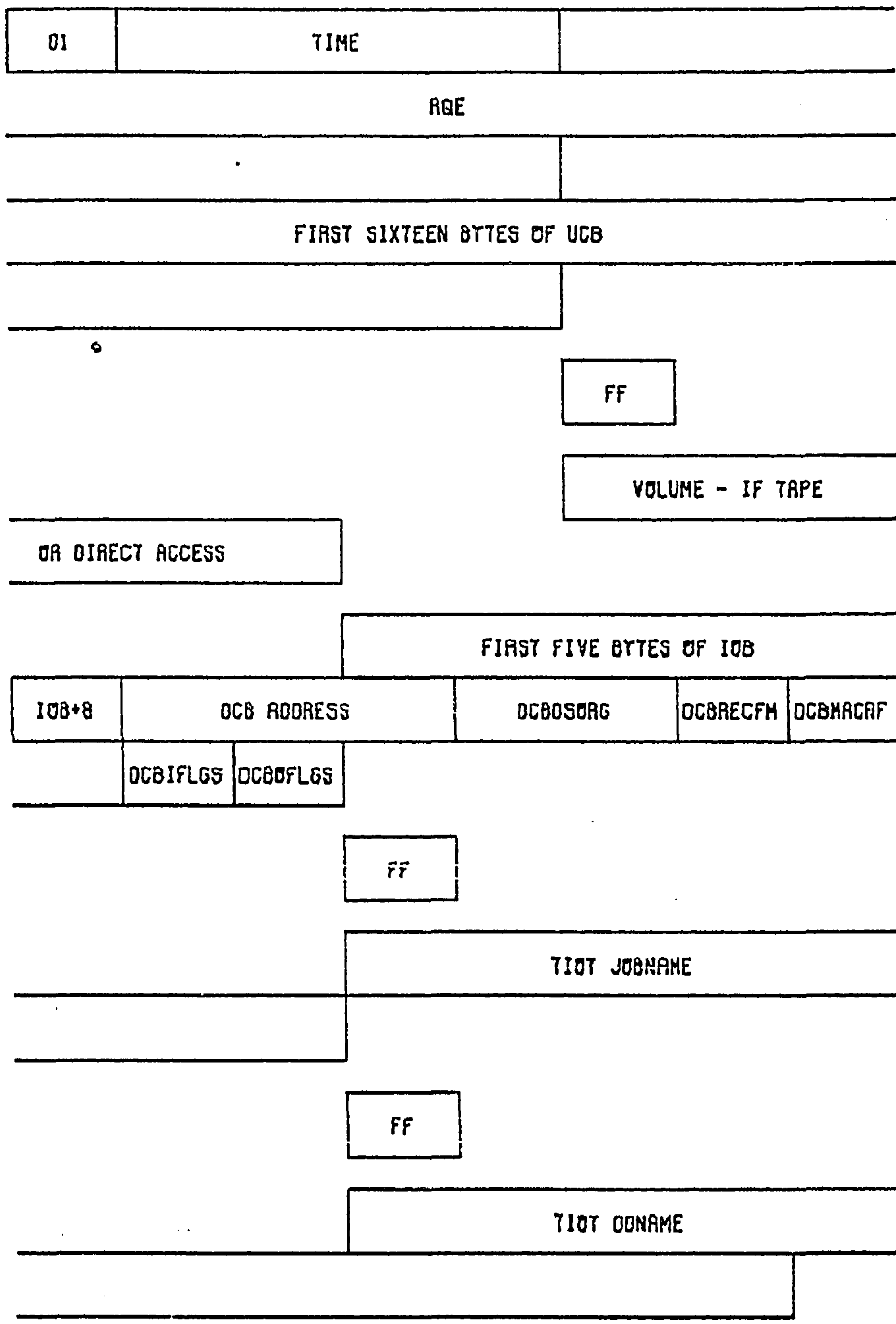


Figure 3. The monitor output record for Error EXCP

The third type of record is produced by an I/O interrupt. The interrupt processing is designed to produce an immediate swap of the program status word (PSW). Certain core locations are reserved for the two types of PSW's. The current PSW at the time of the interrupt is stored into a core location for old PSW's. Another memory location contains the new PSW to be loaded. The PSW contains a mask field, interrupt codes and the current program location. When it is loaded, execution continues from the location specified in the new PSW.

The interrupt handling is started by the hardware PSW swap. The first instruction executed must be a register storage instruction. Addressability may then be established and the remaining registers stored within the monitor's region. Absolute addressability must be used for the first store. The I/O interrupt supervisor uses an area within the first 4096 bytes of memory to save its registers. The location of this save area was obtained during the initialization SVC and filled in at that time. Also at that time, the original system I/O interrupt PSW was stored in the monitor's region.

After the registers are stored, the old I/O interrupt PSW is investigated. The PSW interrupt code contains the I/O device channel and unit addresses. Matching of the unit address with the unit control block (UCB) unit address field then occurs. One of the fields in this UCB is a pointer to the most recent RQE. This RQE is the starting point for the required data. This information is essentially the same as the error EXCP record (Figure 4).

When the data are collected, the registers must be restored to what they were before the interrupt. The system PSW is then loaded to process the interrupt and processing transfers to the I/O interrupt supervisor.

In addition to the I/O data described above, data concerning the CPU activity is valuable. The SMF records only record accumulated time and, therefore, do not provide a distribution of the CPU requirements. The system uses a particular set of code called a dispatcher to assign CPU time to a particular task and to later remove that task from execution. To obtain this task time information, a modification must be made at initialization time to the dispatcher. In this case, the initialization is made by moving selected portions of the code out of the way and bringing in new code to branch to the appropriate location.

Two identical records are written to record the task timing information. The only sources of input for these records are the Task Control Block (TCB) and the timer. Information is recorded to provide the jobname, the time of start or end, and several flags (Figure 5).

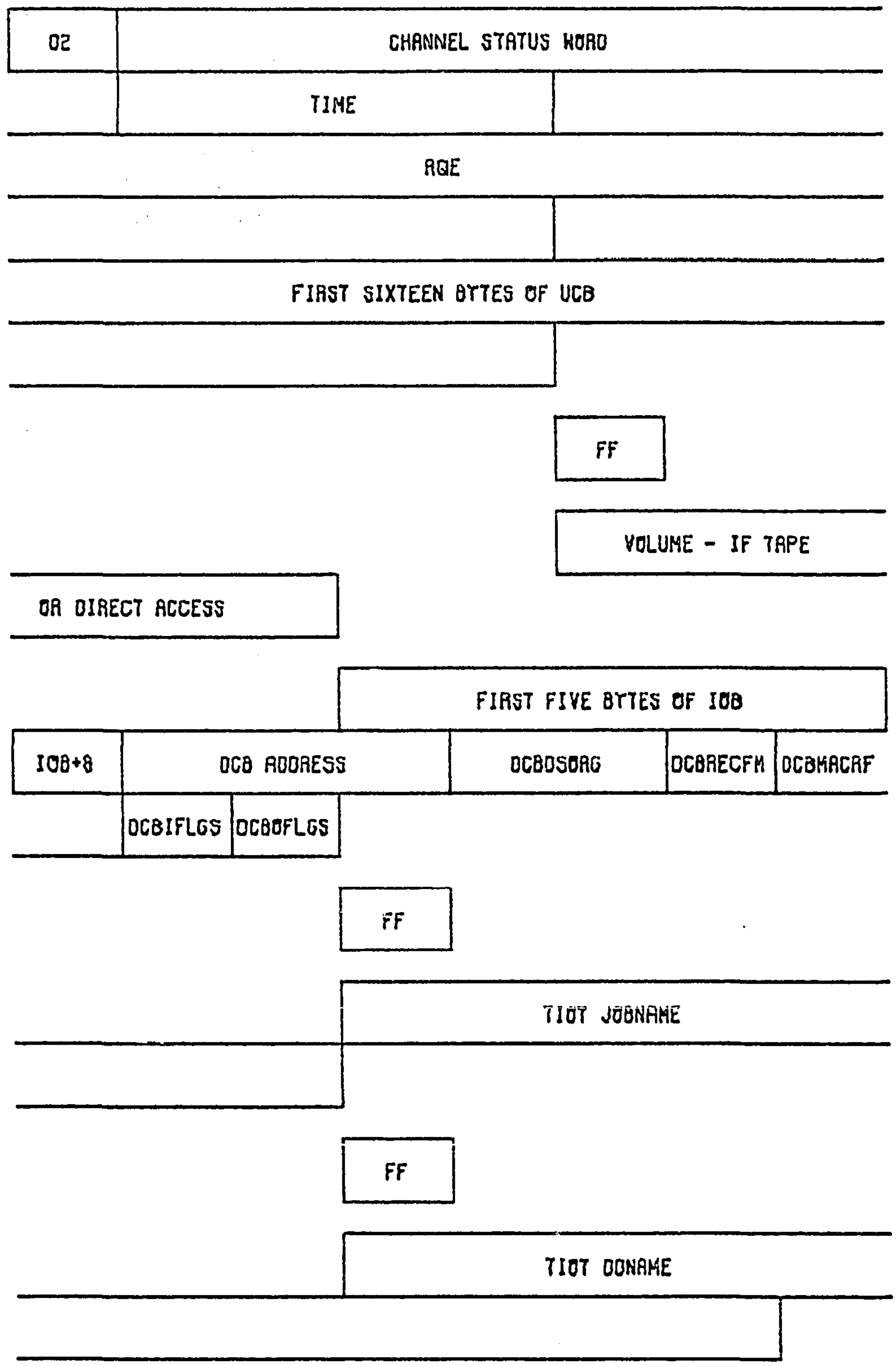


Figure 4. The monitor output for I/O interrupts

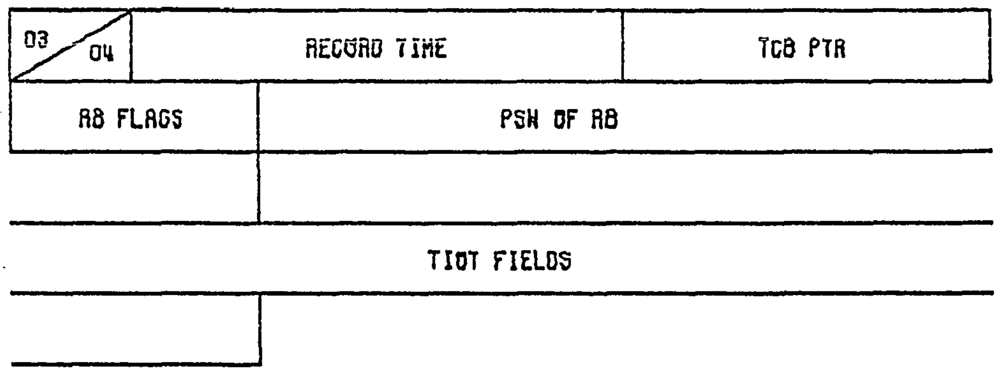


Figure 5. The monitor output for the job dispatching operations

The data collected by the monitor program is intended to provide a detailed description of the activity of all the tasks in the computer. Each record contains at least a time stamp, a protect key, and, if available, a job name. To minimize interference with other tasks, certain information may be only partially computed. For example, the time field requires additional computation based on some fields which are added to the beginning of the records. The proper manipulation of this data produces an actual time of day for timing purposes.

Packing the data is important because of the volume of the data involved. The variable field length approach is used to ensure a minimum size record. Storage of the data on magnetic tape dictates a record size as large as possible. A buffer size of 16,384 bytes was chosen to be written onto magnetic tape. Two buffers are used with an exchanging scheme to switch back and forth between them.

The data provided by the monitor should be sufficient for most analysis requirements. Definition of each process is achieved by the various flags and addresses found in the monitor output. Although all of the data may not be relevant to a paticular study, if all the records are provided, then only one run of the monitor program may be sufficient for many independent analyses.

DATA REDUCTION

After all of the information is recorded on magnetic tape, the next step is to make some sense out of it. The magnitude and nature of the data precludes any manual operations and implies a requirement for efficient programming. In actual fact, two sources of input are available. The monitor program produces the "microscopic" information on I/O activity and CPU cycles. The system has also recorded the "macroscopic" information about each job in its SMF data.

The first operation performed is to separate the required records from the SMF data set and organize these records into individual data set. The first record is read from the monitor output and the time of that record is computed. SMF records occurring before that time are discarded and all data following that time is processed. One SMF record type is not discarded, the IPL devices record. This record is stored into memory for later use. If another IPL devices record appears before the monitor program starts, its data replaces the previous data. If an IPL devices record occurs after the monitor program started, the separation operation is halted and processing continues as if all the records had been processed.

A printed report is begun which will include the time of IPL and statistics about individual jobs. Since several jobs may be executing at any given time, the records in the data sets may be somewhat randomly distributed. Reorganizing the SMF data set at least allows some sequentiality to be apparent in the resultant data sets. Information from a job is produced at job end as defined by the name change in the computer. This information is provided in the form of a printed report showing the job name, the number of times it was dispatched for execution and an identification number which is then used for all future references to that job.

After the SMF data set is split into three data sets, the next routine in the process operates on the monitor program data. The disorder of the information in the monitor data is even more extreme than the SMF data. Data must be recorded within the program storage to enable a logical matching with the various measurements. For this reason, certain variables are input as cards to complete the system definition. Accumulation of totals is done within areas which are set up using the IPL devices record information. Each unit is represented in alphabetical order within this SMF record so a simple transfer is possible.

The output of the program is produced in two forms. The first is the printed report which was started earlier. As all of the information is read, statistics can be produced on various parts of the data. The first information written in this phase is the job summary for the completed jobs as noted above. These records are written in the order in which the jobs complete, as they complete.

The second part of the listing is data to define the time period for the measurement, the total number of records, and the distribution of the records. This part is essentially a statistical description of the records themselves. This information is provided to give an indication of the statistical validity of the data and to date the data so that the configuration might be remembered.

The third part of the printed listing is an I/O activity listing by unit number. Each unit is listed and the total number of each type of record is listed behind it. This listing may be used to show which units are being used the most. Additional information is recorded on each direct access I/O operation which provides the address of the operation on the device. This information may be used to produce a histogram of direct access device addresses and the distance traveled between accesses.

Finally, the printed listing contains the job information for the jobs which are unfinished. This listing includes all of the jobs which are permanent in the system such as writers, readers, and teleprocessing programs. Also in this list are the system requests and tasks, and the system wait time. The monitor program will also appear in this listing.

A second form of information is produced at the same time as the printed listing is created. Some graphical means of presenting the computer activity is considered valuable, since the magnitude of the data is so great. The form chosen is to plot line segments for each period of time that a particular resource is in use. This form allows a pictorial representation of the overlap of I/O activity and CPU activity. In practice, the CPU activity is broken up into jobs and the lines are labeled with the job number. This scheme allows a potential investigation of job activity within that job.

The problem with graphing time periods is the small magnitude of the basic time unit. Since each time unit is approximately 0.016 seconds, many time units are contained within a short period of time. If one second is chosen to be represented by 0.6 inches, then the minimum time period (0.016 seconds) is nearly the same as the minimum increment on the available plotter (0.01 inches). To investigate a long time period would require a very long graph. The ability to look at selected portions of the graph is necessary to overcome this problem.

A problem also exists with the resources axis of the graph. A large number of I/O devices may cause the graph to extend upward a considerable distance. In this case, the graph is split into multiple graphs which may be placed one above the other. Each graph is a complete graph with all axes labeled.

Trace Record Production

After the information described above is produced, the last phase of the program produces a set of records which may be called a jobstream trace. This trace information is produced as three distinct record types(Figure 6). The first record is a job record, which defines overall job information such as the time it was read into the system, the number of steps, the priority, and output information. The job record is a variable length record with an ordinary data set organization.

The job record includes a pointer to the first step record for the job. Both the step records and the I/O records are contained in a common data set because they require the same organization and are the same length. Information in the step record includes both the core storage requested and used, the CPU time, the priority, and the time of step initiation and termination. Pointers are included to obtain both the first of the I/O records and the next step record.

Each I/O record is an indication of seven I/O actions. Each record has a one byte unit number followed by three

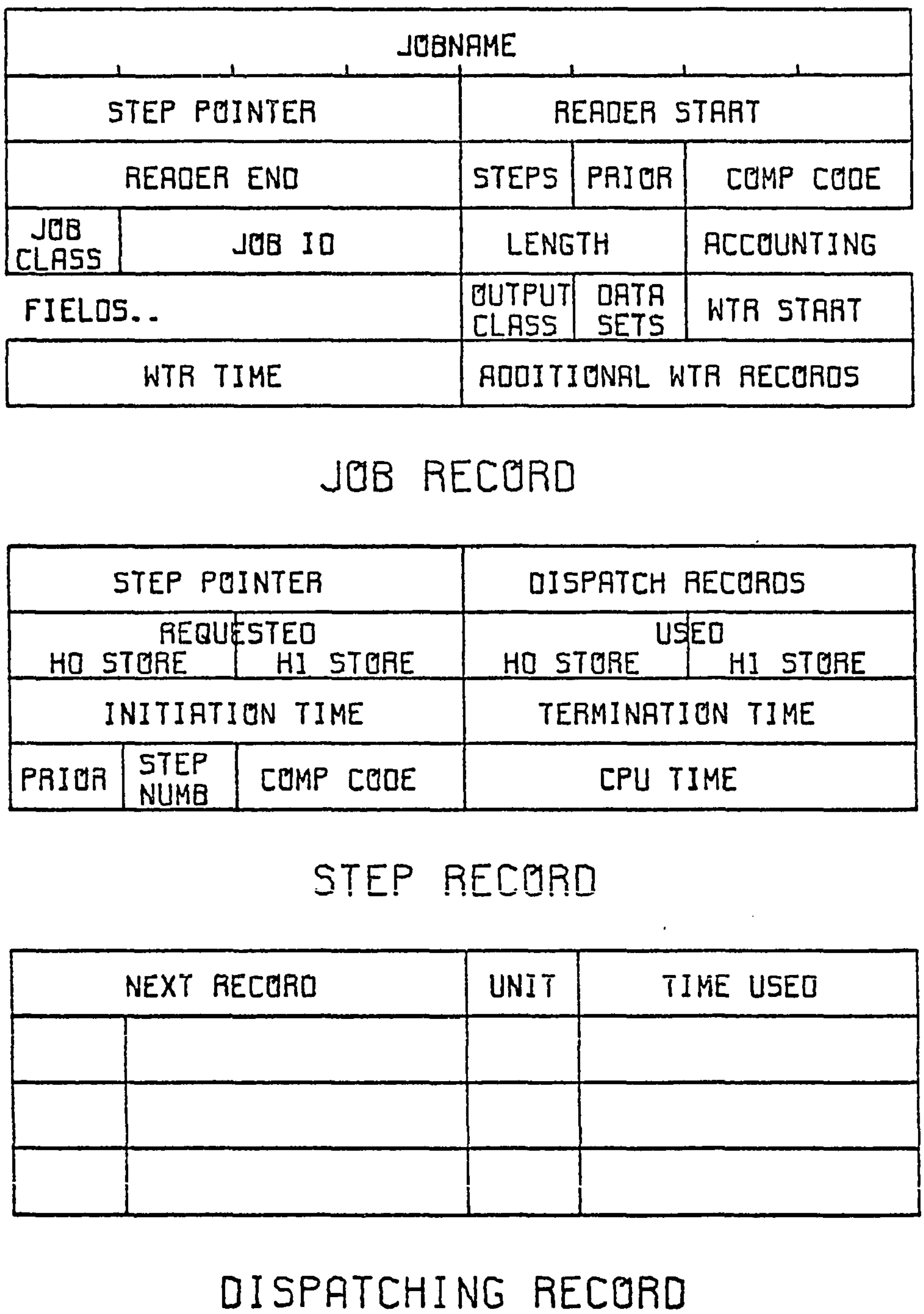


Figure 6. The jobtrace records

bytes of use time. Each record also has an address of the next I/O record.

The information represented by these records is believed to be in excess of the requirements for a system simulation. Few, if any, of the previously cited sources have had as much data to work with in their simulations. Additional information is provided so that the simulation may be as simple or as complex as is desired. To provide the jobs in the same order as they were presented to the system, the data set containing the job records may be sorted. The pointers to the step records will still be valid, so this is an acceptable modification.

DATA ANALYSIS OUTPUT

The output of the data reduction program is interesting for an insight into the working of the system. The records and plots produced show certain immediate information for application in system performance improvement. Figure 7 shows the highlights of one data reduction program output.

The first section of the output shows statistics on the jobs which have run to completion during the monitored time period. The first column of the data provides the users jobname. This jobname has two additional names given to it. First, since the job is started by a particular initiator, that initiator name is given as an "alias" for the job. The second identification is the ID number assigned to the job. This number is added to provide an easy way of referencing each job. The remainder of the information is an indicator of the CPU activity of the job. The total time divided by the number of dispatches of that job is an indication of the time between interrupts for that job. The system has a facility called time-slicing which forces the job to release the CPU so that another job may execute. This time-slicing interval may be selected using this data as a guide.

The second section of the output shows some of the data concerned with the monitor operation. A total number of records processed, and the totals for each type of record show the magnitude of the process. The elapsed time for the

JOB DISPATCHING STATISTICS

Name	Alias	Dispatch	Total Time	ID
BATCH02	B	173	11.18	5
A369IUP4	D	207	8.28	9
A421F5	B	105	7.93	10
C383SMHP	D	63	2.54	13
B2872222	D	124	19.41	16
B2873333	D	70	3.29	17
A233T101	B	227	7.71	14
A273SPLT	C	53	2.39	20
C288BG2	D	233	13.03	18
C393MTHO	C	247	10.08	24
DEKLIST9	C	78	2.94	24
C449FORT	B	379	26.64	19
A401F02	D	287	14.68	22
A254H	C	353	36.46	25
DUANE09	E	964	1:37.76	7
D204SELI	B	320	27.31	26
C346NAAM	D	243	13961	27
A409D9	B	99	5.84	30
C346NAME	D	214	13.23	31
A435ADA	B	255	14.58	32
B229DIFF	B	73	3.04	34
C241BQ	B	66	2.78	35
T406SMF	C	648	37.09	28
A345STAT	D	586	1:06.82	33
C428V77	E	467	49.06	29
D342	C	93	4.99	37
A282TRAN	B	98	3.36	36
A335PLT	C	144	5.98	40
C384AAEF	D	262	9.39	38
C235CJRZ	C	61	2.29	42

Figure 7. Output information from the data reduction program

STATISTICS ANALYSIS RETURN

Total Number of Records Processed 589,345

Time Information

Collection Date	71:232
First Record	11:47:10.01
Last Record	12:18:45.06
Elapsed Time	31:35.05

Distribution of Record Types

Type 0	42392
Type 1	471
Type 2	108682
Type 3	218900
Type 4	218900
Zero time	143463

Figure 7. Output information from the data reduction program (continued)

Distribution of Unit Activity

Name	Type 0	Type 1	Type 2
00B	246	2	501
00C	11541	18	21128
00D	2627	8	4562
00E	2828	18	3118
010	1932	16	2137
01F	201	0	240
020	603	0	680
021	0	0	0
130	319	50	17459
131	199	1	391
132	24	0	34
133	392	10	706
134	286	10	549
135	298	4	526
136	0	0	0
137	98	0	147
280	0	0	2
281	1405	13	1767
282	197	6	288
283	3784	25	3904
297	1421	0	11255
330	1455	57	19047
331	0	0	0
332	1106	42	1525
333	2681	54	5597
334	2288	38	4360
335	2019	65	3812
336	76	0	293
337	0	0	0
380	14	0	36
381	1	0	0
382	3887	32	4088
383	184	2	216

Figure 7. Output information from the data reduction program (continued)

Unfinished Job Summary

Name	Alias	Dispatch	Total Time	ID
SYSTWAIT		4527	13:06.56	0
SYSTREQ		2319	35.13	1
PRT2		1002	47.13	2
PRT1		1264	1:02.54	3
MASTER		392	9.13	4
A335PROG	B	201	6.98	41
OPER		439	29.53	6
C206TOB	E	704	1:11.61	39
IOSTAT		325	18.49	8
C369BGK3	D	212	8.09	43
RDR1		2433	1:49.32	11
PUN1		451	16.78	12
CPS		108	6.44	15
B383CMPL	C	203	34.43	45
MOUNT		27	0.94	23
RDR2		69	2.64	44

Figure 7. Output information from the data reduction program (continued)

collection period is also given. From these numbers, it is apparent that a large amount of activity is present in the computer system. Dividing the total time by the number of records provides a measure of the average time between records. This time period is less than three milliseconds. If only the dispatching records are considered, the interval is still something on the order of nine milliseconds.

The small size of the average time period may also be seen in the "zero time" count. This field represents the number of time periods which were less than one timer unit (0.016 seconds) in duration. As can be seen, nearly two-thirds of the activity was within this category.

The third section is devoted to the I/O unit activity. I/O unit activity is important in determining channel splits and device overloading. These areas are considered when the I/O operations are the limiting factor on a system's performance. These records may be sufficient to provide some information relevant to system performance, but a better guide would be the actual address of the operation. For this reason, a data set is produced which contains the address of each I/O interrupt and both the volume identification and the unit number. This data may then be tabulated into some usable form. This data is then useful to position the data sets on these volumes.

A careful examination of the records will reveal that many more I/O interrupts occur than EXCP operations. Since the I/O interrupt operation is a hardware action, it is assumed to be correct. One of the sources of extreme difference is the use of data transfer methods which do not rely on the system EXCP method. This may be seen in the data for the system volumes 130, 297, and 330. These three volumes contain the principal data sets for the system. Since the actions on these data sets are controlled by the system, EXCP may be bypassed and no records will be written for EXCP to these data sets. Therefore, the only reliable indicator of activity appears to be the I/O interrupt records.

The fourth section of output tabulates the unfinished job information. Included among the unfinished jobs are records which tell how much time various system tasks require. The first data item in this list is an indication of the system wait time during the interval. This information combined with the time period of the monitor, shows the percentage of CPU utilization. In this time period, the CPU utilization was approximately 59%, but this run was during a slack time for computer usage.

Notice that the time required for the monitor program is also listed in this output. In this case, the monitor required slightly less than 1% of the time period. However, it must be remembered that the monitor also requires at least one tape unit and causes some interference with channel activity. One other situation occurs while the monitor is running which may influence these numbers. If a monitor buffer gets full before the previous buffer has been written, data may be overwritten. This could happen if a tape error is detected and automatic error correction actions are applied. To prevent disastrous results, a feature of the operating system is used to effectively lock out all other tasks from execution. This occurs fairly regularly in the time period and no measure of this influence is shown.

The second form of output is the plot produced from these records (Figure 8). The plot is provided as a pair of sections which may be put together. The plot is labeled with I/O unit numbers and a concurrent job number. This job number has no relation to the job identification number, however, the job identification number is used to label each line on the CPU requests section of the plot. Each action is theoretically shown by a line segment extending from the beginning of the action to the end. In many cases, however, the action has a zero time length. In these cases, only a dot will be plotted. It was also found necessary to eliminate multiple dots on the same time coordinate. This is an indication of the number of actions which occur between timer intervals. In the I/O action section, the intent was to measure each I/O action from the EXCP record until the interrupt

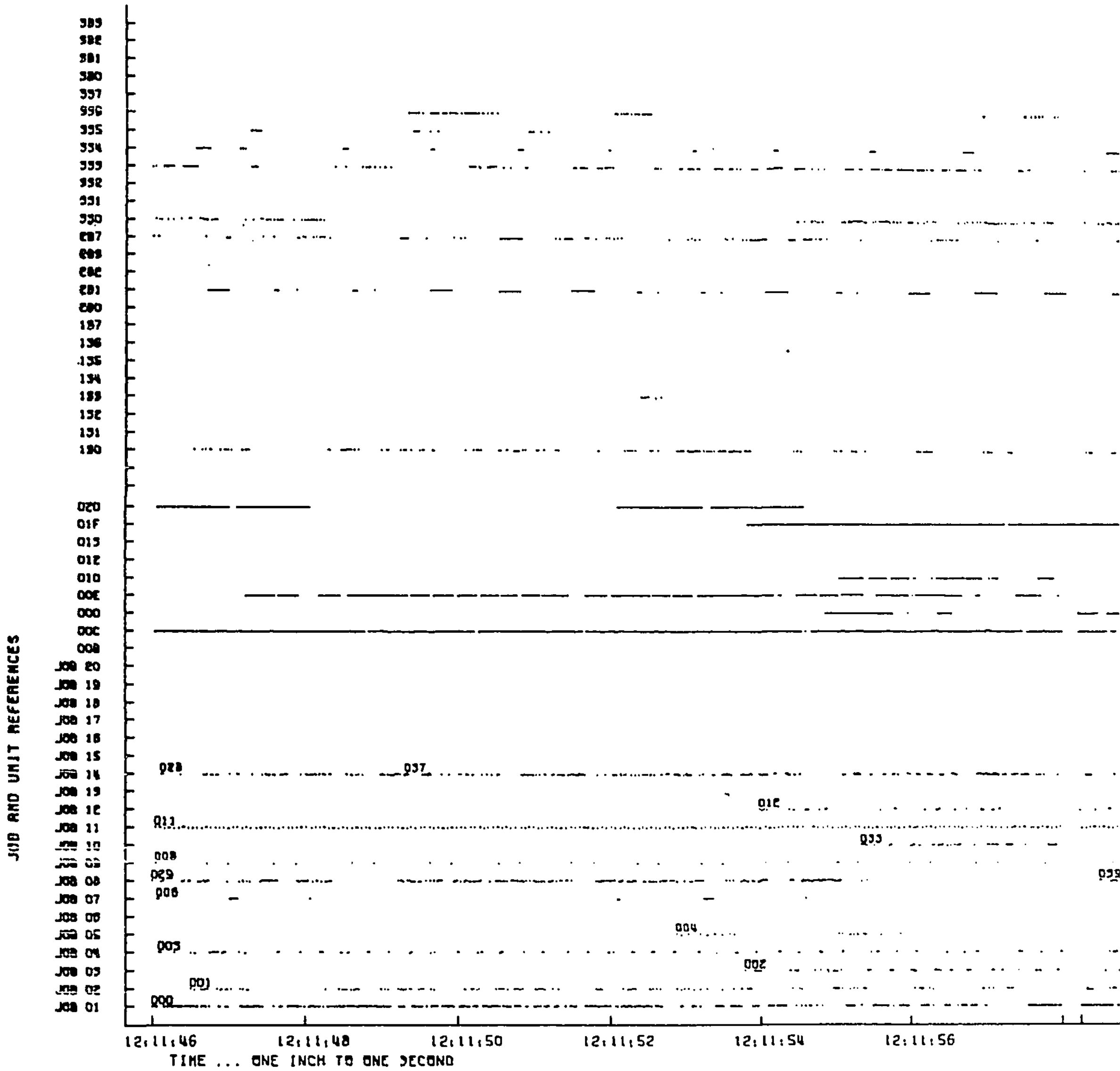
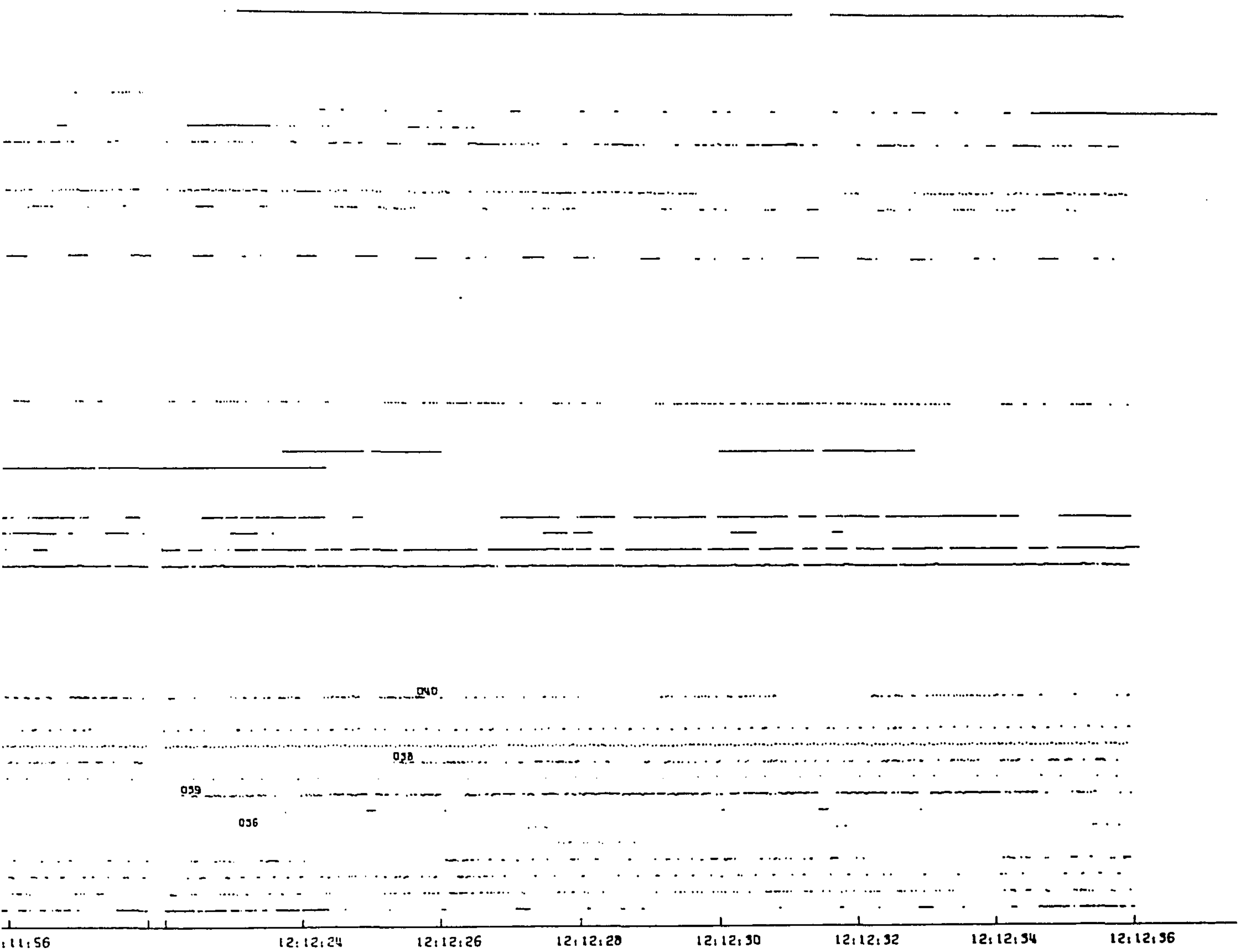


Figure 8. Activity plots of the collected data


040
038
039
036
:11:56
12:12:24
12:12:26
12:12:28
12:12:30
12:12:32
12:12:34
12:12:36

a

record. As shown above, the records do not match, so this was only partially successful. In the cases where only an I/O interrupt was recorded, the record is marked by a single dot. Although it may not be immediately apparent to the eye, many of the "lines" on the plot are really a series of closely spaced dots. This is especially true in the CPU activity and I/O unit 297.

Data Analysis Discussion

To a system programmer, the output from the data analysis program can be very interesting. The data which is tabulated and plotted may show information which will aid in system performance optimization. The printed tables shown in Figure 7 may guide the system programmer in this effort. First, the information provides the measurements necessary to calculate the average CPU time per dispatch. This number should be used to guide the selection of the time-slicing parameters. The time-slice period should be large enough to satisfy 90% of the job requests. This is supposed to allow most of the jobs to progress far enough to start an I/O operation before it is interrupted. From the data presented here, this number might be selected at 80 milliseconds. Previous selections set this number at 200 milliseconds.

The next information of interest is contained in the unfinished job summary. No other measurement scheme allows a measurement of the system tasks. These tasks include the reader and writer programs and other system control tasks for such things as modifying jobs from class to class, displaying job queues and cancelling jobs. Also included in this summary are the teleprocessing jobs. From the table, the readers and writers accumulated 3 minutes, 55.77 seconds out of 31 minutes, or about 10% of the time period. This seems to be a reasonable amount of time for the spooling operation. As was previously noted, this data was collected during a slack day, so the system wait time (SYSWAIT) is quite high. In fact, over this time period, the CPU has less than 59% utilization. The two tasks which represent system activity are SYSTREQ and MASTER. The accumulated total time of these two tasks is 44.26 seconds which is less than 0.5% of the time period.

The two teleprocessing tasks, OPER and CPS are listed with the unfinished jobs. OPER is really a specialized task for operator control purposes, but CPS is a user oriented system. Together, these two used about 0.2% of the time. Of course, very little activity (abnormally low) was recorded on CPS.

The last bit of information on this sheet is the time required for the statistics monitor IOSTAT. This quantity is less than 0.1% but it must be remembered that most of the monitor's time is spent under some other task's time. In fact, this time for IOSTAT may be down within the timer resolution.

Unit activity is the next important measurement. This information may be used to locate critical data sets for optimum performance. The critical data sets exist on the units named 130, 297, and 330. From the numbers recorded, these units appear to be quite evenly accessed.

The plotted information is more interesting as a method of viewing the system activity rather than having any intrinsic value in a detailed analysis. The plots in Figures 8 and 9 are typical of two time periods in the data. Figure 8 shows an active period in the computer. Note that the system wait time (job id number 000) is nearly a solid line to start with and toward the end of the plot the activity becomes much less solid.

Other tasks of interest might be OPER (006) and IOSTAT (008). The pattern of OPER is determined by the option selected and the automatic update time selected by the operator. In this case, the pattern is composed of three little bits of time followed by a four second wait. The first bit is a signal from the timer followed by a read of the operator display. After the read is complete, a little bit of time is required to format the next display, and then the write operation is initiated. The last little bit is used to set up

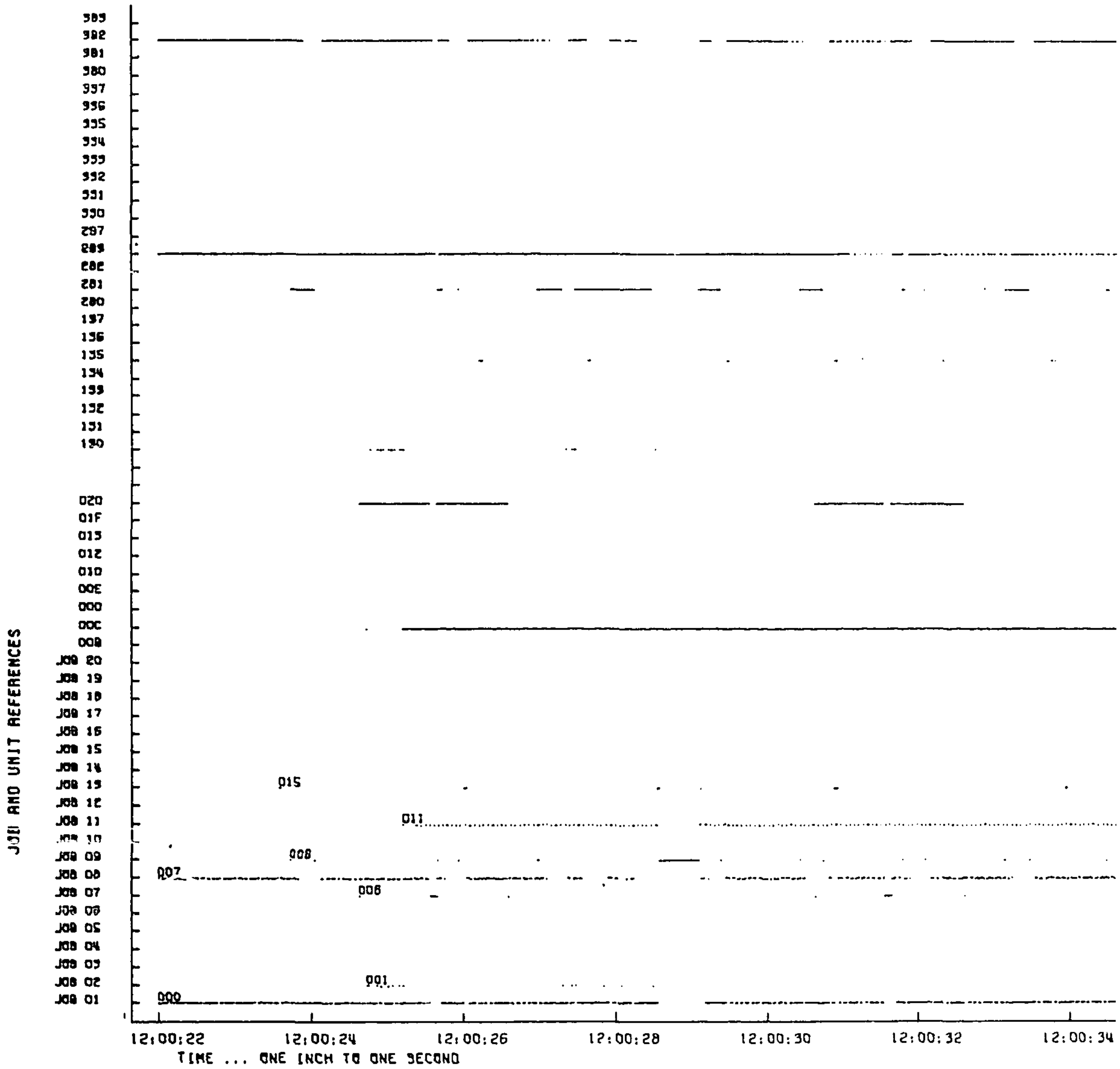
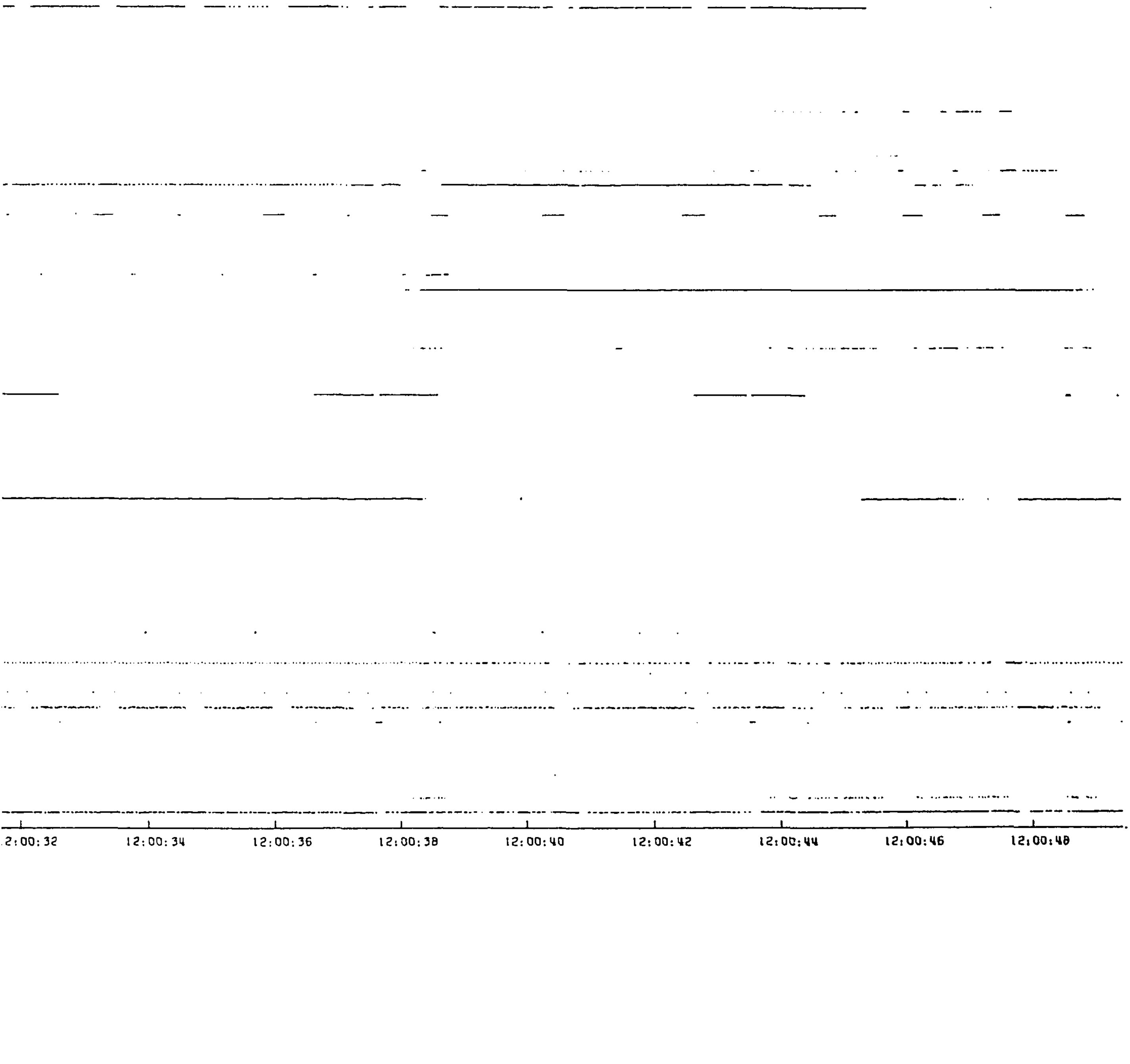


Figure 9. An activity plot for data analysis


.2:00:32
12:00:34
12:00:36
12:00:38
12:00:40
12:00:42
12:00:44
12:00:46
12:00:48

the timer for an interrupt after a specified period of time (4 sec.). The read and write operations may be seen in the line marked 020.

The statistics monitor (008) is a regular pattern of two dots followed by a slightly longer period of wait. The write operation is directed to unit 281. These writes may be closely correlated with the CPU requirements. This regular pattern may be noticed in Figure 9 as well. In Figure 9, the only other job running is 007. This job requires a tape mounted on unit 283 and the effect of channel contention may be easily seen because the CPU time for job 007 is interrupted during the time that unit 281 is used. Since the monitor runs at a higher priority than any other job, its requests are serviced before other job requests. Therefore, the execution of the job using unit 283 is interrupted until the write is complete.

Figure 9 also shows the start up action for reading jobs into the system. As a job is read in, distinct phases may be noted. First, the cards are presented to the physical card reader. After a bit of checking, the reader begins spooling the cards onto a disk unit; in this case the unit is 135. At the end of a step in the job, the step and its control blocks are entered into the job queue on unit 297. These phases may be clearly seen in Figure 9.

PART THREE.

THE SIMULATION OF AN OPERATING SYSTEM

TRACE-DRIVEN SYSTEM SIMULATION

Once the trace information is available, the next step is the modeling of the operating system. An immediate decision must be made to determine the simulation language to be used. Since any possible language must be, first of all, available, only three simulation languages were considered. These are GPSS, SIMSCRIPT, and SOL. Of the three, the only one which is truly a production system at this installation is GPSS. Additional languages are available from external sources, but their capabilities are either unknown or their costs are prohibitive.

Comparing the available features in the three compilers is the next step in the decision. An article by D.E. Knuth and J.L. McNeley is used as the definition of the SOL language(12). Most of the features of SOL are well designed for a simulation language. Inadequate arithmetic capability is a serious problem in SOL, as is the omission of a list or queue creation facility. This ability is important for the simulation of an operating system. The only other questions concerning SOL are the problems of I/O and storage simulation. The I/O statements available seem to be quite powerful. However, there is no way to conveniently add user routines to handle the jobtrace information. The description of SOL does not specify the exact form of the storage requests. For true system simulation, the storage must be a discrete element storage. This is an unknown in the SOL system. Unfortunately, SOL is implemented as an interpreter so extensive simulations would be costly.

The second language to be investigated is SIMSCRIPT. Important problems in the simulation include queueing, I/O, storage, and communication between parts of the model. As in SOL, user controlled queueing does not seem to exist in SIMSCRIPT. In addition, all waits for facilities are done in a first-in, first-out (FIFO) list form. A system would in all probability use a priority queueing for the internal lists in the system. Note that priority queueing is a more general form, since all entries with the same priority are handled as a FIFO list. Since SIMSCRIPT is implemented as a FORTRAN superset, FORTRAN I/O may be used as well as an extensive set of special SIMSCRIPT I/O instructions. Storage considerations are again unknown, but the communication problem does not appear to be solved.

GPSS is the remaining language to be investigated. The definition of GPSS which was used for evaluation is found in the GPSS user's manual (7). Most of the necessary properties are available in GPSS with a few critical exceptions. The most important problem is the storage operations. Storage in GPSS is viewed as a continuous entity with no holes or spaces. Fragmentation cannot occur in GPSS storages. The second problem, the communication between transactions, is, at the least, very difficult in GPSS.

After the above evaluations, it is seen that none of these languages are adequate for the simulation required. The necessary language appears to be some kind of cross between GPSS and SOL. This is the form of the BOSS (Basic Operating System Simulator) language which was developed to fulfill the requirements of this dissertation. This language is specifically designed for the simulation of operating systems. Although it is a combination of GPSS and SOL in its functions, statements such as assignment statements and I/O statements are quite similar to PL/1.

Why Create a New Simulation Language ?

> "Before a designer sets out to develop a new simulation language, he should seriously consider whether a new language is really necessary. A new language, in itself, is not sufficient justification for existence; some demonstration of the usefulness of new features is necessary. Often user complaints about existing languages are not with the language per se but with certain features of the implementation: lack of documentation, lack of training aids, difficulties in incorporating the package into a computer center's monitor system, lack of adequate debugging facilities, and so on."

As the quotation above (31) states, simulation languages should not be created for the pleasure of the designer. The only apparent justifications for a new language are the special features which are required. A careful analysis and comparison must precede the design and implementation of a new simulation language. Sometimes the resulting language may be a special purpose language which may be difficult to compare with a general purpose system. Nevertheless, the known languages should be investigated to determine if the required features are available.

The design of an operating system places some rather unique requirements on a simulation language. The obvious requirements of time advance, reserving resources for a particular job, and controlling an orderly progression of jobs through a system require certain capabilities. One of the first desirable features is some form of input to describe the job stream. The particular form of input is somewhat dependent upon the uses to be made of the data. Some generalized form of input can be used for several purposes, but a specialized input for only jobstream information might also be considered.

Communication between separate transactions or tasks in the simulated system is a very desirable feature. A typical multiprogramming system is usually based on separate tasks which must pass information to other tasks in the system. An example of this communication occurs in the spooling of data onto secondary storage for later execution. The input program is responsible for assemblying all the necessary information into a set of pointers, and then placing that set of information into a list where it waits until it may be executed. Some other task (sometimes called an initiator-terminator) is required to begin the execution of the job. At the completion of the job, the output (usually printed or punched) must be entered into a list for another task to transfer the output from secondary storage to the physical output device. The interdependence of these tasks requires a signal from one task to start the next task. A "mailbox" technique could be used where the tasks keep looking for work at regular intervals. A quicker technique can be used if the tasks have some form of "shoulder-tap" communication. The information may then be obtained as soon as it is available.

A third element of operating system simulation involves the allocation of resources. Both partial and total allocations are used in operating systems. Partial allocation is typical for resources such as primary and secondary storages. An important restriction on partial allocation is the discrete nature of these devices. Allocation must only occur on discrete boundaries and may not be moved from its original position. This leads to problems of fragmentation, where free space may not be contiguous. This means that although the total free space might be sufficient to satisfy a request, it is not in a single area. This fragmentation might be one of the problem areas to be studied.

THE BASIC OPERATING SYSTEM SIMULATOR

A special purpose operating system simulator should be designed to aid the system simulation study as much as possible. The use of terms which are either common to the system simulation programmer or descriptive in nature is an aid to the writing of the simulation. In addition, artificial constraints should be eliminated as much as possible. The form of the language must be easy to remember and might reasonably be based on one of the common computer languages (PL/1, FORTRAN, ALGOL, etc.). Typically, system programmers want more facilities and capabilities than are available, so easy expansion or addition should be provided.

The form of the Basic Operating System Simulator (BOSS) is similar to PL/1. The statement structure has an optional label, a statement identifier, and a trailing semi-colon. The label consists of an identifier followed by a colon to delimit the label from the statement. The statements are free format and may occur anywhere within the card boundaries. Additionally, the statements may be placed on the same card as other statements. Comment statements are allowed which may have any character except a semi-colon in them.

Some of the special features of the BOSS system are concerned with memory management within the model. The memory management keywords ALLOCATE and FREE handle all reserving of memory space. The memory space is defined in discrete increments and maintained as a discrete storage area. The ALLOCATE feature also has a conditional entry feature which allows continuation of the program even though the allocation is not possible.

Another special feature allows the various transactions within the simulation to communicate with each other. This feature is similar to the WAIT-POST facility in IBM/360. The commands are WAIT ON(list) and SIGNAL which allow the simulation to wait until the event is completed.

Input/Output is allowed through a statement structure almost identical to PL/1. Data is processed as a stream of characters from which the requested areas are determined. Two forms of data transfer are allowed, a free format process and a programmer controlled format. In addition, a standard statistical output is generated at the end of the simulation run which may also be obtained at specific intervals (snapshots).

User defined lists or queues are possible with ENQUEUE and DEQUEUE capabilities. The ENQUEUE process also allows an event completion signal to notify other transactions that something has been placed into the queue. The queues are organized as first in-first out within a priority class.

External subroutines may be incorporated into the simulation by using an EXECUTE statement. Either BOSS subprocesses or assembler subroutines may be called in this way. This allows certain standard routines to be written once and used by several simulations. In addition, several standard functions are provided to compute observations from standard probability functions.

The BOSS program is executed as a standard language processor in the IBM system. The program is able to produce object modules and object decks, or use these as subroutines. Checkpoint data may be written at regular intervals for restart of the simulation. The instruction structures are described below.

The implementation of BOSS was achieved with a modified form of the META PI compiler-compiler (17,18). This technique provided the syntactical and some of the semantic operations with a minimum of work. The entire language definition in the META PI language is given in Appendix F. As might be expected, certain features have been added to META PI to accommodate the simulation language definition.

Variable Types

The BOSS simulation language allows several different variable types for simulation purposes. These variables are used to represent values, actions, or physical items necessary for system simulation. Variable types are usually determined by the contextual use of the variable. In some cases, variables must be defined to assign certain characteristics such as length or dimension. Almost all variable types may be arrays if declared as such. Arrays may be n-dimensional with bounds set as required. Both upper and lower bounds may be specified when the array is declared. This is achieved by specifying a bounds pair, two numbers, separated by a colon. The first of these numbers will be used as the lower bound, and the second number will be the upper bound. If only one number is specified, the lower bound defaults to zero, and the number specified is used as the upper bound.

An important variable for simulation is the transaction parameter. This variable type is associated with the current transaction, and remains with that transaction for the duration of its life. Since these parameters are unique to a particular transaction, they may be used to represent information unique to that transaction. Two forms of the parameter may be used. The first form is simply the letter P followed by a number. This form is used to represent an integer parameter value and it will be used as an integer. The second form is the letter pair PF followed by a number. The variable is then used as a floating point number. The postfix number for these parameters is chosen from the numbers zero to seventeen. Both floating point and integer parameters are stored in the same area, so numbers may not be used for both integers and floating point at the same time.

Three variable types are used to store values for later reference. These types are INTEGER, FLOAT, and BOOLEAN. Integer variables are used only for values which do not have a fractional part. These variables are common in simulation for counting and quantity recording. Integer is the type assigned for ordinary assignment statements. FLOAT is used for those applications which must have fractional parts. Among these applications is the measurement of time for a process. Statistical distributions provided within the language usually return floating point data. BOOLEAN variables may be used to set switch information for later testing. The common Boolean connectives may be used to form Boolean expressions.

Two types of variables are used to represent storage type entities. These are STORAGE and QUEUE. A STORAGE entity is used to represent the physical act of storing data or reserving space for data. A transaction must request space for storage from a particular storage unit. After using that space it must be released or freed so that another transaction may use the same space. Storage is a discrete entity and discrete requests must be made. Blocks or units of storage must be specified, so the variable must be declared. Fragmentation is possible because freed space need not be adjacent to the current free space.

QUEUE variables are used to produce waiting lists of transactions. Queues may have a maximum capacity and may be either ordered by priority or first-in, first-out. The number of transactions waiting in the queue may be limited by a declaration. When an entry is removed from the list, the entry removed is the top-most entry in the list. Transaction removal may only occur if an entry exists in the queue.

FACILITY variables are used to represent items which may only handle one transaction at a time. These devices may be considered valuable resources because the other transactions in the system may be competing for its use. The SEIZE and RELEASE commands are used to service facilities. If a transaction finds another transaction has already pre-empted the use af the facility, the current transaction is placed on a waiting list. This waiting list is ordered according to the priority of the transactions in it.

The EVENT variable is used to record the occurrence of some action. Many situations require a coordination effort between several transactions. Event variables record information which is used to determine if a transaction has completed the event, cleared the event, or if another transaction is waiting for the event to be completed.

SIMULATING AN IBM/360 OS SYSTEM

The simulation of an operating system requires a great deal of investigation. In fact, the simulation designer should be as familar with the workings of the operating system as he is with applications programs. Many questions about a system's operation must be answered before it can be simulated. This minute investigation of the system often proves as useful to the designer as are the final simulation results in understanding the system operation.

OS is best modeled in three parts. The first part of the model is the reader procedure. The reader procedure is used to bring the job into the system for execution. The reader is the software entity which translates JCL to control block information and spools the user data onto secondary storage devices. At the end of the input data for a particular job, the job is placed into the job queue where it awaits execution. A limit is placed on the number of reader procedures in the system. In simulation models, each reader may be represented by one transaction which continually loops through a series of operations. The required actions are: 1) read in the job information; 2) wait until the proper time as recorded in the job information; 3) enqueue the job in the proper input queue as described by the job information; 4) return to get the next job. It is the duty of the reader to signal the next part of the simulation that a new job is ready.

The second part of the simulation is the initiator-executor. This section responds to the prompting of the reader, and picks up a job from the job queue. From the step records, storage and other resources are allocated. The executor portion then passes the job through all of its steps, causing the proper waits in the storage. When the last step has been executed, the job is put into the output queue for the last part of the simulation.

The third part of the simulation is the writer program. This section is used only to output the job. Information is picked up from the output queue, and this data determines the length of time the writer is busy with this job. As with both the reader and the initiator-executor, a limited number of writers are available. Each writer is represented by a transaction which loops back to the first of its section.

Timing information is picked up with the records which are part of the jobtrace. In addition, the simulation is terminated by one of two conditions. The first possibility occurs if the model reaches a state where no transactions can be dispatched for execution. This might occur if the model runs out of work, or if a mutually exclusive lock-out condition occurs. The second form of model termination occurs when a preset transaction termination limit is reached. This may occur because a clock was produced which generates transactions at regular intervals and these are then immediately destroyed. In this way a clock effect may be obtained by terminating the transactions and using these transactions as the limiting count.

A sample simulation model is shown in Figure 10. This particular simulation is designed as a study on the effect of storage requirements in a system, but it is probably more important as a sample of the form of simulation. It is assumed that the input data consists of four data items per job step. The first number is the elapsed time period between the previous job and the current job as they are read in. The second number is the amount of storage required by this job step. The storage residency time is represented by the third data item and the fourth item is the number of pages printed by the job. The time information are floating point numbers, but the other two items are integer numbers. It should be apparent from this example that the language is well structured for this type of simulation. More extensive examples may be found in Appendix F.

Simulation of specific hardware devices depends upon the characteristics of these devices. The SEIZE and RELEASE commands are used to reserve exclusive control of a facility for one transaction. If another transaction requests a facility which is already in use, the new transaction is queued into a list based on the priority of the transactions. In this way

```
SAMPLE  :  SYSTEM 50,5  ;
    NOTE   THIS IS A SAMPLE TO SHOW THE FACILITIES OF
            THE SYSTEM SIMULATION LANGUAGE CALLED BOSS.
             THIS IS IN NO WAY REPRESENTATIVE OF ALL OF THE
            FEATURES AVAILABLE IN THE LANGUAGE.   ;
    DCL   (JOBQ,OUTQ)  QUEUE(75) , MAIN STORAGE(370) ,
         CPU FACILITY , EXTIME FLOAT , (QIN,QOUT) EVENT ;
    NOTE  THE FIRST SECTION DEFINES THE READER TRANSACTIONS
         WHICH OBTAIN THE INFORMATION FROM AN EXTERNAL
         SOURCE  ;
    GENERATE  MAX(2),MEAN(0),START(0)   ;
         PF1 = 0 ;
         RDRIN : WAIT UNTIL(PF1) ;
    GET EDIT(PF1,P2,PF3,P4) (SKIP,2(F(8,4),X(4),F(4),X(4)))   ;
         ENQUEUE  JOBQ,QIN ;      GO TO RDRIN  ;
    NOTE   NOW SIMULATE THE INITIATOR-EXECUTOR PART  ;
    GENERATE  MAX(4),MEAN(0),START(0)  ;
    INEXEC : WAIT ON(1,QIN) ;   DEQUEUE JOBQ ;
         ALLOCATE  MAIN,P2  ;      WAIT UNTIL(PF3)  ;
         FREE MAIN,P2  ;     ENQUEUE  OUTQ,QOUT  ;
         GO TO INEXEC  ;
    NOTE   NOW SIMULATE THE WRITER ACTIONS  ;
    GENERATE  MAX(2),MEAN(0),START(0)  ;
    INWTR : WAIT ON(1,QOUT) ;      DEQUEUE OUTQ ;
         IF P4 > 10 THEN  EXTIME = .25 * P4 ;
              ELSE  DO ;     P4 = P4 + 10 ;
                 EXTIME = .15 * P4  ;  END ;
         WAIT UNTIL(EXTIME)  ;
         GO TO INWTR  ;
    NOTE   THE NEXT SECTION DEFINES A CLOCK TO BE USED AS
         A TIME LIMITER  ;
    GENERATE  MEAN(1),DEVI(0)  ;
    TERMINATE  1  ;
  END  ;
```

Figure 10.   A sample simulation in BOSS

an I/O request can be simulated by issuing a SEIZE on a channel and a unit address. To represent the actual data transfer, a WAIT must be issued to hold these facilities until the simulated data transfer completes. After the time period is complete, the facilities are made available for other transactions. The use of a subprocess to do this entire operation will allow a savings in the programming and will allow easy modification to change device type.

In the simulation of an operating system, one of the changes desired might be the total number of resources such as readers, printers, direct access devices, and even central processors. In many cases, the only changes necessary to simulate these modifications would be to change a constant. For example, to increase the amount of primary storage should be just a constant in a declaration, increased printers just means the number of writer transactions is increased, and increased readers is the same form of increase in the number of reader transactions.

Another possible modification might be the addition of multiple central processors. If all of the processors use a common storage device, then a subprocess for the CPU must decide which CPU is free and use that CPU. A more extensive task would be to increase the number of direct access devices. Some preprocessing may be necessary to select particular job accesses for the new devices. These new devices may be difficult to use properly.

Software modifications might mean changes to the simulation model itself. Certain things such as using more job classes for jobs, are more or less trivial changes. On the other hand, modifying the criteria used to select a particular job queue for a job may require some form of external preprocessing.

PART FOUR.

CONCLUSIONS AND SUMMARY

This paper has proposed the combination of two evaluation techniques into one procedure. This combination is postulated to provide a more accurate evaluation for a complete computer system, because the simulation is driven by the detailed data obtained by the monitor. Naturally, the simulation is costly because of the magnitude of the data to be processed, but the data structures and the special simulation language are efficient means of handling this magnitude of data. Simulations of computer systems are generally recognized to be the most generally applicable form of evaluation, so the procedure presented here is postulated to be useful in all forms of performance evaluation.

Although the measurement step of the procedure is primarily designed to provide data to the simulation step, the insight into the operation of the system must not be ignored. The data produced by these measurements may suggest particular areas to investigate. For performance monitoring applications, these measurements may be sufficient to evaluate the potential problem areas in the computer system.

The measurement step is dependent upon a software probe which must be tailored to fit the system on which it is to be run. This fitting process must be done by someone with an intimate knowledge of the computer system. The proper locations must be found to be modified and the data obtained must be properly presented. This portion of the process has been verified by experiments with an IBM 360/65 operating system. It is postulated that other computer systems can be measured in the same way. This particular point is necessary in order to apply this procedure to a general class of computer systems. In fact, this process has been applied to at least one other computer system (see Schwetman (24)).

The creation of a simulation language which will easily provide a model of the operating system is an important part of the total system. The features built into the BOSS language allow the simulation designer to accurately model the computer system. Of course, the designer must still have a certain level of familiarity with the system, but the degree of familiarity varies with the required simulation. It is the author's belief that the BOSS system is misnamed, because it appears to be much more general than just an operating system simulator (see Appendix D).

The production of a job stream trace and the use of these records may be important to a serious system simulation. However, it is believed that the data in these records is seriously degraded because of the timer resolution. Since such a large number of records (approximately two-thirds) have an apparently zero time period, a randomizing factor would have to be applied. This would then make the simulation less accurate in an area where accuracy is very important. The lack of I/O operation start times for so many operations also degrades the accuracy in a similar way. These short-comings are probably sufficient to put a severe burden on the user in the area of the estimation of time distributions. With so many records in these classes, the accuracy of the resultant simulation may depend on some individual's insight into the system actions in these areas.

Another possible shortcoming of the system is the fact that the measurement and subsequent simulation may be dependent upon the software-hardware system which was used. By careful choice of the measurements, this effect should be minimized. Careful study of the system may allow the evaluation of changes in the software or hardware. If certain areas in the configuration are frequently used, the use of these areas may be measured. Modifications may then be studied by varying the simulation to match the proposed modifications. The simulation phase may therefore study deficiencies in the system being evaluated.

The choice of parameters to be measured seems to be adequate for most job-oriented system analysis. In fact, several parts of the data have been used to improve system performance at ISU. I/O activity records are carefully studied to guide the placement of data sets on disks and even to order the information within these data sets. The job dispatching records are being used to guide a new selection of time-slicing parameters, as the previous values are apparently too high.

Several extensions could be considered in this area of system evaluation. First of all, the information provided by this scheme will probably be used in the near future for an evaluation of the HASP(Houston Automatic Spooling Priority) system, compared with a system without HASP. If preliminary information is correct, HASP is an aid to the I/O actions for input and output, but is a degradation to the CPU requirements. These theories will be verified by measurement with the system described here.

Many other programs or systems produce a trace of operations during their execution. A good example is the Time Sharing Option(TSO) of the operating system. Provided with the system is a special trace program. The information available from this program is typical of information required in time-sharing measurement. Swapping, user interaction time, program storage requirements, commands executed, and several other parameters are measured. These data items produce a trace of the activity which may be later processed into a form suitable for driving a simulation model.

An ambitious approach to the problems encountered during this study would involve a combination of hardware and software monitoring schemes. The problem with the I/O operations could be circumvented by a hardware monitor which recorded information on the I/O instructions executed by the computer. These instructions must be a part of any I/O access, no matter what program requires it. This hardware monitor could be a small computer which might also be responsible for the data collection and at least part of the timing. A feature of the IBM/360 computers is available which allows the direct transfer of eight bits of data between two machines. This feature, the direct control feature, would be a possible method of communication between the IBM/360 and another device. It seems that an immediate possiblity would be the addition of a high resolution timer, accessed through the direct control feature. If a combined hardware-software monitor were produced, each part would be able to obtain the data that was most compatible with its characteristics.

In summary, the methods of evaluation should move further into the area of measurement. In particular, each installation needs to measure its system with its normal jobstream. Only by measuring the normal jobstream can realistic evaluations be obtained. Admittedly, the measurement process is a time consuming and sometimes dangerous process, but the results are worth the time and risk. The simulation of the system can be a definite aid to the prediction of future needs, but the initial measurements are probably as important for improvements in the current system's performance. Optimization of a system is still largely a matter of witchcraft, but the modifications may be tested by the system described here. At least, the goals can be recognized when they are obtained.

BIBLIOGRAPHY

1. Bonner, A. J. Using system monitor output to improve performance. IBM Systems Journal 8, No. 4: 290-298. 1969.

2. Buchholz, W. A synthetic job for measuring system performance. IBM Systems Journal 8, No. 4: 309-312. 1969.

3. Calingaert, Peter. System performance evaluation: survey and appraisal. Communications of the ACM 10, No. 1: 12-18. 1967.

4. Cheng, P. S. Trace-driven system modeling. IBM Systems Journal 8, No. 4: 220-239. 1969.

5. Drummond, M. E., Jr. A perspective on system performance evaluation. IBM Systems Journal 8, No. 4: 252-263. 1969.

6. IBM System/360 Operating System Advanced Multiprogramming Analysis Procedure Service Description Manual. Form GH20-0725-0. Poughkeepsie, N.Y., IBM Corporation. 1970.

7. IBM System/360 Operating System General Purpose Simulation System/360 Users Manual. Form GH20-0326-8. Poughkeepsie, N.Y., IBM Corporation. 1970.

8. IBM System/360 Operating System Operator's Reference. Form GC28-6691-2. Poughkeepsie, N.Y., IBM Corporation. 1971.

9. IBM System/360 Operating System System Management Facilities. Form GC28-6712-3. Poughkeepsie, N.Y., IBM Corporation. 1970.

10. IBM System/360 Operating System System Programmers Guide. Form C28-6550-7. Poughkeepsie, N.Y., IBM Corporation. 1970.

11. Katz, Jesse H. An experimental model of System/360. Communications of the ACM 10, No. 11: 694-702. 1967.

12. Knuth, Donald E. and J. L. McNeley. A formal definition of SOL. IEEE Transactions on Electronic Computers 13, No. 8: 409-414.

13. Lucas, Henry C., Jr. Performance evaluation and monitoring. Computing Surveys 3, No. 3: 80-91. 1971.

14. Martin, James. Design of real time computer systems. New

York, N.Y., Prentice-Hall, Inc. 1967.

15. Miller, Edward F., Jr. Bibliography on techniques of computer performance analysis. An unpublished paper. Santa Barbara, California, General Research Corp. 1971.

16. Nielsen, Norman R. The simulation of time-sharing systems. Communications of the ACM 10, No. 7: 397-412. 1967.

17. O'Neil, John T., Jr. Meta pi - an on-line interactive compiler-compiler. Fall Joint Computer Conference Proceedings 1968: 210-218. 1968.

18. Reschly, Christian J. The extension of an ISU version of Metapi to allow for the execution of user defined primatives loaded from a job library dataset. An unpublished Master's degree paper. Ames, Iowa, Computer Science Dept., Iowa State University. 1970.

19. Rothstein, Micheal F. Guide to the design of real-time systems. New York, N.Y., Wiley-Interscience. 1970.

20. Saltzer, Jerome H. and John W. Gintell. The instrumentation of Multics. Communications of the ACM 13, No. 8: 495-500. 1970.

21. Scherr, Allan L. An analysis of time-shared computer systems. Cambridge, Mass., MIT Press. 1967.

22. Scherr, Allan L. Time-sharing measurement. Datamation 12, No. 4: 22-26. 1966.

23. Schorre, D. V. Meta-II a syntax-oriented compiler writing language. Proceedings - 1964 ACM National Conference. 1964.

24. Schwetman, H. D., Jr. A study of resource utilization and performance evaluation of large-scale computer systems. Technical Systems Note-12. Computation Center, University of Texas at Austin. 1970.

25. Seaman, P. H. and R. C. Soucy. Simulating operating systems. IBM Systems Journal 8, No. 4: 264-279. 1969.

26. SHARE Corporation. SHARE Education Committee. SHARE Glossary. New York, N.Y., author, 1967.

27. Sherman, Stephen, Forest Baskett III, and J. C. Browne. Trace driven modeling and analysis of CPU scheduling in a multi-programming system. ACM Workshop on System Perform-

ance Evaluation Proceedings 1971: 173-199. 1971.

28. Stanley, W. I. Measurement of system operational statistics. IBM Systems Journal 8, No. 4: 299-308. 1969.

29. Stanley, W. I. and H. F. Hertel. Statistics gathering and simulation for the Apollo real time operating system. IBM Systems Journal 7, No. 2: 85-102. 1962.

30. Steel, T. B., Jr. Operating systems. Datamation 10, No. 5: 26-28. 1964.

31. Teichroew, Daniel and John Francis Lubin. Computer simulation - discussion of the technique and comparison of languages. Communications of the ACM 9, No. 10: 723-741. 1966.

APPENDIX A

ACRONYMS OF THE IBM OPERATING SYSTEM

These acronyms are taken from IBM reference manuals. No attempt has been made to include a complete set, but only to include those which were used by the author.

BSAM	Basic sequential access method
CPU	Central processing unit
CSS	Computer system simulator
DCB	Data control block
DEB	Data extent block
EXCP	Execute channel program
FLIH	First level interrupt handler
GPSS	General purpose simulation system
ID	Identification
IOB	Input/output block
I/O	Input/output
IPL	Initial program load
MVT	Multiprogramming: variable tasks
PRTY	Priority
PSW	Program status word
QSAM	Queued sequential access method
RQE	Request queue element
SMF	System management facility
SVC	Supervisor call
TCB	Task control block
TIOT	Task I/O table
UCB	Unit control block

APPENDIX B

GLOSSARY OF TERMS

The following definitions were taken from either the IBM Operators Reference Guide(8) or the Share Glossary(26). Some of the definitions have been modified to correspond with current usage.

ACCESS METHOD... A method for transferring data between main storage and a direct access storage or input/output devices.

ADDRESS CONSTANT... A number, or a symbol representing a number, used in calculating storage addresses.

ALIAS... Another name for a member of a partitioned data set; another name for an entry point of a program.

ALLOCATE... To assign a resource for use in performing a specific job, job step, subtask of a job step, or job support task.

APPLICATION PROGRAM... A problem state program written by a user. A job.

ASYNCHRONOUS... Without regular time relationship; unexpected or unpredictable with respect to the execution of a program's instructions.

ATTACH (task)... To create a task and present it to the supervisor.

ATTRIBUTE... A trait; for example, attributes of data include record length, record format, data set name, associated device type and volume information, use, creation date, etc.

AUXILIARY STORAGE... Data storage other than main storage.

AVAILABILITY... The degree to which a software/hardware system is available when needed to process data.

BASIC ACCESS METHOD... Any access method in which each input/output statement causes an input/output operation to occur.

BATCH-PROCESSING... The operational procedure of collecting several jobs together to be input all at one time. The operating system is then responsible for all scheduling and execution. See also BATCHED JOB PROCESSING.

BATCHED JOB PROCESSING... A technique whereby job definitions are placed one behind another on a common input device to form a batch of job definitions that are processed by the CPU with as little operator intervention as possible.

BLOCK (records)...
1. To group records to conserve storage space or to increase the efficiency of access or processing.
2. A blocked record.
3. A portion of a telecommunications message defined as a unit of data transmission.

BUFFER, MAIN STORAGE... An area of main storage that is temporarily reserved for use in performing an input/output operation.

BYTE... Continuous storage equal to eight bits. (Eight bits in the IBM System/360 and System/370).

CALL... The transfer of control from one routine to another routine.

CATALOG...

1. In the operating system, a collection of data set indexes that are used by the control program to locate a volume containing a specific data set.
2. To include the volume information for a data set in the catalog.

CATALOGED PROCEDURE... A set of job control statements that has been placed in a cataloged data set, called the procedure library, and can be retrieved by naming it in an execute statement or started by the START command.

CENTRAL PROCESSING UNIT... All that portion of a computer exclusive of the input, output, peripheral and in some instances, storage units. Also, a unit of a computing system that performs the work of processing data by executing predefined sequences of instructions, such as add, subtract, multiply, and divide instructions.

CHANNEL... A hardware device that connects a CPU and main storage with input/output control units.

CHANNEL ADDRESS WORD... A word in main storage that specifies the location in main storage where a channel program begins.

CHANNEL COMMAND WORD... A doubleword at the location in main storage specified by the CAW. One or more CCWs make up the channel program that directs the channel operations.

CLASS SCHEDULING... The concept of grouping jobs with similar characteristics for input. Class scheduling attempts to present a more optimal job mix to the system.

CLASS, JOB... A set of jobs with similar characteristics.

COMMAND LANGUAGE... The set of commands, succommands, and operands recognized by the system.

COMMAND PROCESSING... The reading, analyzing, and performing of commands issued via a console or a system input stream.

COMPUTING SYSTEM... A central processing unit together with the main storage, input/output channels, control units, direct access storage devices, and input/output devices connected to it.

CONTROL BLOCK... A storage area used by the operating system to hold control information.

CONTROL PROGRAM... A program that is designed to schedule and supervise the performance of data processing work by a computing system.

CONTROL SECTION... That part of a program specified by the programmer to be a relocatable unit, all of which is to be loaded into adjoining main storage locations.

CPU TIME... The amount of time denoted by the central processing unit to the execution of instructions.

DATA CONTROL BLOCK... A control block used by access routines in storing and retrieving data.

DATA DEFINITION NAME... A name appearing in the data control block of a program which corresponds to the name field of a data definition statement.

DATA FILE...

1. A collection of related data records organized in a specific manner. For example, a payroll file (one record for each employee showing his rate of pay, deductions, etc.) or an inventory file (one record for each inventory item, showing the cost selling price, number in stock, etc.).
2. In the operating system, a data set.

DATA MANAGEMENT... A major function of the operating system that includes organizing, cataloging, locating, storing, retrieving, and maintaining data.

DATA SET... The major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. (see also DATA FILE).

DEBUG... To detect, locate, and remove mistakes from a routine.

DEDICATION... Describing the assignment of a system resource (e.g., an I/O device, a program, or a whole system) to one application or purpose.

DIRECT ACCESS... Retrieval or storage of data by reference to its location on a volume rather than relative to the previously retrieved or stored data.

DIRECT ACCESS DEVICE... A device in which the access time is effectively independent of the location of the data.

DIRECTORY... An index that is used by the operating systems control program to locate one or more sequential blocks of data (called members) that are stored in separate partitions of a partitioned data set in direct access storage.

DISABLED... A state of the CPU that prevents the occurrence of certain types of interruptions.

DISPATCHING PRIORITY... A number assigned to tasks to determine the order in which they will use the central processing unit in a multitask situation.

DUMP (main storage)...
1. To copy the contents of all or part of main storage onto an output device, so that it can be examined.
2. The data resulting from number 1.
3. A routine that will accomplish number 1.

DYNAMIC AREA... An area of main storage that is allocated for performing job step or job support tasks.

ENABLED... A state of the CPU that allows the occurrence of certain types of interruptions determined by the current program status word.

EVENT... An occurrence of significance to a task; typically, the completion of an asynchronous operation, such as an input/output operation.

EVENT CONTROL BLOCK... A control block used to represent the status of an event.

EXTERNAL REFERENCE... A reference to a symbol defined in another module.

EXTERNAL SYMBOL... A control section name, entry point name, or external reference; a symbol contained in the external symbol dictionary.

FACILITY...

1. A measure of how easy it is for people to operate, use, and manage the use of a software/hardware system. Together with system performance, the facility of a system is a major factor on which the total productivity of an installation depends.
2. A feature of the operating system designed to serve a particular purpose -- for example, the checkpoint/restart facility.

FIXED STORAGE AREA... That portion of main storage occupied by the resident portion of the control program (nucleus).

GENERAL PURPOSE OPERATING SYSTEM... An operating system designed to handle a wide variety of computing system applications.

GLOSSARY... A collection of glosses.

HARDWARE... The mechanical, magnetic, electrical, and electronic devices from which a computer is constructed.

HARDWARE RESOURCES... CPU time, main storage space, input/output channel time, direct access storage space, and input/output devices, all of which are required to do the work of processing data automatically and efficiently.

HEXADECIMAL... A numbering system with a base of 16; therefore, valid digits range from 0 through F, where F represents the highest units position (15).

HIERARCHY STORAGE... A division of main storage that allows hierarchy 0 and hierarchy 1 to be addressed separately. For MFT and MVT systems with hierarchy support and an IBM 2361 Core Storage Unit, processor storage is addressed as hierarchy 0, and the 2361 is addressed as hierarchy 1. For MVT with hierarchy support, but with no 2361, there are still two hierarchies: both are in processor storage.

"HUMAN ORIENTED" LANGUAGE... A programming language that is more like a human language than a machine language.

I/O-PROCESSOR OVERLAP... The automatic process by which channels control I/O operations while the CPU carries out normal instruction execution.

IBM SYSTEM/360 OPERATING SYSTEM... A comprehensive collection of control program options, language processors, I/O support, application programs, and service programs designed to meet the needs of the users who require the extensive facilities of a large operating system.

INITIAL PROGRAM LOAD... As applied to the system, the initialization procedure that loads the supervisor and the job control processor and begins normal operations.

INITIATOR/TERMINATOR... A part of the job scheduler. In an MFT or MVT configuration of the control program, the initiator/terminator selects a job from the input work queue, allocates resources required to perform a step of the job, loads and transfers control to the program that is executed to perform the job step, and terminates the job step when execution of the program is completed.

INPUT BUFFER... An area of main storage used to store a data block received from an input device for processing by the CPU.

INPUT JOB QUEUE... A collective term for the fifteen queues of job information which the job scheduler uses to select the jobs and job steps to be processed. Each of the fifteen queues is associated with one input job class. (see INPUT WORK QUEUE)

INPUT WORK QUEUE... A queue (waiting list) of job definitions in direct access storage assigned to a job class and arranged in order of priority assignment. Job definitions are entered into an input work queue by one or more reader/interpreters, and are selected and removed by one or more initiator/terminators.

INSTALLATION... A particular computing system in terms of the overall work it does and the people who manage it, operate it, apply it to problems, service it, and use the results it produces.

INTERACTION... In time-sharing applications, a basic unit used to record system activity, consisting of acceptance of a line of terminal input, processing of the line, and response, if any. Interactions are recorded when a user task starts its wait for a line of terminal input.

INTERRUPTION... A transfer of CPU control to the supervisor that is initiated automatically by the computing system or by a problem state program through the execution of a supervisor call (SVC) instruction. The transfer of control occurs in such a way that control can later be restored to the interrupted program, or, in systems that perform more than one task at a time, to a different program.

JOB... The major unit of work performed under operating system control. A job consists of one or several related steps. It is defined by a series of job control language statements.

JOB CLASS... Any one of a number of job categories that can be defined at an installation when using an MFT or MVT control program configuration. Each job can be assigned to any one of several predefined job classes and each initiator/terminator can be directed to initiate jobs from one to three different classes. By classifying jobs and directing initiator/terminators to initiate specific classes of jobs, it is possible to control the mixture of jobs that are performed concurrently.

JOB CONTROL LANGUAGE... A high-level programming language used to code job control statements.

JOB CONTROL STATEMENT... Any one of the control statements in the input job stream that identifies a job or defines its requirements.

JOB MANAGEMENT... A major function of the operating system involving the reading and interpretation of job definitions, the scheduling of jobs, the initiation and termination of jobs and job steps, and the recording of job output data.

JOB PRIORITY... A value assigned to an MVT job that, together with an assigned job class, determines the priority (relative to other jobs) to be used o to be used in scheduling the job and allocating resources to it.

JOB STEP... A unit of work for the computing system from the standpoint of the user, presented to the system by job control statements as a request for execution of a specific program and a description of the resources required by it.

LINK LIBRARY... A partitioned data set which, unless otherwise specified, is used in fetching load modules referred to in execute statements and in other load type operations.

LINK PACK AREA... An area in upper main storage containing a list of track addresses for routines that reside in SYS1.LINKLIB, routines from SYS1.SVCLIB and SYS1.LINKLIB as selected by the user, types 3 and 4 routines, and master scheduler and system modules required resident by system tasks. The link pack area is set up by the nucleus initialization program (NIP) at the time of initial program loading.

LINKAGE CONVENTIONS... A set of operating system conventions that should be adhered to when passing control from one program module to another. Adherence to the conventions helps to ensure program sharing and compatibility.

LINKAGE EDITOR... A processing program that can be used to combine program segments or modules that are independently compiled or assembled. The linkage editor also enables a program that is too large for the space available in main storage to be divided so that executed segments of the program can be overlaid by segments yet to be executed.

LOAD... To place a program into main storage so that it can be executed.

LOAD MODULE... A program or part of a program formed of one or more object modules, the object modules, that is ready to be loaded into main storage by the control program for execution by the CPU.

"MACHINE ORIENTED" LANGUAGE... A programming language that is more like a machine language than a human or mathematical language.

MACRO INSTRUCTION... An instruction in a source language that is equivalent to a specific sequence of machine instructions.

MAIN STORAGE... The storage in a computing system from which a central processing unit can directly obtain instructions and data and to which it can directly return results.

MAIN STORAGE REGION... In an MVT control program configuration, a section of main storage that is allocated by the control program for use in performing a job step or a job support task.

MASTER SCHEDULER... A part of the control program that serves as a two-way communications link between the operator and the system, usually by way of the operator's console. It is used to relay messages from the system to the operator, to execute operator commands, and to respond to replies from the operator. In MFT and MVT control program configurations, the master scheduler is used to start and stop the reader/interpreter, initiator/terminator, and output writer tasks.

MULTIPROCESSING... A technique whereby the work of processing data is shared among two or more interconnected central processing units under integrated control that directly or indirectly communicate with one another, other than through direct human intervention.

MULTIPROCESSING SYSTEM... A computing system employing two or more interconnected processing sys interconnected processing units to execute programs simultaneously.

MULTIPROGRAMMING... A technique by which a computer system can interleave execution of two or more generally unrelated programs, parts of which are residing together in main storage.

NETWORK... In teleprocessing, a number of communication lines connecting a computer with remote terminals.

NUCLEUS... The portion of a control program that always remains in main storage.

OPERATING SYSTEM... An application of a computing system, in the form of organized collections of programs and data, that is specifically designed for use in creating and controlling the performance of other applications.

OPERATIONS STAFF... The members of a data processing installation who receive jobs from the programmers, schedule the order in which the jobs are presented to the system and performed, and direct the overall operation of the system in performing the jobs.

OPERATOR... A member of a data processing installation operations staff who is responsible for directing the operation of a computing system. The same, or a different operator, may perform routine functions such as mounting tape reels and loading card decks.

OUTPUT BUFFER... An area of main storage used to store a data block before it is transferred to an output device.

OUTPUT CLASS... In an MFT or MVT control program configuration, any one of up to 36 different output classes, defined at an installation, to which output data produced during a job step can be assigned. When an output writer is started, it can be directed to process from one to eight different classes of output data.

OUTPUT WRITER... A part of the job scheduler that writes output data sets onto a system output unit, independently of the program that produced such data sets.

OVERLAY... To place a load module or segment of a load module into main storage locations occupied by another (already executed) load module or segment.

PAGING... The process of transmitting pages of information between mainstorage and auxiliary storage, especially when done for the purpose of assisting the allocation of a limited amount of main storage among a number of concurrently executing programs.

PERFORMANCE... Together with facility, one of the two major factors on which the total productivity of a hardware/software system depends. Performance is largely determined by a combination of three other factors: throughput, response time, and availability.

PHYSICAL RECORD... A record that is defined in terms of physical qualities rather than by the information it contains.

POST... To note the occurrence of an event.

PRIORITY... The relative standing a job or task has in the system as opposed to the other jobs and tasks in the system at a given time.

PRIORITY SCHEDULING SYSTEM... A form of job scheduler which uses input and output work queues to improve system performance.

PRIVILEGED INSTRUCTION... An instruction that can only be executed when the CPU is in the supervisor state.

PROBLEM STATE... A state of the central processing unit during which input/output and other privileged instructions cannot be executed. Opposite of supervisor state.

PROBLEM STATE PROGRAM... Any program that is executed when the central processing unit is in the problem state. This includes IBM-distributed programs, such as language translators and service programs, as well as programs written by a user.

PROCESSOR...

1. In hardware, a central processing unit (CPU).
2. In software, a problem state program such as a language translator or service program that is usually provided by IBM and is widely used at an installation.

PRODUCTIVITY... A measure of the work performed by a software/hardware system. Productivity largely depends on a combination of two factors: the facility (ease of use) of the system and the performance (throughput, response time, and availability) of the system.

PROGRAM... A logically self-contained sequence of instructions that can be executed by a computing system to attain a specific result.

PROGRAM STATUS WORD... A doubleword in main storage used to control the order in which instructions are executed, and to hold and indicate the status of the computing system in relation to a particular program.

PROTECTION KEY... A task-oriented indicator (key) that appears in the current PSW whenever a task is active (i.e., has control of the system); this indicator must match the storage keys of all main storage blocks that the task is to use.

QUEUE... A waiting line or list.

QUEUED ACCESS METHOD... An access method that automatically governs the movement of data between the program using the access method and the input/output devices.

READER... A software device which reads a system input stream from a specific input device and deposits it in the input queue with pointers to its data on scratch disk space.

READER/INTERPRETER... A part of the job scheduler that reads and interprets a series of job definitions from a job input stream.

REAL-TIME APPLICATION... An application in which a computing system is used to assist in or guide a process while the process actually transpires.

RECORD... One or more data fields that represent an organized body of related data, such as all of the basic accounting information concerning a single sales transaction.

RELOCATABILITY ... The ability of a program (in the form of a load module) to be dynamically loaded anywhere in main storage.

RESPONSE TIME...

1. The time between the submission of an item of work to the computing system and the return of the results. Loosely, turnaround time.

2. In online systems, the time between the end of a block of user input and the display of system response at the terminal.

RETURN CODE... A number placed in a designated register (the "return code register") at the completion of a program. The number is established by user-convention and may be used to influence the execution of succeeding programs or, in the case of an abnormal end of task, it may simply be printed for programmer analysis.

ROUTINE... A part of a program or subprogram that may have general or frequent use.

SEIZE... In simulation, the action of seizing a facility to prevent other transactions from using that facility.

SERVICE PROGRAM... A processing program such as the linkage editor, sort/merge program, or a utility program that is designed mainly to perform specific services for a user of the program.

SETUP... The act of preparing a computing system to perform a job or job step. Setting up is usually performed by an operator or assistant operator and often involves performing routine function, such as mounting tape reels and loading card decks.

SETUP TIME... The time required by an operator to prepare a computing system to perform a job or job step.

SOFTWARE... The totality of programs and routines used to extend the capabilities of computers, such as generators, compilers, assemblers and operating systems.

SPOOLING... The process of reading job information from a physical reader and making the information available on a faster device. Spooling depends on multiprogramming for concurrent operation of the spooling program and allows virtual card readers for multiprogramming. Spooling may also be applied to the output of data.

STORAGE BLOCK... An area of main storage consisting of 2048 bytes to which a storage key can be assigned.

STORAGE DUMP... A listing of the contents of a storage device or selected parts of it. Synonymous with memory dump and core dump.

SUBPROGRAM... A sequence of instructions stored in a library, that can be incorporated as part of a program.

SUBROUTINE... A relatively short sequence of instructions that can be incorporated into a program to perform a specific function, such as finding the square root of a number.

SUBTASK... A task that is initiated and terminated by a higher order task.

SUPERVISOR... A major part of the operating system control program that is executed when the CPU is in the supervisor state. The supervisor directs and controls the execution of problem state programs and provides them with a variety of services.

SUPERVISOR CALL INSTRUCTION... An instruction that interrupts the program being executed and passes control to the supervisor for the purpose of performing a specific service indicated by the instruction.

SUPERVISOR STATE... A state of the central processing unit during which input/output and other privileged instructions can be executed.

SYNCHRONOUS... Occurring with a regular or predictable time relationship.

SYSIN... A system input stream, also a name used as the data definition name of a data set in the input stream.

SYSOUT... A system output stream. Also, an indicator used in data definition statements to signify that a data set is to be written on a system output unit.

SYSTEM...
(1) An assembly of components united by some form of regulated interaction to form an organized whole. (2) A collection of consecutive operations and procedures required to accomplish a specifc objective.

SYSTEMS ANALYSIS... The examination of an activity, procedure, method, technique, or a business to determine what must be accomplished and how the necessary operations may best be accomplished.

SYSTEM AVAILABILITY... The portion of time a system is or can be used for productive purposes.

SYSTEM GENERATION... The process of using one operating system to assemble and link together into a coherent whole all the required, alternative and optional parts that form a new operating system.

SYSTEM INPUT DEVICE... A device that is assigned to read a job input stream.

SYSTEM MANAGEMENT FACILITIES... An optional control program feature that provides the means for gathering and recording information that can be used to evaluate system usage.

SYSTEM PROGRAMMER...
1. A programmer who is assigned to plan, generate, maintain, extend, and control the use of an operating system with the aim of improving the overall productivity of an installation.
2. A programmer who designs programming systems and other applications.

SYSTEM QUEUE AREA... An area in main storage adjacent to the fixed main storage area. The system queue area is set up by the nucleus initialization program (NIP) at the time of the initial program loading.

SYSTEM RESIDENCE VOLUME... The volume that contains the IPL program, the volume index of the SYSCTLG data set, and the system data sets. The system residence volume must reside on the I/O device which is addressed when initial program loading is performed.

TASK... An independent unit of work that can compete for the resources of the system.

TASK CONTROL BLOCK... The consolidation of control information related to a task.

TASK DISPATCHER... The control program routine that selects from the task queue the task that is to be performed by the central processing unit.

TASK MANAGEMENT... The part of the supervisor that controls and directs the concurrent performance of data processing tasks.

TELECOMMUNICATIONS... The transmission of messages from one location to another over telephone and other communication lines.

THROUGHPUT... The total volume of work performed by a computing system over a given period of time.

TIME-SHARING... A method of using a computing system whereby a number of users can concurrently execute programs with which the users may interact during execution, and generally be assured some minimum amount of program execution per unit time.

TIME SLICE... A uniform interval of CPU time allocated for use in performing a task. Once the interval is over, CPU time is allocated to another task. Thus, a task cannot monopolize CPU time beyond a fixed limit.

TRANSACTION... The units of traffic that are created and moved through processing blocks by a simulation language.

TURNAROUND TIME... The time required for a job to pass through the entire system; the difference between the time the job is returned to a pick-up station and the time the job was submitted to the station.

UNIT ADDRESS... The three-character address of a particular device, specified at the time a system is installed.

UNIT AFFINITY... Forced allocation of a data set on the same unit as another data set.

USER...

1. Anyone who requires the services of a computing system.
2. Under time-sharing systems, anyone with an entry in a user attribute or accounting data set; anyone eligible to log on the system.

UTILITY PROGRAM... A standard routine used to assist in the operation of the computer, e.g., a conversion routine, a sorting routine, a printout routine, or a tracing routine.

VIRTUAL MEMORY... A conceptual form of main storage which does not really exist, but is made to appear as if it exists through the use of hardware and programming.

VOLUME... A section or unit of auxiliary storage space that is serviced by a single read/write mechanism whose operation is entirely independent of any other read/write mechanism.

WAIT CONDITION... The condition of a task that needs one or more events to occur before the task can be ready to be performed by the central processing unit.

WAIT STATE... The state of the system when no instructions are being processed, but the system is not fully stopped. The system can accept I/O and external interruptions, and can be put through the IPL procedure.

WRITER... A software device which selects data sets from designated output classes of the output queue, and routes them as an output stream to a physical output device.

APPENDIX C

DATA COLLECTION MONITOR PROGRAM LISTING

The program used to produce the monitor data is listed on the following pages. This listing is provided as an example of the extreme system dependence of such a program. In many places the addresses to be modified are not apparent. Only careful study will produce the correct results. Also notice the modularity of the program and how it must all fit together into one system. The use of WAIT and POST is the only reliable way to communicate between the various parts of the monitor.

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                     F15OCT70  8/27/71

                                      2          PRINT NOGEN                                                IOST0020
000000                                3 COLLECT  CSECT      A STUDY IN ASYNCHRONOUS ROUTINES                IOST0030
000000 47FF 000C            0000C     4          BC    15,12(15)                                           IOST0040
000004 07C3D6D3D3C5C3E3               5          DC    X'07',CL7'COLLECT'                                  IOST0050
00000C 90EC D00C            0000C     6          STM   14,12,12(13)                                        IOST0060
000010 05C0                           7          BALR  12,0                                                IOST0070
000012                                8          USING *,12                                                IOST0080
000012 50D0 C906            00918     9          ST    13,SAVE+4                                           IOST0090
000016 4110 C902            00914    10          LA    1,SAVE                                              IOST0100
00001A 501D 0008            00008    11          ST    1,8(13)                                             IOST0110
00001E 18D1                          12          LR    13,1  SAVE AREA SET UP                              IOST0120
                                     13 *      ISSUE WRITE TO OPERATOR                                     IOST0130
                                     14          WTO   '   ***  ***  ***  ***  ***  ***  ***  ***  ***'    IOST0140
                                     23          WTO   '      UNLESS YOU LIKE TO IPL,'                     IOST0150
                                     32          WTO   '        CANCEL THIS PROGRAM BY'                    IOST0160
                                     41          WTO   '        REPLYING "GOTIT" ONLY'                     IOST0170
                                     50          WTOR  '    SIGNED ... DANA',REPL,5,OPECB                  IOST0180
                                     62          OPEN  (TAPES,OUTPUT)                                      IOST0190
00089E                               68 TCPP     EQU   X'89E'                                              IOST0200
                                     69          REGS                                                      IOST0210
000102 9110 CA72      00A84          86          TM    TAPES+48,X'10'    IS IT OPEN  ?                     IOST0220
000106 4780 C374            00386    87          BZ    HELPOT                                              IOST0230
                                     88          GETMAIN  R,LV=32768,HIARCHY=1                             IOST0240
000126 4111 0008            00008    98          LA    R1,8(R1)                                            IOST0250
00012A 5010 CA3A            00A4C    99          ST    R1,CURLNG                                           IOST0260
00012E 1861                         100          LR    R6,R1                                               IOST0270
000130 5010 CA0A            00A1C   101          ST    R1,BUFAD1                                           IOST0280
000134 5A10 CA42            00A54   102          A     R1,LENG                                             IOST0290
000138 5010 CA12            00A24   103          ST    R1,HIPOINT   CURRENT HIPOINT FOR START              IOST0300
00013C 5B10 CA1E            00A30   104          S     1,SAFED                                             IOST0310
000140 5010 CA1A            00A2C   105          ST    1,DANGER                                            IOST0320
000144 5A10 CA1E            00A30   106          A     1,SAFED                                             IOST0330
000148 4111 010C            0010C   107          LA    R1,268(R1)                                          IOST0340
00014C 5010 CA0E            00A20   108          ST    R1,BUFAD2        SECOND BUFFER                      IOST0350
000150 D703 CA4E CA4E 00A60 00A60   109          XC    TAPES+12(4),TAPES+12                                IOST0360
000156 5810 0010            00010   110          L     1,16                                                IOST0370
00015A D203 6000 1008 00000 00008   111          MVC   0(4,6),8(1)    LINKLIB DCB                          IOST0380
000160 D203 6004 100C 00004 0000C   112          MVC   4(4,6),12(1)     JOBQE DCB SOMEWHERE HERE           IOST0390
000166 D203 6008 1038 00008 00038   113          MVC   8(4,6),56(1)    DATE                                IOST0400
00016C D203 600C 1054 0000C 00054   114          MVC   12(4,6),84(1)        SVCLIB DCB                     IOST0410
                                    115          TIME  BIN                                                 IOST0420
000178 5006 0010            00010   118          ST    0,16(6)                                             IOST0430
00017C 4166 0014            00014   119          LA    6,20(6)                                             IOST0440
000180 5060 CA3A            00A4C   120          ST    6,CURLNG                                            IOST0450
000184 5810 0010            00010   121          L     1,16                                                IOST0460
000188 5811 0000            00000   122          L     1,0(1)                                              IOST0470
00018C 5811 0004            00004   123          L     1,4(1)                                              IOST0480
000190 5010 CA3E            00A50   124          ST    1,CURTCB                                            IOST0490
000194 D703 CA2A CA2A 00A3C 00A3C   125          XC    OPECB(4),OPECB                                      IOST0500
00019A D703 CA2E CA2E 00A40 00A40   126          XC    TIMECB(4),TIMECB                                    IOST0510
                                    127          WTOR  'REPLY GO TO BEGIN COLLECTION',REPL,2,OPECB         IOST0520
                                    139          WAIT  ECB=OPECB                                           IOST0530
0001D8 D703 CA2A CA2A 00A3C 00A3C   143          XC    OPECB(4),OPECB                                      IOST0540
0001DE D501 CACE CAD3 00AE0 00AE5   144          CLC   REPL(2),OKREP                                       IOST0550
0001E4 4770 C374            00386   145          BNE   HELPOT                                              IOST0560
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70   8/27/71

                                    146 *    NOW INITIALIZE EVERYTHING                                          IOST0570
0001F8 5810 C996            009A8   147          L     1,INITZ  ADDRESS OF INITIALIZATION ROUTINE             IOST0580
0001FC 0AFE                         148          SVC   254   HERE WE GO ...                                   IOST0590
                                    149 *  WHEN ... INITIALIZATION OVER                                       IOST0600
                                    150 *    START AUTOMATIC UPDATE FOR WRITING RECORDS                       IOST0610
                                    151 UPDAT    WAIT  1,ECBLIST=ECBS                                         IOST0620
                                    156 *   GOT SOMETHING ... CHECK IT OUT                                    IOST0630
0001FA 9140 CA2A      00A3C         157          TM    OPECB,X'40'                                            IOST0640
0001FE 4710 C296            002A8   158          BO    OPACT   OPERATOR WANTS SOMETHING                       IOST0650
000202 5820 CA2E            00A40   159          L     2,TIMECB   LOCATION OF OUTPUT                          IOST0660
000206 5830 0010            00010   160          L     3,16     CVT ADDRESS                                   IOST0670
00020A 5833 0058            00058   161          L     3,88(3)      PSEUDO CLOCKS                             IOST0680
00020E 5833 0008            00008   162          L     3,8(3)    LOCAL PSEUDO CLOCK                           IOST0690
000212 5032 0004            00004   163          ST    3,4(2)                                                 IOST0700
000216 D202 CAA7 CA2F 00AE9 00A41   164          MVC   CCWS+1(3),TIMECB+1                                     IOST0710
00021C D703 CA2E CA2E 00A40 00A40   165          XC    TIMECB(4),TIMECB    RESET THE ECB                      IOST0720
000222 4842 0000            00000   166          LH    R4,0(R2)    LENGTH FIELD                               IOST0730
000226 4040 CAAC            00AEE   167          STH   R4,CCWS+6                                              IOST0740
00022A D703 CA76 CA76 00A88 00A88   168          XC    EECB(4),EECB     CLEAR ECB                             IOST0750
000230 D703 CA16 CA16 00A28 00A28   169          XC    DANGECB(4),DANGECB                                     IOST0760
                                    170          EXCP  IOADR                                                  IOST0770
                                    173          WAIT  1,ECBLIST=DESE                                         IOST0780
000246 9140 CA16      00A28         178          TM    DANGECB,X'40'                                          IOST0790
00024C 4710 C276            00288   179          BO    HANGIT                                                 IOST0800
000250                              180 COMPE    EQU   *                                                      IOST0810
                                    181 *  WAIT FOR COMPLETION OF EXCP                                        IOST0820
000250 9101 CA86      00A98         182          TM    CSWF+4,X'01'    EOV NECESSARY  ??                      IOST0830
000254 4780 C24A            0025C   183          BZ    *+8                                                    IOST0840
000258 4580 C382            00394   184          BAL   R8,EOVS     END OF VOLUME PROCESSING                   IOST0850
00025C 917F CA76      00A88         185          TM    EECB,X'7F'     NORMAL COMPLETION                       IOST0860
000260 47E0 C2EC            002FE   186          BNO   OUTOFIT                                                IOST0870
000264 9140 CA16      00A28         187          TM    DANGECB,X'40'                                          IOST0880
000268 4780 C1DC            001EE   188          BZ    UPDAT                                                  IOST0890
00026C 940F C75B      0076D         189          NI    ENQP1+1,X'0F'                                          IOST0900
                                    190          DEQ   (QNM,RNM2,,SYSTEM),RMC=SYSTEM                          IOST0910
000282 47F0 C1DC            001EE   201          B     UPDAT                                                  IOST0920
                                    202 HANGIT   ENQ   (QNM,RNM2,E,,SYSTEM),SMC=SYSTEM                        IOST0930
                                    213          WAIT  1,ECB=EECB                                             IOST0940
0002A4 47F0 C23E            00250   217          B     COMPE                                                  IOST0950
0002A8 D504 CACE CAD3 00AE0 00AE5   218 OPACT    CLC   REPL(5),OKREP     IS THE REPLY VALID  ?                IOST0960
0002AE 4780 C2EC            002FE   219          BE    OUTOFIT                                                IOST0970
                                    220          WTOR  'CAREFUL WHAT YOU SAY...THATS NOT A VALID REPLY',     XIOST0980
                                                       REPL,5,OPECB                                           IOST0990
0002F4 D703 CA2A CA2A 00A3C 00A3C   232          XC    OPECB(4),OPECB                                         IOST1000
0002FA 47F0 C1DC            001EE   233          B     UPDAT                                                  IOST1010
0002FE 5810 C99A            009AC   234 OUTOFIT  L     1,REDOS          SYSTEM FIX-UP ROUTINE                 IOST1020
000302 0AFE                         235          SVC   254                                                    IOST1030
000304 5850 CA0E            00A20   236          L     5,BUFAD2                                               IOST1040
000308 5860 CA3A            00A4C   237          L     6,CURLNG                                               IOST1050
00030C 1965                         238          CR    6,5                                                    IOST1060
00030E 4740 C304            00316   239          BL    BUFR2                                                  IOST1070
000312 5850 CA0A            00A1C   240          L     5,BUFAD1   CHANGE BUFFERS                              IOST1080
000316 5B50 C9D2            009E4   241 BUFR2    S     5,FOUR                                                 IOST1090
00031A 5B50 C9D2            009E4   242          S     5,FOUR                                                 IOST1100
00031E 5830 0010            00010   243          L     3,16     CVT ADDRESS                                   IOST1110
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                             F15OCT70   8/27/71

000322 5833 0058            00058   244            L      3,88(3)     PSEUDO CLOCKS                                  IOST1120
000326 5833 0008            00008   245            L      3,8(3)    LOCAL PSEUDO CLOCK                               IOST1130
00032A 5035 0004            00004   246            ST     3,4(5)      STORE INTO OUTPUT                              IOST1140
00032E 1B65                         247            SR     6,5                                                        IOST1150
000330 D703 5000 5000 00000 00000   248            XC     0(4,5),0(5)    CLEAR LENGTH FIELD                          IOST1160
000336 4065 0000            00000   249            STH    6,0(5)                                                     IOST1170
00033A 4060 CAAC            00ABE   250            STH    R6,CCWS+6                                                  IOST1180
00033E 5050 CA2E            00A40   251            ST     R5,TIMECB                                                  IOST1190
000342 D202 CAA7 CA2F 00AB9 00A41   252            MVC    CCWS+1(3),TIMECB+1     DATA ADDRESS                        IOST1200
000348 D703 CA76 CA76 00A88 00A88   253            XC     EECB(4),EECB                                               IOST1210
                                    254            EXCP   IOADR                                                      IOST1220
                                    257            WAIT   1,ECB=EECB                                                 IOST1230
00035E D703 CA76 CA76 00A88 00A88   261            XC     EECB(4),EECB                                               IOST1240
000364 4110 CAAE            00AC0   262            LA     R1,TPMK                                                    IOST1250
000368 5010 CA8A            00A9C   263            ST     R1,CSWP+8    NOW WRITE TAPEMARK                            IOST1260
                                    264            EXCP   IOADR                                                      IOST1270
                                    267            WAIT   1,ECB=EECB                                                 IOST1280
                                    271 *     OUT NOW ... READY FOR ANALYSIS                                         IOST1290
                                    272            CLOSE  (TAPES,)                                                   IOST1300
000386 58D0 C906            00918   278 HELPOT     L      13,SAVE+4                                                  IOST1310
00038A 98EC D00C            0000C   279            LM     14,12,12(13)                                               IOST1320
00038E 1BFF                         280            SR     15,15                                                      IOST1330
000390 07FE                         281            BR     14        OUT ... OUT ... OUT                              IOST1340
                                    282 EOVS       ENQ    (QNM,RNM,E,,SYSTEM),SMC=SYSTEM                             IOST1350
                                    293            EOV    TAPES                                                      IOST1360
                                    296            DEQ    (QNM,RNM,,SYSTEM),RMC=SYSTEM                               IOST1370
0003BE D703 CA4E CA4E 00A60 00A60   307            XC     TAPES+12(4),TAPES+12                                       IOST1380
0003C4 4188 0008            00008   308            LA     R8,8(R8)                                                   IOST1390
0003C8 07F8                         309            BR     R8                                                         IOST1400
                                    310 *                                                                            IOST1410
                                    311 *   END OF VOLUME ROUTINE                                                    IOST1420
                                    312 *                                                                            IOST1430
                                    313            DROP   12                                                         IOST1440
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70   8/27/71

0003CA                              315 INITIZ   EQU   *                                                    IOST1460
                                    316 *                                                                   IOST1470
                                    317 *   INITIALIZE THE PROGRAMS                                         IOST1480
                                    318 *                                                                   IOST1490
0003CA 0520                         319          BALR  2,0                                                  IOST1500
0003CC                              320          USING *,2                                                  IOST1510
0003CC 5831 000C            0000C   321          L     3,12(1)                                              IOST1520
0003D0 4030 21F2            005BE   322          STH   3,NERR+2                                             IOST1530
0003D4 4030 2200            005CC   323          STH   3,NERR1+2                                            IOST1540
0003D8 4030 23BA            00786   324          STH   3,SAVI+2                                             IOST1550
0003DC 4030 23C4            00790   325          STH   3,SAVEE+2                                            IOST1560
0003E0 4030 2442            0080E   326          STH   3,SAVII+2    INTERRUPT SAVE AREA                     IOST1570
0003E4 4030 244C            00818   327          STH   3,PSW+2    PSW TO BE LOADED                          IOST1580
0003E8 5831 0000            00000   328          L     3,0(1)    IBMORG                                     IOST1590
0003EC D203 2600 3000 009CC 00000   329          MVC   OEXCP(4),0(3)                                        IOST1600
0003F2 D203 2604 303C 009D0 0003C   330          MVC   OERR(4),60(3)                                        IOST1610
0003F8 D207 260C 0078 009D8 00078   331          MVC   OPSW(8),120                                          IOST1620
0003FE 5840 25E4            009B0   332          L     4,EXCPAD                                             IOST1630
000402 5043 0000            00000   333          ST    4,0(3)     SVC 0    EXCP                             IOST1640
000406 5840 25E8            009B4   334          L     4,ERRAD                                              IOST1650
00040A 5043 003C            0003C   335          ST    4,60(3)     SVC 15   ... 15*4                        IOST1660
00040E D7C3 0078 0078 00078 00078   336          XC    120(4,0),120(0)                                      IOST1670
000414 5840 25EC            009B8   337          L     4,INTPTAD                                            IOST1680
000418 5040 007C            0007C   338          ST    4,124    INTERRUPT ADDRESS                           IOST1690
                                    339 *   INITIALIZE FOR THE DISPATCHER                                   IOST1700
00041C 5831 0010            00010   340          L     R3,16(R1)      ADDRESS OF DISPATCHER                 IOST1710
000420 D209 261C 3006 009E8 00006   341          MVC   DISPT1(10),6(3)                                      IOST1720
000426 D209 3006 2632 00006 009FE   342          MVC   6(10,R3),MONTIME1                                    IOST1730
00042C D20B 2626 3042 009F2 00042   343          MVC   DISPT2(12),X'42'(R3)                                 IOST1740
000432 D209 3042 263E 00042 00A0A   344          MVC   X'42'(10,R3),MONTIME2                                IOST1750
000438 9204 0021      00021         345          MVI   33,X'04'    SUPERVISOR STATE FOR THE JOB             IOST1760
00043C 07FE                         346          BR    14    OUT OF INITZ                                   IOST1770
                                    347          DROP  2                                                    IOST1780


00043E 0520                         349 REDOIT   BALR  2,0                                                  IOST1800
000440                              350          USING *,2     BASE REGISTER                                IOST1810
000440 5831 0000            00000   351          L     3,0(1)                                               IOST1820
000444 D203 3000 258C 00000 009CC   352          MVC   0(4,3),OEXCP                                         IOST1830
00044A D203 303C 2590 0003C 009D0   353          MVC   60(4,3),OERR                                         IOST1840
000450 D207 0078 2598 00078 009D8   354          MVC   120(8,0),OPSW    FIXUP COMPLETE ??                   IOST1850
000456 5831 0010            00010   355          L     R3,16(R1)      DISPATCHER ENTRY                      IOST1860
00045A D209 3006 25A8 00006 009E8   356          MVC   6(10,R3),DISPT1                                      IOST1870
000460 D20B 3028 25B2 00028 009F2   357          MVC   40(12,R3),DISPT2                                     IOST1880
000466 9601 0021      00021         358          OI    33,X'01'    BACK TO PROGRAM STATE                    IOST1890
00046A 07FE                         359          BR    14                                                   IOST1900
                                    360          DROP  2                                                    IOST1910
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70   8/27/71

00046C 0520                         362 NEXCP    BALR  2,0                                                   IOST1930
00046E                              363          USING *,2    BASE REGISTER                                  IOST1940
00046E 92F0 24A5         00913      364          MVI   SAVE-1,C'0'                                           IOST1950
000472 900F 24EE               0095C 365          STM   0,15,ERRSAVE                                          IOST1960
000476 5860 25DE               00A4C 366          L     6,CURLNG   CURRENT LOCATION IN BUFFER                 IOST1970
00047A 9200 6000         00000      367          MVI   0(6),X'00'    EXCP RECORD                             IOST1980
00047E 5870 0010               00010 368          L     7,16                                                  IOST1990
000482 5877 0058               00058 369          L     7,X'58'(7)     PSEUDO CLOCKS                          IOST2000
000486 5887 0000               00000 370          L     8,0(7)     SHPC                                       IOST2010
00048A 5E87 0004               00004 371          AL    8,4(7)     T4PC                                       IOST2020
00048E 5890 0050               00050 372          L     9,80     TIMER                                        IOST2030
000492 8A90 0001               00001 373          SRA   9,1                                                   IOST2040
000496 1F89                         374          SLR   8,9      TIME IN TIMER UNITS (ALMOST)                 IOST2050
000498 5080 252E               0099C 375          ST    8,ERRSAVE+64                                          IOST2060
00049C D203 6001 252E 00001 0099C   376          MVC   1(4,6),ERRSAVE+64                                     IOST2070
0004A2 4166 0004               00004 377          LA    6,4(6)                                                IOST2080
0004A6 D204 6001 1000 00001 00000   378          MVC   1(5,6),0(1)     IOB INFO                              IOST2090
0004AC D202 6007 1015 00007 00015   379          MVC   7(3,6),21(1)      DCB ADDRESSS                        IOST2100
0004B2 4166 0003               00003 380          LA    6,3(6)                                                IOST2110
0004B6 5871 0014               00014 381          L     7,20(1)     DCB ADDRESS                              IOST2120
0004BA 9110 7030         00030      382          TM    48(7),X'10'      IS IT OPEN ?                          IOST2130
0004BE 4780 2126               00594 383          BZ    NOWRT                                                 IOST2140
0004C2 D201 6007 701A 00007 0001A   384          MVC   7(2,6),26(7)        DSORG                             IOST2150
0004C8 D200 6009 7024 00009 00024   385          MVC   9(1,6),36(7)      RECFM                               IOST2160
0004CE D202 600A 702A 0000A 0002A   386          MVC   10(3,6),42(7)     MACRF & IFLGS                       IOST2170
0004D4 D200 600D 7030 0000D 00030   387          MVC   13(1,6),48(7)     DCBOFLGS                            IOST2180
                                    388 *     UCB INFORMATION                                                IOST2190
0004DA 5887 002C               0002C 389          L     8,44(7)     DEB ADDRESS                               IOST2200
0004DE D502 1015 8019 00015 00019   390          CLC   21(3,1),25(8)                                         IOST2210
0004E4 4770 2126               00594 391          BNE   NOWRT                                                 IOST2220
0004E8 4188 0000               00000 392          LA    8,0(8)                                                IOST2230
0004EC D200 600E 8018 0000E 00018   393          MVC   14(1,6),24(8)    PROTECT KEY                          IOST2240
0004F2 4166 0001               00001 394          LA    6,1(6)     FOR PROTECT KEY                            IOST2250
0004F6 9180 701A         0001A      395          TM    26(7),X'80'    INDEXED SEQUENTIAL ?                   IOST2260
0004FA 4780 2094               00502 396          BZ    ISORG                                                 IOST2270
0004FE 4188 0010               00010 397          LA    8,16(8)    FOR IS                                     IOST2280
000502 5888 0020               00020 398 ISORG    L     8,32(8)     UCB ADDRESS                               IOST2290
000506 D20F 600E 8000 0000E 00000   399          MVC   14(16,6),0(8)                                         IOST2300
00050C 92FF 601E         0001E      400          MVI   30(6),X'FF'      FLAG IF NO VOL SER                   IOST2310
000510 91A0 8010         00010      401          TM    16(8),X'A0'                                           IOST2320
000514 4780 20B4               00522 402          BZ    NOVOL                                                 IOST2330
000518 D205 601E 801C 0001E 0001C   403          MVC   30(6,6),28(8)    VOLUME=SER                           IOST2340
00051E 4166 0005               00005 404          LA    6,5(6)                                                IOST2350
000522 9520 8012         00012      405 NOVOL    CLI   18(R8),X'20'                                          IOST2360
000526 4770 20C6               00534 406          BNE   NODASD                                                IOST2370
00052A D207 601F 1020 0001F 00020   407          MVC   31(8,6),32(1)                                         IOST2380
000530 4166 0008               00008 408          LA    R6,8(R6)                                              IOST2390
000534 5884 000C               0000C 409 NODASD   L     8,12(4)                                               IOST2400
000538 92FF 601F         0001F      410          MVI   31(6),X'FF'                                           IOST2410
00053C 1288                         411          LTR   8,8                                                   IOST2420
00053E 4780 2102               00570 412          BZ    NONAME     NO TIOT                                    IOST2430
000542 D207 601F 8000 0001F 00000   413          MVC   31(8,6),0(8)     JOBNAME                              IOST2440
000548 4166 0008               00008 414          LA    6,8(6)                                                IOST2450
00054C 92FF 601F         0001F      415          MVI   31(6),X'FF'                                           IOST2460
000550 D501 7028 255A 00028 009C8   416          CLC   40(2,7),ZERO                                          IOST2470
```

```
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70  8/27/71

000556 4780 2102             00570   417            BE    NONAME                                                 IOST2480
00055A 9101 7030       00030         418            TM    48(7),X'01'   IS IT BUSY WITH OPEN ?                   IOST2490
00055E 4710 2102             00570   419            BO    NONAME   YES                                           IOST2500
000562 4A87 0028             00028   420            AH    8,40(7)                                                IOST2510
000566 D20B 601F 8000 0001F 00000    421            MVC   31(12,6),0(8)     DDNAME ENTRY                         IOST2520
00056C 4166 000B             0000B   422            LA    6,11(6)                                                IOST2530
000570 4166 0020             00020   423 NONAME     LA    6,32(6)                                                IOST2540
000574 5960 25BE             00A2C   424            C     6,DANGER                                               IOST2550
000578 4740 2122             00590   425            BL    NOWRT2                                                 IOST2560
00057C 9140 261A       00A88         426            TM    EECB,X'40'                                             IOST2570
000580 4710 213E             005AC   427            BO    HITEST1                                                IOST2580
000584 45E0 22FC             0076A   428            BAL   R14,ENQP                                               IOST2590
000588 5960 25B6             00A24   429            C     6,HIPOINT                                              IOST2600
00058C 47B0 2126             00594   430            BNL   NOWRT                                                  IOST2610
000590 5060 25DE             00A4C   431 NOWRT2     ST    6,CURLNG                                               IOST2620
000594                               432 NOWRT      EQU   *                                                      IOST2630
000594 980F 24EE             0095C   433            LM    0,15,ERRSAVE                                           IOST2640
000598 58A0 255E             009CC   434            L     10,OEXCP                                               IOST2650
00059C 07FA                          435            BR    10                                                     IOST2660
00059E 92FF 600E       0000E         436 BADDCB     MVI   14(6),X'FF'                                            IOST2670
0005A2 5B60 2572             009E0   437            S     6,SXTN                                                 IOST2680
0005A6 0660                          438            BCTR  6,0                                                    IOST2690
0005A8 47F0 2102             00570   439            B     NONAME                                                 IOST2700
0005AC 5960 25B6             00A24   440 HITEST1    C     6,HIPOINT                                              IOST2710
0005B0 4740 2122             00590   441            BL    NOWRT2                                                 IOST2720
0005B4 45E0 22AC             0071A   442            BAL   R14,OUTPUX                                             IOST2730
0005B8 47F0 2126             00594   443            B     NOWRT                                                  IOST2740
                                     444            DROP  2                                                      IOST2750


0005BC 5020 0000             00000   446 NERR       ST    2,0                                                    IOST2770
0005C0 0520                          447            BALR  2,0                                                    IOST2780
0005C2                               448            USING *,2                                                    IOST2790
0005C2 92F1 2351       00913         449            MVI   SAVE-1,C'1'                                            IOST2800
0005C6 900F 239A             0095C   450            STM   0,15,ERRSAVE     SAVE REGISTERS                        IOST2810
0005CA 5860 0000             00000   451 NERR1      L     6,0                                                    IOST2820
0005CE 5060 23A2             00964   452            ST    6,ERRSAVE+8                                            IOST2830
0005D2 5860 248A             00A4C   453            L     6,CURLNG                                               IOST2840
0005D6 9201 6000       00000         454            MVI   0(6),X'01'                                             IOST2850
0005DA 5830 23FA             009BC   455            L     3,COMAD                                                IOST2860
0005DE 05A3                          456            BALR  10,3                                                   IOST2870
0005E0 92F4 2351       00913         457            MVI   SAVE-1,C'4'                                            IOST2880
0005E4 58A0 240E             009D0   458            L     10,OERR                                                IOST2890
0005E8 50A0 23C2             00984   459            ST    10,ERRSAVE+40                                          IOST2900
0005EC 980F 239A             0095C   460            LM    0,15,ERRSAVE                                           IOST2910
0005F0 07FA                          461            BR    10     OFF TO IBMS EXCPERR                             IOST2920
                                     462            DROP  2                                                      IOST2930
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70  8/27/71

0005F2                              464          USING COMMON,3                                                IOST2950
0005F2 92F3 3321      00913         465 COMMON   MVI   SAVE-1,C'3'                                             IOST2960
0005F6 5870 0010            00010   466          L     7,16            CVT                                     IOST2970
0005FA 5877 0058            00058   467          L     7,X'58'(7)     PSEUDO CLOCKS                            IOST2980
0005FE 5887 0000            00000   468          L     8,0(7)      SHPC                                        IOST2990
000602 5E87 0004            00004   469          AL    8,4(7)      T4PC                                        IOST3000
000606 5890 0050            00050   470          L     9,80     TIMER                                          IOST3010
00060A 8A90 0001            00001   471          SRA   9,1                                                     IOST3020
00060E 1F89                         472          SLR   8,9        TIME IN TIMER UNITS (ALMOST)                 IOST3030
000610 5080 33AA            0099C   473          ST    8,ERRSAVE+64                                            IOST3040
000614 D203 6001 33AA 00001 0099C   474          MVC   1(4,6),ERRSAVE+64                                       IOST3050
00061A 4166 0004            00004   475          LA    6,4(6)                                                  IOST3060
00061E D20F 6001 1000 00001 00000   476          MVC   1(16,6),0(1)    RQE        ***  TEST  ***               IOST3070
000624 4166 0010            00010   477          LA    6,16(6)                    ***  TEST  ***               IOST3080
000628 4841 0002            00002   478          LH    4,2(1)                                                  IOST3090
00062C D20F 6001 4000 00001 00000   479          MVC   1(16,6),0(4)    UCB IFORMATION                          IOST3100
000632 92FF 6011      00011         480          MVI   17(6),X'FF'  FLAG FOR NO VOL SER                        IOST3110
000636 91A0 4010      00010         481          TM    16(4),X'A0'                                             IOST3120
00063A 4780 3056            00648   482          BZ    NOTDATP                                                 IOST3130
00063E D205 6011 401C 00011 0001C   483          MVC   17(6,6),28(4)     VOLUME=SER                            IOST3140
000644 4166 0005            00005   484          LA    6,5(6)                                                  IOST3150
000648 5851 0004            00004   485 NOTDATP  L     5,4(1)                                                  IOST3160
00064C 4155 0000            00000   486          LA    5,0(5)                                                  IOST3170
000650 D204 6012 5000 00012 00000   487          MVC   18(5,6),0(5)    IOB FLAGS                               IOST3180
000656 D200 6017 5008 00017 00008   488          MVC   23(1,6),8(5)                                            IOST3190
00065C 9520 4012      00012         489          CLI   18(4),X'20'                                             IOST3200
000660 4770 307C            0066E   490          BNE   NOTDA                                                   IOST3210
000664 D207 6018 5020 00018 00020   491          MVC   24(8,R6),32(R5)                                         IOST3220
00066A 4166 0008            00008   492          LA    R6,8(R6)                                                IOST3230
00066E 5841 000C            0000C   493 NOTDA    L     4,12(1)                                                 IOST3240
000672 4144 0000            00000   494          LA    4,0(4)     TCB                                          IOST3250
000676 5851 0008            00008   495          L     5,8(1)                                                  IOST3260
00067A D202 6018 5019 00018 00019   496          MVC   24(3,6),25(5)                                           IOST3270
000680 4166 0003            00003   497          LA    6,3(6)       DCB ADDRESS ADDITION                       IOST3280
000684 5855 0018            00018   498          L     5,24(5)     DCB                                         IOST3290
000688 D201 6018 501A 00018 0001A   499          MVC   24(2,6),26(5)      DSORG                                IOST3300
00068E D200 601A 5024 0001A 00024   500          MVC   26(1,6),36(5)    RECFM                                  IOST3310
000694 D202 601B 502A 0001B 0002A   501          MVC   27(3,6),42(5)    MACRF&IFLGS                            IOST3320
00069A D200 601E 5030 0001E 00030   502          MVC   30(1,6),48(5)    OFLGS                                  IOST3330
0006A0 5874 000C            0000C   503          L     7,12(4)    TIOT                                         IOST3340
0006A4 92FF 601F      0001F         504          MVI   31(6),X'FF'                                             IOST3350
0006A8 1277                         505          LTR   7,7                                                     IOST3360
0006AA 4780 30F2            006E4   506          BZ    NOTIOT                                                  IOST3370
0006AE D207 601F 7000 0001F 00000   507          MVC   31(8,6),0(7)     JOBNAME                                IOST3380
0006B4 4166 0008            00008   508          LA    6,8(6)                                                  IOST3390
0006B8 92FF 601F      0001F         509          MVI   31(6),X'FF'                                             IOST3400
0006BC D501 5028 33D6 00028 009C8   510          CLC   40(2,5),ZERO                                            IOST3410
0006C2 4780 30F2            006E4   511          BE    NOTIOT                                                  IOST3420
0006C6 9110 5030      00030         512          TM    48(5),X'10'   IS IT OPEN ?                              IOST3430
0006CA 4780 30F2            006E4   513          BZ    NOTIOT    NOT OPEN                                      IOST3440
0006CE 9101 5030      00030         514          TM    48(5),X'01'   IS IT BEING OPENED ?                      IOST3450
0006D2 4710 30F2            006E4   515          BO    NOTIOT     YES                                          IOST3460
0006D6 4A75 0028            00028   516          AH    7,40(5)                                                 IOST3470
0006DA D20B 601F 7000 0001F 00000   517          MVC   31(12,6),0(7)     DD  ENTRY                             IOST3480
0006E0 4166 000B            0000B   518          LA    6,11(6)                                                 IOST3490
```

138

```
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                         F15OCT70   8/27/71

0006E4 4166 0020             00020   519 NOTIOT   LA    6,32(6)                                                  IOST3500
0006E8 5960 343A             00A2C   520          C     6,DANGER                                                 IOST3510
0006EC 4740 3114             00706   521          BL    B102                                                     IOST3520
0006F0 9140 3496       00A88         522          TM    EECB,X'40'                                               IOST3530
0006F4 4710 311A             0070C   523          BO    HITEST2                                                  IOST3540
0006F8 185A                          524          LR    R5,R10                                                   IOST3550
0006FA 45E0 3178             0076A   525          BAL   R14,ENQP                                                 IOST3560
0006FE 18A5                          526          LR    R10,R5                                                   IOST3570
000700 5960 3432             00A24   527          C     6,HIPOINT                                                IOST3580
000704 07BA                          528          BCR   11,R10                                                   IOST3590
000706 5060 345A             00A4C   529 B102     ST    R6,CURLNG                                                IOST3600
00070A 07FA                          530          BR    R10                                                      IOST3610
00070C 5960 3432             00A24   531 HITEST2  C     6,HIPOINT                                                IOST3620
000710 4740 3114             00706   532          BL    B102                                                     IOST3630
000714 18EA                          533          LR    R14,R10                                                  IOST3640
000716 47F0 3128             0071A   534          B     OUTPUX                                                   IOST3650
                                     535          DROP  3                                                        IOST3660
                                     536 *    OUTPUT ROUTINE FOR POST OPERATION                                  IOST3670
00071A 05F0                          537 OUTPUX   BALR  15,0                                                     IOST3680
00071C                               538          USING *,R15                                                    IOST3690
00071C 58A0 F304             00A20   539          L     10,BUFAD2                                                IOST3700
000720 58B0 F300             00A1C   540          L     11,BUFAD1                                                IOST3710
000724 19A6                          541          CR    10,6                                                     IOST3720
000726 4740 F014             00730   542          BL    NOSWPX                                                   IOST3730
00072A 17AB                          543          XR    10,11                                                    IOST3740
00072C 17BA                          544          XR    11,10                                                    IOST3750
00072E 17AB                          545          XR    10,11      SWAPPED REGISTERS                             IOST3760
000730 50B0 F330             00A4C   546 NOSWPX   ST    11,CURLNG                                                IOST3770
000734 5AB0 F338             00A54   547          A     11,LENG                                                  IOST3780
000738 50B0 F308             00A24   548          ST    11,HIPOINT                                               IOST3790
00073C 5BB0 F314             00A30   549          S     11,SAFED                                                 IOST3800
000740 50B0 F310             00A2C   550          ST    11,DANGER                                                IOST3810
000744 5BA0 F2C8             009E4   551          S     10,FOUR                                                  IOST3820
000748 5BA0 F2C8             009E4   552          S     10,FOUR                                                  IOST3830
00074C 1B6A                          553          SR    6,10                                                     IOST3840
00074E D703 A000 A000 00000 00000    554          XC    0(4,10),0(10)    CLEAR LENGTH                            IOST3850
000754 406A 0000             00000   555          STH   6,0(10)                                                  IOST3860
                                     556 *  SET UP FOR POST OPERATION                                            IOST3870
000758 41B0 F324             00A40   557          LA    11,TIMECB                                                IOST3880
00075C 58C0 F334             00A50   558          L     12,CURTCB                                                IOST3890
000760 5840 0010             00010   559          L     4,16                                                     IOST3900
000764 58F4 0098             00098   560          L     15,152(4)                                                IOST3910
000768 07FF                          561          BR    15       BRANCH TO THE POST                              IOST3920
                                     562          DROP  R15                                                      IOST3930
00076A 05F0                          563 ENQP     BALR  R15,0                                                    IOST3940
00076C                               564          USING *,15                                                     IOST3950
00076C 070E                          565 ENQP1    BCR   0,14                                                     IOST3960
00076E 96F0 F001       0076D         566          OI    ENQP1+1,X'F0'                                            IOST3970
000772 41B0 F2BC             00A28   567          LA    11,DANGECB                                               IOST3980
000776 58C0 F2E4             00A50   568          L     12,CURTCB                                                IOST3990
00077A 5840 0010             00010   569          L     4,16                                                     IOST4000
00077E 58F4 0098             00098   570          L     15,152(4)                                                IOST4010
000782 07FF                          571          BR    15      POST THE DANG ECB                                IOST4020
                                     572          DROP  15                                                       IOST4030
```

```
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                       F15OCT70   8/27/71

                                     574 *                                                                 IOST4050
                                     575 *    I/O INTERRUPTS NOW                                           IOST4060
                                     576 *                                                                 IOST4070
000784                               577 NEWIO    EQU   *                                                  IOST4080
000784 5020 0008             00008   578 SAVI     ST    2,8                                                IOST4090
000788 0520                          579          BALR  2,0                                                IOST4100
00078A                               580          USING *,2                                                IOST4110
00078A 900F 21D2             0095C   581          STM   0,15,ERRSAVE                                       IOST4120
00078E 5810 0008             00008   582 SAVEE    L     1,8                                                IOST4130
000792 5010 21DA             00964   583          ST    1,ERRSAVE+8                                        IOST4140
                                     584 *    WHEW ... HAVE REGISTERS SAVED                                IOST4150
000796 92F2 2189       00913         585          MVI   SAVE-1,C'2'                                        IOST4160
00079A 5830 0010             00010   586          L     3,16    CVT POINTER                                IOST4170
00079E 5843 0000             00000   587          L     4,0(3)                                             IOST4180
0007A2 5844 0004             00004   588          L     4,4(4)    CURRENT TCB                              IOST4190
0007A6 5854 0000             00000   589          L     5,0(4)   CURRENT RB                                IOST4200
0007AA 5860 22C2             00A4C   590          L     6,CURLNG                                           IOST4210
0007AE 9202 6000       00000         591          MVI   0(6),X'02'  RECORD TYPE                            IOST4220
0007B2 D207 6001 0040 00001 00040    592          MVC   1(8,6),64  CSW&CAW                                 IOST4230
                                     593 *    NOW FIND THE IOB FOR THIS INTERRUPT                          IOST4240
0007B8 4166 0008             00008   594          LA    6,8(6)                                             IOST4250
0007BC 1B77                          595          SR    7,7                                                IOST4260
0007BE 1887                          596          LR    8,7   ZEROS                                        IOST4270
0007C0 4380 003A             0003A   597          IC    8,58  FIRST  CHANNEL                               IOST4280
0007C4 5480 228E             00A18   598          N     8,MASK1       X'00000007'                          IOST4290
0007C8 5A83 0024             00024   599          A     8,36(3)   IECILK1                                  IOST4300
0007CC 4378 0000             00000   600          IC    7,0(8)        K                                    IOST4310
0007D0 1B88                          601          SR    8,8                                                IOST4320
0007D2 4380 003B             0003B   602          IC    8,59                                               IOST4330
0007D6 8880 0004             00004   603          SRL   8,4                                                IOST4340
0007DA 5480 228A             00A14   604          N     8,MASK2       X'0000000F'                          IOST4350
0007DE 1A78                          605          AR    7,8                                                IOST4360
0007E0 5A73 0024             00024   606          A     7,36(3)    IECILK1                                 IOST4370
0007E4 4387 0000             00000   607          IC    8,0(7)     L                                       IOST4380
0007E8 1B77                          608          SR    7,7                                                IOST4390
0007EA 4370 003B             0003B   609          IC    7,59                                               IOST4400
0007EE 5470 228A             00A14   610          N     7,MASK2                                            IOST4410
0007F2 1A78                          611          AR    7,8                                                IOST4420
0007F4 1A77                          612          AR    7,7                                                IOST4430
0007F6 5A73 0028             00028   613          A     7,40(3)    + IECILK2                               IOST4440
0007FA 4887 0000             00000   614          LH    8,0(7)    UCB                                      IOST4450
0007FE 4818 0014             00014   615          LH    1,20(8)    RQE                                     IOST4460
000802 5830 2232             009BC   616          L     3,COMAD                                            IOST4470
000806 05A3                          617          BALR  10,3                                               IOST4480
000808 92F5 2189       00913         618          MVI   SAVE-1,C'5'                                        IOST4490
00080C D207 0008 224E 00008 009D8    619 SAVII    MVC   8(8,00),OPSW      PSW SET-UP                       IOST4500
000812 980F 21D2             0095C   620          LM    0,15,ERRSAVE                                       IOST4510
000816 8200 0008       00008         621 PSW      LPSW  8(0)    GOOD-BYE                                   IOST4520
                                     622          DROP  2                                                  IOST4530
```

140

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70   8/27/71

                                    624 *                                                                   IOST4550
                                    625 *     OLD TASK TIME RECORD                                          IOST4560
                                    626 *                                                                   IOST4570
00081A                              627           USING *,10                                                IOST4580
00081A 900F A142            0095C   628 OLDTM     STM   0,15,ERRSAVE    SAVE REGISTERS                      IOST4590
00081E 5860 A232            00A4C   629           L     R6,CURLNG       CURRENT LENGTH OF BUFFER            IOST4600
000822 9203 6000      00000         630           MVI   0(R6),X'03'     RECORD TYPE                         IOST4610
000826 5870 0010            00010   631           L     R7,16                                               IOST4620
00082A 5887 0000            00000   632           L     R8,0(R7)     TCB POINTER                            IOST4630
00082E 4188 0004            00004   633           LA    R8,4(R8)   OLD TCB ADDRESS                          IOST4640
000832 45E0 A05C            00876   634           BAL   14,TIMED                                            IOST4650
                                    635           DROP  10                                                  IOST4660
000836                              636           USING *,14                                                IOST4670
000836 980F E126            0095C   637           LM    R0,R15,ERRSAVE     RESTORE REGS                      IOST4680
                                    638           DROP  14                                                  IOST4690
00083A 58F0 0010            00010   639           L     R15,16                                              IOST4700
00083E 9180 F0E4      000E4         640           TM    X'E4'(R15),X'80'                                    IOST4710
000842 478E 001E            0001E   641           BZ    X'1E'(R14)                                          IOST4720
000846 47FE 0010            00010   642           B     16(R14)                                             IOST4730
                                    643 *                                                                   IOST4740
                                    644 *   NEW TASK TIME RECORD                                            IOST4750
                                    645 *                                                                   IOST4760
00084A 0540                         646 NEWTM     BALR  4,0                                                 IOST4770
00084C                              647           USING *,4                                                 IOST4780
00084C 5860 4200            00A4C   648           L     R6,CURLNG                                           IOST4790
000850 9204 6000      00000         649           MVI   0(R6),X'04'    RECORD TYPE                          IOST4800
000854 90BC 4110            0095C   650           STM   11,12,ERRSAVE                                       IOST4810
000858 5870 0010            00010   651           L     R7,16                                               IOST4820
00085C 5887 0000            00000   652           L     R8,0(R7)                                            IOST4830
000860 45E0 402A            00876   653           BAL   R14,TIMED                                           IOST4840
                                    654           DROP  4                                                   IOST4850
000864                              655           USING *,R14                                               IOST4860
000864 98BC E0F8            0095C   656           LM    11,12,ERRSAVE                                       IOST4870
000868 D207 0180 C010 00180 00010   657           MVC   X'180'(8,0),16(12)                                  IOST4880
00086E 980F B030            00030   658           LM    0,15,48(11)                                         IOST4890
000872 8200 0180      00180         659           LPSW  X'180'                                              IOST4900
                                    660           DROP  14                                                  IOST4910
                                    661 *                                                                   IOST4920
                                    662 *   TIME FIELD FORMATION                                            IOST4930
                                    663 *                                                                   IOST4940
000876 0550                         664 TIMED     BALR  5,0                                                 IOST4950
000878                              665           USING *,5                                                 IOST4960
                                    666 *   OUTPUT TIME FIELD                                               IOST4970
000878 5877 0058            00058   667           L     7,X'58'(7)                                          IOST4980
00087C 58A7 0000            00000   668           L     10,0(7)                                             IOST4990
000880 5EA7 0004            00004   669           AL    10,4(7)                                             IOST5000
000884 5890 0050            00050   670           L     9,80                                                IOST5010
000888 8A90 0001            00001   671           SRA   9,1                                                 IOST5020
00088C 1FA9                         672           SLR   10,9    ALMOST TIME                                 IOST5030
00088E 50A0 5124            0099C   673           ST    10,ERRSAVE+64                                       IOST5040
000892 D203 6001 5124 00001 0099C   674           MVC   1(4,6),ERRSAVE+64                                   IOST5050
000898 D202 6005 8001 00005 00001   675           MVC   5(3,6),1(R8)                                        IOST5060
00089E 5888 0000            00000   676           L     R8,0(R8)          TCB POINTER                       IOST5070
0008A2 D200 6008 801C 00008 0001C   677           MVC   8(1,6),28(8)         PROTECT KEY                    IOST5080
0008A8 5878 0000            00000   678           L     R7,0(R8)                                            IOST5090
```

```
DISPATCHER RECORDS FOR CPU UTILIZATION                                                              PAGE   11

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                 P15OCT70   8/27/71

0008AC D201 6009 700A 00009 0000A   679             MVC   9(2,6),10(7)       RB FLAGS               IOST5100
0008B2 D207 600B 7010 0000B 00010   680             MVC   11(08,6),16(R7)    OLD PSW                IOST5110
0008B8 5878 000C            0000C   681             L     R7,12(R8)                                 IOST5120
0008BC 1277                         682             LTR   R7,R7                                     IOST5130
0008BE 92FF 6013      00013         683             MVI   19(6),X'FF'   MARK NONE                   IOST5140
0008C2 4780 5058            008D0   684             BZ    NONNAM                                    IOST5150
0008C6 D207 6013 7000 00013 00000   685             MVC   19(08,6),0(7)                             IOST5160
0008CC 4166 0007            00007   686             LA    R6,07(R6)                                 IOST5170
0008D0 4166 0014            00014   687 NONNAM      LA    R6,20(R6)                                 IOST5180
0008D4 5960 51B4            00A2C   688             C     R6,DANGER                                 IOST5190
0008D8 4740 507C            008F4   689             BL    B142                                      IOST5200
0008DC 9140 5210      00A88         690             TM    EECB,X'40'                                IOST5210
0008E0 4710 5082            008FA   691             BO    HITEST3                                   IOST5220
0008E4 187E                         692             LR    R7,R14                                    IOST5230
0008E6 58F0 5148            009C0   693             L     R15,ENQPTA                                IOST5240
0008EA 05EF                         694             BALR  R14,R15                                   IOST5250
0008EC 18E7                         695             LR    R14,R7                                    IOST5260
0008EE 5960 51AC            00A24   696             C     6,HIPOINT                                 IOST5270
0008F2 07BE                         697             BCR   11,R14                                    IOST5280
0008F4 5060 51D4            00A4C   698 B142        ST    R6,CURLNG                                 IOST5290
0008F8 07FE                         699             BR    R14                                       IOST5300
0008FA 5960 51AC            00A24   700 HITEST3     C     6,HIPOINT                                 IOST5310
0008FE 4740 507C            008F4   701             BL    B142                                      IOST5320
000902 58F0 514C            009C4   702             L     R15,OUTPUXA                               IOST5330
000906 07FF                         703             BR    R15                                       IOST5340
                                    704             DROP  5                                         IOST5350
```

```
  LOC  OBJECT CODE      ADDR1 ADDR2  STMT   SOURCE STATEMENT                                F15OCT70   8/27/71

000908 40E2C1E5C540C1D9               706           DC    CL12' SAVE AREA'                             IOST5370
000914 0000000000000000               707 SAVE      DC    18F'0'                                       IOST5380
00095C 0000000000000000               708 FPRSAVE   DC    18F'0'                                       IOST5390
0009A4                                709           DS    F                                            IOST5400
0009A8 000003CA                       710 INITZ     DC    A(INITIZ)                                    IOST5410
0009AC 0000043E                       711 REDOS     DC    A(REDOIT)                                    IOST5420
0009B0 0000046C                       712 EXCPAD    DC    A(NEXCP)                                     IOST5430
0009B4 000005BC                       713 ERRAD     DC    A(NERR)                                      IOST5440
0009B8 00000784                       714 INTPTAD   DC    A(NEWIO)                                     IOST5450
0009BC 000005F2                       715 COMAD     DC    A(COMMON)                                    IOST5460
0009C0 0000076A                       716 ENQPTA    DC    A(ENQP)                                      IOST5470
0009C4 0000071A                       717 OUTPUXA   DC    A(OUTPUX)                                    IOST5480
0009C8 00000000                       718 ZERO      DC    F'0'                                         IOST5490
0009CC                                719 OEXCP     DS    F                                            IOST5500
0009D0                                720 OERR      DS    F                                            IOST5510
0009D8                                721 OPSW      DS    D                                            IOST5520
0009E0 00000010                       722 SXTN      DC    F'16'                                        IOST5530
0009E4 00000004                       723 FOUR      DC    F'4'                                         IOST5540
0009E8                                724 DISPT1    DS    CL10                                         IOST5550
0009F2                                725 DISPT2    DS    CL12                                         IOST5560
0009FE 58A0E00A05BA                   726 MONTIME1  DC    XL6'58A0E00A05BA'                            IOST5570
000A04 0000081A                       727           DC    A(OLDTM)                                     IOST5580
000A08                                728           DS    H                                            IOST5590
000A0A 58A0E04607FA                   729 MONTIME2  DC    XL6'58A0E04607FA'                            IOST5600
000A10 0000084A                       730           DC    A(NEWTM)                                     IOST5610
000A14 0000000F                       731 MASK2     DC    X'0000000F'                                  IOST5620
000A18 00000007                       732 MASK1     DC    X'00000007'                                  IOST5630
000A1C                                733 BUFAD1    DS    F                                            IOST5640
000A20                                734 BUFAD2    DS    F                                            IOST5650
000A24                                735 HIPOINT   DS    F                                            IOST5660
000A28                                736 DANGECB   DS    F                                            IOST5670
000A2C                                737 DANGER    DS    F                                            IOST5680
000A30 000000A0                       738 SAFED     DC    F'160'                                       IOST5690
000A34 00000A2880000A88               739 DESE      DC    A(DANGECB),XL1'80',AL3(EECB)                 IOST5700
000A3C                                740 OPECB     DS    F'0'                                         IOST5710
000A40 00000000                       741 TIMECB    DC    F'0'                                         IOST5720
000A44 00000A3C                       742 ECBS      DC    A(OPECB)                                     IOST5730
000A48 80000A40                       743           DC    X'80',AL3(TIMECB)                            IOST5740
000A4C                                744 CURLNG    DS    F                                            IOST5750
000A50                                745 CURTCB    DS    F                                            IOST5760
000A54 00003EF4                       746 LENG      DC    F'16116'                                     IOST5770
                                      747 TAPES     DCB   MACRF=(E),DDNAME=OUTTAPE,DEVD=TA             IOST5780
000A88                                782           DS    0F                                           IOST5790
000A88 40000000                       783 EECB      DC    XL4'40000000'                                IOST5800
000A8C                                784           DS    0F                                           IOST5810
000A8C 02                             785 IOADR     DC    X'02'     SUPPRESS LENGTH INDICATION         IOST5820
000A8D                                786           DS    XL3                                          IOST5830
000A90 00000A88                       787 ECBA      DC    A(EECB)     ECB ADDRESS                      IOST5840
000A94                                788 CSWP      DS    2F                                           IOST5850
000A9C 00000AB8                       789           DC    A(CCWS)    CCW ADDRESS                       IOST5860
000AA0 00000A54                       790           DC    A(TAPES)    DCB ADDRESS                      IOST5870
000AA4                                791           DS    F                                            IOST5880
000AA8 0001                           792           DC    H'1'                                         IOST5890
000AAA                                793           DS    H                                            IOST5900
000AAC 0000000000000000               794           DC    2F'0'                                        IOST5910
```

```
  LOC  OBJECT CODE      ADDR1 ADDR2  STMT    SOURCE STATEMENT                                         F15OCT70    8/27/71

000AB4 00000000
000AB8 0100000020000000               795 CCWS      CCW    01,0,32,0                                         IOST5920
000AC0 1F00000020000000               796 TPMK      CCW    31,0,32,0                                         IOST5930
000AC8 E2E8E2C4E2D54040               797 QNM       DC     CL8'SYSDSN'                                       IOST5940
000AD0 E2E8E2F14BD3C9D5               798 BNM       DC     C'SYS1.LINK'                                      IOST5950
000AD9 E2E8E2C3E3D3C7                 799 FNM2      DC     C'SYSCTLG'                                        IOST5960
000AE0                                800 REPL      DS     CL5                                               IOST5970
000AE5 C7D6E3C9E3                     801 OKREP     DC     CL5'GOTIT'                                        IOST5980
                                      802           END                                                      IOST5990
```

APPENDIX D

BOSS LANGUAGE STATEMENTS

SYSTEM

label:SYSTEM terminations [,snap_interval];

Because the SYSTEM statement is the primary entry point of a program, it may only appear as the first statement of each simulation program. Additionally, the SYSTEM statement may only appear once in a given simulation program and must also be labeled to provide a label for the program. Execution parameters which control the number of terminations and the "snapshot" interval are passed to the program by this statement. The proper form of the statement is shown above where "terminations" is an integer specifying the number of TERMINATE counts to be used as a maximum. The optional "snap_interval" parameter will cause a "snapshot" of the current termination dump at intervals equal to the integer value of this parameter.

Assignment Statements

variable = expression;

Assignment statements in the BOSS language are similar to assigment statements in many other languages. Precedence relations are used which cause multiplication and division evaluations to be performed before addition and subtraction. Left to right evaluation is performed with these precedence relations. To force the evaluation of subgroups, parenthesized expressions are allowed within assignment statements. Only single assignments may be made for BOSS variables.

Three distinct assignment statement variable types are implemented. These three(integer, floating point, and Boolean) are used to set values into variables for later use. Since variable type may be dependent upon context, a particular order of test is necessary.

The preferred variable type for system simulation was chosen to be integer. The first attempt to class an assignment statement tries to find an integer expression for the right-hand part. If an integer expression (one constructed totally of integers and integer variables) is found, simple assignment occurs and the program continues. If, however, the expression contains noninteger parts, then the entire expression is evaluated in floating-point arithmetic and then converted to integer form by truncation.

Floating point variables are useful only for certain variables such as time periods. These variables must be declared, and caution must be taken in their use. If an expression is evaluated for assignment to a floating point variable, conversions may be necessary. If the entire expression is integer then the conversion is done after the expression is evaluated. If the expression contains other floating point items, then the entire expression is evaluated in floating point.

Boolean variables are the third form for the assignment statement. Boolean variables are only useful as arguments of IF statements. However, used properly for an expression which is repeated many times, Boolean variables may save both time and space. To achieve this goal, Boolean variables must be declared. The precedence relations provide that arithmetic comparison will be performed first, followed by negation, and then the Boolean intersection and union functions. Evaluation is also left to right with parentheses to group sub-expressions.

A number of built-in functions are also available to the BOSS system. Most of these functions return floating point arguments. These functions include ABS (absolute value), RANDOM (a random number generator), LN (the natural logarithm function), SQRT (square root function), NORMAL (returns an observation from a normal distribution), MARK (returns the current time), and MOD (the remainder function). These functions may be used in an expression anywhere that a variable may be used.

NOTE

```
NOTE any_string_not_containing_semi-colons ;
```

The NOTE statement is a nonexecutable statement which is used to insert comments into the simulation. The input string following this keyword is ignored until the first semi-colon is detected, when normal translation continues.

END

```
END;
```

The END statement is the direction to the compiler to perform housekeeping functions and subsequently to signal completion of execution. This statement must follow each complete simulation program or subroutine.

EXIT

```
EXIT;
```

The EXIT statement effects a return from the current program or subroutine. The appropriate housekeeping is performed, and execution returns to the calling program.

GO TO

```
GO TO label;
```

The GO TO (or GOTO) statement is an unconditional branch to the label identifier following the keywords GO TO. Subscripted or variable label identifiers are not allowed.

SAVE

```
SAVE;
```

The SAVE function stores all of the current information necessary to restart the simulation at a later time. If the SAVE instruction is executed more than once, only the most recent data is retained.

RESTORE

```
RESTORE;
```

The complementary function for the SAVE instruction, RESTORE restarts the system at the point of the last SAVE. If no data set exists for the RESTORE operation, no action results. Typically, the RESTORE would be used immediately following the SYSTEM statement.

EXECUTE

```
EXECUTE process_name[(parm1,parm2,...,parmN)];
```

The EXECUTE statement is the subroutine call operation. The processing is transferred to the subroutine specified in this statement. Parameters are passed through a parameter list in the statement. The proper form of the EXECUTE statement is shown above.

PROCESS

```
PROCESS process_name[(parm1,parm2,...,parmN)];
```

In order to define a common subroutine for a process, the PROCESS statement is used as a header. The PROCESS statement directs the compiler to form a new section of code with unique names and locations. The only communication between the main program and these subprocesses must be achieved through the parameter lists. Each process must have an END statement as its last statement. All subprocesses must follow the main program in the input deck. The PROCESS statement is formed exactly like the EXECUTE statement.

DECLARE

To reserve space for variables, a DECLARE (or DCL) statement is used. This statement is the only way to reserve or declare the dimensions and type of subscripted variables. A discussion of variable types may be found elsewhere in this description.

WAIT

```
WAIT[UNTIL(expression)][ON(event_variables)];
```

The function of accumulating time is primarily performed by the WAIT statement. The WAIT statement causes the current transaction to suspend execution and allows another transaction to become the current transaction. Two types of events and the combinations of these events are used to signal completion of the wait interval. These types are dependent upon either the completion of a specified time interval, or the completion of some simulation activity as defined by that activity itself. The second type of event performs the action of communicating between different transactions in the simulated system. The action of waiting for a certain time period is achieved by using the UNTIL form of the statement. The desired time period is specified by the expression contained within the parentheses. An example of the time period WAIT operation is:

```
WAIT UNTIL(5);
```

which will cause the transaction to remain at this point for a period of five clock cycles.

The event completion form of the WAIT statement allows a delay until any specified number of events have completed.

```
WAIT ON(1,AEVE,BEVE,CEVE);
```

which specifies that the WAIT will continue until one of the three events completes.

If both methods are combined, the WAIT continues only until one of the operations completes.

SIGNAL

SIGNAL event_name;

The SIGNAL statement is half of the communication effort between various transactions. This allows a transaction to tell another transaction to resume its execution. The event name to be used for the SIGNAL statement is used to signal the event for the completion of the WAIT period.

CLEAR

CLEAR event_name;

The CLEAR statement is used to reset the event variable which is tested by the WAIT statements.

ALLOCATE

ALLOCATE store_name,size[,[sub_ident],[conditional]];

To indicate the storage of a discrete item, the ALLOCATE instruction reserves a certain number of storage units. The storage area must have a declaration statement to indicate its maximum size. This storage area is designated by the identifier following the ALLOCATE keyword. A storage hierarchy may be set up by a subscripted storage variable reference. The second parameter of the ALLOCATE operation must be an integer expression which specifies the amount of storage to be reserved by this instruction.

Two optional parameters may also be specified. The first is an identifier or integer which serves as an identifier for the allocated storage. This allows a further subdivision of the memory into areas. The second operation allows a conditional allocation. If the storage area does not have a large enough free block, then the label identifier of this parameter receives control. If no alternate processing is specified, the ALLOCATE function suspends processing of the transaction until the space is available. Suspended processes are kept in list ordered by their priority to determine the next request to be attempted when some additional space is available.

FREE

FREE store_name,size[,[sub_ident],[conditional]];

The FREE instruction removes storage units from the reserved status and returns them to an unallocated status. The identifiers for the process are the same as those used by ALLOCATE. If the FREE instruction fails, the alternate label address is entered. The particular area freed is determined by the size and sub-identifier. The oldest area is freed first.

ENQUEUE

```
ENQUEUE queue_name[,queue_post_list];
```

The ENQUEUE operation is used to create a list or queue of transactions ordered by their priority. The queue name defines a queue to which the transaction is added. Execution of the transaction continues with the next sequential instruction, but an exact copy of the transaction is created to be added into the queue. The event variables specified in the queue post list are signaled to indicate the occurrence of the event.

DEQUEUE

```
DEQUEUE queue_name;
```

The copy of a transaction in the specified queue or list is copied into the current transaction and execution continues sequentially. DEQUEUE then destroys the copy in the list and removes its entry. If a list is empty, a wait occurs until something is added into the queue.

GET/PUT

```
{GET | PUT}[FILE(file_name)]iotype;
```

The input/output instructions GET and PUT are used for generalized I/O forms. These instructions allow the simulation to output or input data for use in the simulation. The file name may either be specified by the FILE subparameter, or may have a default name. The default input file name is SYSIN. Its attributes define it to be a card image stream file with a record length of eighty characters. This data set may also be blocked if desired.

The default output file name is SYSPRINT. This data set may not be changed by the user, because it is the error message output file. No simulation program should be attempted without this data set. If the data set is not available to the program, an abnormal end condition is created.

I/O files to be used by the simulation may be only sequential organizations. The record format may be any format supported by the operating system.

```
LIST(variable_list)
```

The I/O type may be either LIST-oriented or EDIT-oriented. The LIST form shown above is the form which allows free format input and output defined by the compiler. The only information required is the list of identifiers to be processed. Proper type conversions are performed internally for the processed data.

EDIT(variable_list)(format_list)

The EDIT I/O type is designed to allow specification of the format for the I/O operation. The variable list is used to point to the variables needed for I/O, but conversion is now performed according to the directions in the format list. The format list is a collection of format codes which may have duplication factors associated with them. The format codes which are implemented are:

SKIP[(number)] ... Skip the number of lines designated. If no number is specified, skip one line.

COLUMN(number) ... Move to column specified by the number. If the number is smaller than the current column, move to the indicated column on the next line or record.

PAGE ... Begin a new page.

A[(number)] ... Process a character string. If the number is specified, output that number of characters. Otherwise, process the current length of the string.

X(number) ... Insert the specified number of blanks in the record.

F(number) ... Process an integer number. The length of the number (the number of decimal digits) is specified. Leading zeros are suppressed.

E(number,number) ... Process a floating point number. The total number of characters is given by the first number and the number of digits following the decimal point is given by the second operand.

MAP ... Output a representative map of the storage specified by the instruction, showing the areas presently reserved and their identifiers.

SPACE ... Output the numbers representing the total available space and the largest free space in the specified storage.

DISPLAY(queue variable) ... Output the number of transactions waiting in the specified queue.

DISPLAY(event variable) ... Output the word "WAIT" if a wait is in effect for this variable, output the word "COMP" if the event is complete, and output the word "CLEAR" if the event has been cleared.

The last four of these format codes may only occur in the PUT statement and never in the GET statement.

IF

IF boolean_expr THEN statements ELSE statements;

The IF statement is a decision making statement which allows logical testing and conditional statement execution dependent upon the test results. The test condition is a Boolean expression using several Boolean connectives. These Boolean connectives are:

>=	greater than or equal to
<=	less than or equal to
=	equal to
<	less than
>	greater than
¬=	not equal to
&	and
|	or
¬	not

The first six of these (>=,<=,=,<,>,¬=) are used as connectives between numeric expressions. The last three (&,|,¬) are used as Boolean connectives between Boolean variables or expressions.

If the Boolean expression is true, then the statement or statement group following the THEN instruction is executed. If the expression is false, the ELSE clause is executed. If the ELSE clause is omitted, then no special action is taken if the statement is false. In order to group more than one statement together, the statements must be preceded by the word DO followed by a semi-colon, and followed by the word END followed again by a semi-colon. The allowed statements are selected from the group of executable statements excluding the GENERATE statement.

<u>SEIZE</u>

SEIZE facility_name;

The SEIZE action exclusively reserves a particular facility for the transaction issuing the SEIZE. No other transaction is allowed to SEIZE a facility until the current transaction is finished with the facility. A transaction which is prevented from SEIZEing a facility is linked to a chain of transactions waiting for this facility. When the facility is free, the highest priority and eldest transaction at that priority is allowed to SEIZE the facility.

<u>RELEASE</u>

RELEASE facility_name;

After a facility has been SEIZEd, it may be freed by the RELEASE command. This command allows a waiting transaction to SEIZE the facility.

PRIORITY

```
PRIORITY integer_expression;
```

The PRIORITY operation allows a change in the priority of the entering transaction. The integer value of the expression is used as the priority of the transaction until it is TERMINATEd or explicitly changed again.

TERMINATE

```
TERMINATE[integer_expression];
```

The TERMINATE operation eliminates or destroys the entering transaction. The overall TERMINATE count is decremented by the value of the expression. If no expression is specified, no decrement occurs.

GENERATE

```
GENERATE [,MAX(int) ][,MEAN(int) ][START(int) ][DEVI(int) ]
                    [,parmlist];
```

The GENERATE statement creates transactions according to the parameters specified in the statement. Each option uses an integer number to determine its value. The option MAX specifies the maximum number of transactions which will be created by this GENERATE statement. The three options MEAN, START, and DEVI define the mean time between creations, the first transaction creation time and the standard deviation around the mean for creation times. The parameter list option allows the initialization of transaction parameters as they are created. Either floating point or integer variables are allowed in the parameter list.

TABULATE

```
TABULATE table_id;
```

The TABULATE statement is used to output statistics compiled for a specified set of variable names. The output is produced by the system in a standard form. The variables to be tabulated are specified by the table identified in the statement. All statistics are output as recorded to the time of the TABULATE statement.

APPENDIX E

FORMAL LANGUAGE DEFINITION OF BOSS

The following META PI definitions were produced from the data set which is used as input for the compiler-compiler. These definitions totally define the syntax and the semantics of the BOSS compiler.

```
boss     :=  lblstmt2 ';' .ERR('04  EXPECTED SEMI-COLON') $(
             lblstmt2 .ERR('08  UNDECODABLE STATEMENT') ';'
             .ERR('04  EXPECTED SEMI-COLON')) ;

lblstmt2  :=  lblstmt | .IGN(-) (.LATCH(labx) | .EMPTY) 'END'
             .OUT('58F.' *1 , A2 , '051F') .EXREF('ZBOUT')
             .LABEL(*1) .OUT('58F10000' , '05EF') .OUT('58DD0004'
             , '98ECD00C' , '1BFF' , '07FE') .DECK ;

labx     :=  .ID .SAV(*) ( ':' .NOP(C) .RES .TYPE('AOLABEL')
             .ERR('08  PREVIOUSLY DEFINED ...  NOT A LABEL')
             .LABEL(* S) | .IGN(R) ':' ) ;

lblstmt   :=  .LATCH(labx) lblstmt .NOP(I) | .NOP(C) systmt | (
             estmt | iostmt | tabu | exec | ifst | decl | genr |
             allo | prcss | qcom | trcst | waitstm | .LATCH(assgn)
             | .IGN(-) .LATCH(bassgn)) ;

estmt    :=  'NOTE' .TO(';') | 'EXIT'
             .OUT('58DD0004','98ECD00C','1BFF','07FE' ) | 'TERMI-
             NATE' (.INT .OUT('581'.IGEN)| .EMPTY .OUT('1B11'))
             .OUT('58F.@termi' , '07FF') | ('GOTO' | 'GO' 'TO' )
             .ID .TYPE('AOLABEL') .ERR('08  BRANCH TO A NON-LABEL
             VARIABLE') .OUT('58E.' * , '07FE') | 'RELEASE' .ID
             .TYPE('88FACIL') .ERR('04  RELEASE OF A NON-FACILITY
             VARIABLE') .OUT('581.' * , '94F71000' , '58F.@relse'
             , '05EF') | 'SEIZE' .ID .TYPE('88FACIL') .ERR('04
             NON-FACILITY VARIABLE MAY NOT BE SEIZED') .OUT('581.'
             * , '58F.' *1 , '91081000' , '078F' , '58F.@seiz' ,
             '05EF') .LABEL(*1) .OUT('581.@me' , '58110014' ,
             'D2031000E000' , '96081000') | 'SAVE'.OUT('0510' ,
             '45110008') .EXREF('ZBSVE') .OUT('58F10000' , '05EF')
             | 'RESTORE' .OUT('0510' , '45110008')
```

```
          .EXREF('ZBRSTR') .OUT('58F10000' , '05EF') | 'PRIORI-
          TY' iexpr .OUT('58E.@me' , '42' OF 'E0000').IGN(-)|
          'SIGNAL' $(.ID .TYPE('84EVENT') .ERR( '08  ONLY EVENT
          VARIABLES MAY BE SIGNALED') .OUT('58E.' *1 , A2 ,
          '051E') .EXREF('ZBSIG') .LABEL(*1) .OUT('58F10000' ,
          '581.' * , '05EF')( ',' | .EMPTY)) | 'CLEAR' .ID
          .TYPE('84EVENT') .ERR('08  NOT AN EVENT VARIABLE')
          .OUT('58E.' * , '947FE000') ;

assgn  :=  .LATCH(prmasgn) | .ID .SAV(* S) .RES indx '='
          (.LATCH(expri) (.RES) (.TYPE('84INTE') .OUT('50' OF P
          '0000') .IGN(-) | .TYPE('84FLOAT') .OUT('18E' OF ,
          '10FE' , '54E.X8000') .SAV(W8 S) .OUT('90EF' R ,
          '964E' R , '2B' +2 0 , '6A' 0 R8 , '70' 0 OF
          '0000').IGN(-) | .IGN(-) '¢') .IGN(-) | expr (.RES)
          (.TYPE('84FLOAT') .OUT('70' 0 OF '0000').IGN(- -2) |
          .TYPE('84INTE') .ERR('16  IMPROPER TYPE VARIABLE')
          .SAV(0 S S) .OUT('38' +2 R , '2B' R R , '3A' -2 ,
          '6E' 0 '.X4E00' , '58F' W8 , '60' 0 'F0000' ,
          '58FF0004') .IGN(R8) .OUT('0510' , '47B10006' ,
          '13FF' , '50F' OF '0000') .IGN(-) | .IGN(- R -) ';'
          )) ;

iexpr   :=  .LATCH(expri) | iexprx ;

iexprx  :=  expr .SAV(0 S S) .OUT('38' + R , '2B' R R , '3A' -2
          , '6E' 0 '.X4E00' , '58F' W8 , '60' 0 'F0000' , '58F'
          + '0004' , '0510' , '47B10006' , '13' OF OF) .IGN(R8
          -2) ;

systmt  :=  'SYSTEM' .ONCE .ERR('04  ONLY ONE SYSTEM STMT AL-
          LOWED') .SAV('@snap') .NOP('84INTE') .TSET
          .SAV('@terms') .TSET .IGN(0) .INT .OUT('58E'.IGEN ,
          '50E.@terms') (',' .INT .OUT('58E'.IGEN ,
          '50E.@snap') | .EMPTY .OUT('1BEE' , '50E.@snap'))
          inistf ;

inistf  :=  .OUT('58E.@extr' , '58EE0000' , '41F.@me' ,
          '50FE000C' , '58F.@dispt' , '50FE0008' , '581.@genrt'
          , '58F.@init' , '58FF0000' , '05EF' , '58F.@dispt' ,
          '07FF' , A4) .LABEL('@init') .EXREF('ZBINIT')
          .LABEL('@end') .EXREF('ZBEND') .LABEL('@atab')
          .EXREF('ZBATAB') .LABEL('@extr') .EXREF('ZBEXTRC')
          terminte  seize  release

 *     Other routines which may be resident are placed here
          .LABEL('@dispt') .OUT('58E.@extr' , '58E0E000' ,
          '5810E000' , '41101000' , '582.' *1 , '1211' , '0772'
          , '58F.@end' , '58F0F000' , '05EF' , '58D0D00C' ,
```

```
                '98ECD00C' , '1BFF' , '07FE') .LABEL(*1)
                .OUT('501.@me' , '58F10010' , '50F.@now' , '58F01000'
                , '50F0E000' , '58F0100C' , 'D503E0041010' , '072F' ,
                '18EF' , '58F.@atab' , '58F0F000' , '07FF') ;

bassgn  :=  .ID .TYPE('81BOOL') .ERR('08  IMPROPER BOOLEAN')
                indx'='.NOP(C) boole .OUT('D200' P '000' OF '000')
                .IGN(- -)

blvar   :=  '¬' blvar .OUT('13' OF OF , '06' OF '0' ) | ('TRUE'
                | '1') .OUT('1B' + OF , '06' OF '0') | ('FALSE' |
                '0') .OUT('1B' + OF) | .ID .TYPE('81BOOL') indx
                .OUT('18E' OF , '1B' OF OF , '43' OF 'E0000') ;

bprim   :=  .LATCH(blvar) | '(' boole ')' | .LATCH(icompr) |
                expr bltst expr .OUT('1BEE' , '39' -2 , '18' + 'E' ,
                '58E.' *1 , '07' R 'E' , '13' OF OF , '06' OF '0')
                .LABEL(*1) .IGN(-2) ;

icompr  :=  expri bltst expri .OUT('1BEE' , '19' - , '18' OF
                'E' , '58E.' *1 , '07' R 'E' , '13' OF OF , '06' OF
                '0') .LABEL(*1) ;

bltst   :=  '>=' .SAV('B') | '<=' .SAV('D') | '=' .SAV('8') |
                '<' .SAV('4') | '>' .SAV('2') | '¬=' .SAV('7') ;

bterme  :=  bprim $('&' bprim .OUT('14' - ) | '|' bprim
                .OUT('16' -)) ;

bterm   :=  bterme .ERR('08  IMPROPER BOOLEAN EXPRESSION');

boole   :=  ('¬' bterm .OUT('13' OF OF , '06' OF '0') | bterm)
                .OUT('58E.' W1 , '42' OF 'E0000' , '18' OF 'E')
                .IGN(R1) ;

ifst  :=  'IF' boole .OUT('58E.' *1 , '95FF' OF '000' , '077E')
                .IGN(-) 'THEN' boss2 .OUT('58E.' *2 , '07FE')
                .LABEL(*1) (.LATCH(eclse) | .EMPTY) .LABEL(*2) ;

eclse   :=  ';' 'ELSE' .NOP(C) boss2 ;

boss2   :=  'DO' ';' $(lblstmt ';') .IGN(-) 'END' .ERR('08
                UNDECODABLE STATEMENT') | lblstmt .ERR('08
                UNDECODABLE STATEMENT') ;

varb  :=  .LATCH(prmflt) | .ID .SAV(* S) .RES indx .RES typcon
                ;

typcon  :=  .TYPE('84FLOAT') .OUT('78' +2 OF '0000') .IGN(-) |
```

```
        .TYPE('84INTE') .OUT('58E' OF '0000') .IGN(-) conflt
        ;

prim   := .LATCH(elemf) | 'ABS(' expr ')' .OUT('30' 0 0) | '('
        expr ')' | .NUM .OUT('78' +2 .NGEN) | .INT
        .OUT('58E'.IGEN) conflt | varb ;

conflt  :=  .OUT('10FE' , '54E.X8000') .SAV(W8 S) .OUT('58' +
        W8 , '90EF' OF '000' , '964E' OF '000' , '2B' +2 0 ,
        '6A' 0 OF '0000') .IGN(R8 -) .EMPTY ;

elemf  :=  ('RANDOM(' .INT .OUT('580'.IGEN) .SAV('ZBRNDM')
        .SAV(*2) .SAV(*1) elcom1 | 'NORMAL(' expr .OUT('58F.'
        *2 , '70' 0 'F0004') .IGN(-2) (',' expr .OUT('70' 0
        'F0008') | .EMPTY .OUT('41100001' , '501F0008'))
        .SAV('ZBNRML') .SAV(*2) .SAV(*1) elcom1
        .OUT('00000000') | ('SQRT(' .SAV('ZBSQRT') | 'LN('
        .SAV('ZBLNX')) expr .OUT('58F.' *2 , '70' 0 'F0004' )
        .SAV(*2) .SAV(*1) elcom1) .LABEL(*1) .OUT('58F10000'
        , '05EF' , '58E.' *2 , '78' +2 'E0004');

elcom1  :=  .OUT('58E.' R , A2 , '051F') .LABEL(R) .EXREF(R)
        .OUT('00000000') ')' ;

secn   :=  prim $('*' prim .OUT('3C' -2) | '/' prim .OUT('3D'
        -2));

term   :=  '-' secn .OUT('33' 0 0) | '+' secn | secn ;

expr   :=  term $('+' term .OUT('3A'-2) | '-' term
        .OUT('3B'-2));

expri  :=  termi $('+' termi .OUT('1A'-) | '-' termi
        .OUT('1B'-));

termi  :=  '-' secni .OUT('13' OF OF) | '+' secni | secni;

secni  :=  primi $(('*' primi .SAV('C') | '/' primi .SAV('D'))
        .OUT('18E' OF) .IGN(-) .OUT('180' OF , '8E000020' ,
        '1' R '0E' , '18' OF '1')) ;

primi  :=  .LATCH(elemi) .OUT('58E.' *1 , A2 , '051E')
        .EXREF(R) .LABEL(*1) .OUT('58F10000' , '05EF' , '18'
        OF '1') | 'MARK' .OUT('58' + '.NOW' , '58' OF OF
        '0000') | 'MOD(' expri ',' expri ')' .OUT('18E' OF)
        .IGN(-) .OUT('180' OF , '8E000020' , '1D0E' , '18' OF
        '0') | 'ABS(' expri ')' .OUT('10' OF OF) | '(' expri
        ')' | ivar ;
```

```
elemi  := 'RANDOM(' .INT .OUT('580' .IGEN) .SAV('ZBRANDI') ')'
          ;

waitstm  :=  'WAIT' ('UNTIL(' expr ')' .OUT('58E.' *1 , '30' 0
          0 , '078E' , '58E.' *2 , '70' 0 'E0004') | .EMPTY)
          .OUT('581.' *2) (oncla .OUT('50E10008') | .EMPTY)
          .OUT( '47F1000C' , A4) .LABEL(*2) .EXREF('ZBWT')
          .OUT('00000000' , '00000000' , '58F10000' , '05EF')
          .LABEL(*1) ;

oncla  :=  'ON(' ivarx ',' .OUT('581.' *1 , A2 , '05E1' ,
          '00000000') evnts .LABEL(*1) .OUT('5B1.I0004' ,
          '96801000' , '50' OF 'E0000') ')' .IGN(-) ;

ivarx  :=  ivar .ERR('12  ONLY SIMPLE INTEGER VARIABLES AL-
          LOWED');

ivar  :=  .NUM 'ø' | .INT .OUT('58' + .IGEN) | .LATCH(prmint) |
          .ID .TYPE('84INTE') indx .OUT('58' OF OF '0000') ;

evnts  :=  .ID .TYPE('84EVENT') .ERR('08  ONLY EVENT VARI-
          ABLES') .OUT('000,' *) $(',' .ID .TYPE('84EVENT')
          .ERR('08  ONLY EVENT VARIABLES') .OUT('000.' *)) ;

allo  :=  ('ALLOCATE' .SAV('ZBALLO') | 'FREE' .SAV('ZBFREE'))
          .ID .TYPE('04STOR') .ERR('12  NOT A STORAGE') indx
          .OUT('581.' *1 , '50' OF '10004') .IGN(-) ',' iexpr
          .OUT('50' OF '10008') .IGN(-) (',' (.ID .OUT('58E.'
          *) | .INT .OUT('58E'.IGEN) | .EMPTY .OUT('58E.@me' ,
          '58EE0000'))(',' .ID .TYPE('A0LABEL') .ERR('08  LABEL
          VARIABLE EXPECTED') .SAV('580.' *)
          |.EMPTY.SAV('1B00')) | .EMPTY .OUT('58E.@me',
          '58EE0000').SAV('1B00')) .OUT('50E1000C' , '47F10010'
          ,A4) .LABEL(*1) .EXREF(X R) .OUT('00000000' ,
          '00000000' , '00000000' , '58F10000' , R , '05EF');

trcst  :=  'TRACE' .SAV('ZBTR') trctps .OUT('58F.' *2) .SAV(*2)
          .SAV(*1) parms ')' .LABEL(*1) .OUT('58F10000' ,
          '05EF');

trctps  :=  'JOB(' .SAV(R 'J') | 'STEP(' .SAV(R 'S') | 'IOP('
          .SAV(R 'I') | 'EXIT(' .SAV(R 'E') ;

iostmt  :=  ('PUT' .SAV('O') .SAV('SYSPRINT') | 'GET' .SAV('I')
          .SAV('SYSIN')) iocall ;

iocall  :=  .OUT('58E.' *1 , '07FE') .LABEL(*2) ('FILE(' .ID
          .OUT(I '#' #* ':') ')' | .EMPTY .RES .OUT('#' #*
          ':')) .LABEL(*1) .SAV(*2) iotype;
```

```
iotype  :=  'LIST(' .SAV(X 'ZBOUTL' R) .OUT( '58F.' *2 , X
        '581.' R , '501F0004' , '41FF0004') .SAV(*2).SAV(*1)
        parms ')' .OUT('00000000') .LABEL(*1) .OUT('58F10000'
        , '05EF') | 'EDIT(' .SAV(X 'ZBOUTE' R) .OUT('58F.' *2
        , X '581.' R , '501F0004' , '41FF0004') .SAV(*2)
        .SAV(*1) parms ')' .OUT('00000000') .LABEL(*1)
        .OUT('58F10000' , '05EF') edtfmt ;

edtfmt  :=  .OUT('58E.' *1 , '07FE') .LABEL(*2) '(' formats ')'
        .OUT('47F.' *2) .LABEL(*1) ;

formats  :=  frmti (',' formats | .EMPTY) ;

frmti  :=  .INT .OUT('58' + .IGEN) .LABEL(*1) ('(' formats ')'
        | frmcde) .OUT('46' OF '.' *1) .IGN(-) | frmcde ;

frmcde  :=  ('SKIP' ('(' intprt ')' | .EMPTY .OUT('41' +
        '00001' , '50' OF '10000') .IGN(-)) .OUT('1B11') |
        ('COLUMN(' | 'COL(') intprt ')' .OUT('41100004') |
        'PAGE' .OUT('41100008') | 'A' ('(' intprt ')' |
        .EMPTY .OUT('1B' + OF , '50'OF '10000') .IGN(-))
        .OUT('4110000C') | 'X(' intprt .OUT('41100010') ')' |
        'F(' intprt (',' .OUT('41110004') intprt | .EMPTY)
        ')' .OUT('41100014') | 'E(' intprt
        .OUT('D20310041000') ',' intprt ')' .OUT('41100018')
        | 'MAP' .OUT('4110001C') | 'SPACE' .OUT('41100020'))
        .OUT('18FE' , '05EF');

intprt  :=  .INT .OUT('58' + .IGEN , '50' OF '10000') .IGN(-) ;

exec  :=  'EXECUTE' .ID.TYPE('80SUBS') .ERR('12  ILLEGAL
        PROCESS NAME') .SAV(*) .OUT('58F.' *2) .SAV(*2)
        .SAV(*1) ('(' parms ')' | .EMPTY .OUT('9680F000' , A2
        , '051E') .LABEL(R) .EXREF(R)) .LABEL(*1)
        .OUT('58F10000','05EF');

parms  :=  prmx .OUT('41FF0004' , '50' OF 'F0000' , '92' R
        'F000') .IGN(-)(',' parms .OUT('00000000') | .EMPTY
        .OUT( '9680F000' , '58E.' R , A2 , '051E') .LABEL(R)
        .EXREF(R) .OUT('00000000')) ;

prmx  := prmprm|.ID prmtyp indx | .NUM .OUT('41' + .NGEN)
        .SAV('02') | .INT .OUT('41' + .IGEN) .SAV('01') | .SR
        .OUT('58' + '.' *2 '581.' *1 , '07F1') .LABEL(*2)
        .OUT('#' #* ':' , '0700') .LABEL(*1) .SAV('03') ;

prmtyp  :=  .TYPE('04INTE') .SAV('01') | .TYPE('04FLOAT')
        .SAV('02') | .TYPE('80CHAR') .SAV('03') |
        .TYPE('00LABEL') .SAV('04') | .TYPE('04EVENT')
```

```
          .SAV('05') | .TYPE('08FACIL') .SAV('06') |
          .TYPE('01BOOL') .SAV('07') | .TYPE('00STOR')
          .SAV('08') | .TYPE('00QNAM') .ERR('08  IMPROPER PA-
          RAMETER') .SAV('09') ;

prcss  :=  'PROCESS' .ID .LABEL(*) ('(' prms ')' | .EMPTY) ;

prms  :=  .ID .ERR('08  ONLY SIMPLE IDENTIFIER NAMES MAY BE
          PARMS') .OUT('41E.' * , '41110004' , 'D203E0001000')
          (',' prms | .EMPTY) ;

tabu  :=  'TABULATE' .ID .TYPE('00TABLE') .OUT('58F.' *1 , A2 ,
          '051F') .EXREF('ZBTBIT') .OUT('00000000') .LABEL(*1)
          .OUT('58E.' * , '50E10004' , '58F10000' , '05EF') ;

qcom  :=  'DEQUEUE' .ID .TYPE('01QNAM') .ERR('08  NON-QUEUE
          VARIABLE') indx .OUT('58F.' *1 , A2 , '051F')
          .EXREF('ZBQOUT') .LABEL(*1) .OUT('58F10000' , '181'
          OF , '05EF').IGN(-)| 'ENQUEUE' .ID .TYPE('01QNAM')
          .ERR('08  NON-QUEUE VARIABLE') indx .OUT('58F.' *2 ,
          '41FF0004') .SAV(*2) .SAV(*1) (','
          qposts.OUT('00000000')|.EMPTY .OUT('9680F000' , A2 ,
          '051E') .LABEL(R) .EXREF('ZBQIN') .OUT('00000000'))
          .LABEL(*1) .OUT('58F10000' , '50' OF '10004' ,
          '05EF') .IGN(-) ;

qposts  :=  qpsts .OUT('41FF0004' , '50' OF 'F0000') .IGN(-)
          (',' qposts .OUT('00000000') | .EMPTY .OUT(
          '9680F000' , '58E.' R , A2 , '051E') .LABEL(R)
          .EXREF('ZBQIN') .OUT('00000000'));

qpsts  :=  .ID .TYPE('84EVENT') .ERR('08  ILLEGAL EVENT VARI-
          ABLE') indx;

indx  :=  .OUT('58' + '.' *) ('(' .OUT('18E' OF , '58' OF OF
          '0000') indexl $(',' indexl) ')' .ERR('08  UNMATCHED
          PARENTHESES') | .EMPTY) ;

indexl  :=  expri .OUT('41EE0004' , '181' OF , '1B00' ,
          '5C0E0000') .IGN(-) .OUT('1A' OF '1') ;

decl  :=  ('DECLARE' | 'DCL') declist $(',' declist) ;

declist  :=  ('(' dlist | decelm typset .TSET) .IGN(*) ;

dlist  :=  decelm (',' dlist .TSET | ')' typset .TSET) ;

decelm  :=  .ID .SAV(*) ('(' decbnds $(',' decbnds) ')' |
          .EMPTY) ;
```

```
decbnds  :=  .INT .SAV(R ',' *)(':' .INT .SAV(R ':' *) |
         .EMPTY) ;

typset  :=  'LABEL' .NOP('80LABEL') | 'EVENT' .NOP('84EVENT') |
         'FACILITY' .NOP('88FACIL') | 'FLOAT' .NOP('84FLOAT')
         | 'INTEGER' .NOP('84INTE') | 'BOOLEAN' .NOP('81BOOL')
         | 'ENTRY' .NOP('80SUBS') | 'INFILE' .NOP('84SYSIN') |
         'OUTFILE' .NOP('84PRINT') | 'STORAGE('  .INT
         .NOP('00STOR/00' * ) ')' | 'QUEUE(' .INT .SAV(*) (','
         'PRTY' .NOP('01QNAM/00' R) | (',' 'FIFO' | .EMPTY)
         .NOP('01QNAM/80' R)) ')' | 'TABLE(' .OUT('58E.' *1 ,
         '07FE') .SAV(R S) .LABEL(R) tabfrm .LABEL(*1)
         .NOP('00TABLE') ')' ;

tabfrm  :=  .OUT('58F.' *1 , '58EF0000') .SAV(*1)
         .SAV('ZBTBITA') parms ;

genr  :=  'GENERATE' .GENRT .OUT('58F.' *2) .SAV(*2) .SAV(*1)
         .OUT('41100001' , '401F000A') gnstf .OUT('41FF0008')
         gpars .LABEL(*1) .OUT('58F10000' , '05EF') ;

gnstf  :=  gnprt (',' gnstf | .EMPTY) ;

gnprt  :=  ('MAX' .SAV('04') | 'MEAN' .SAV('06') | 'START'
         .SAV('08') | 'DEVI' .SAV('0A')) .INT .OUT('58' +
         .IGEN , '40' OF 'F00' R) .IGN(-) ;

gpars  :=  prmx .OUT('41FF0004' , '50' OF 'F0000' , '92' R
         'F0000') .IGN(-) (',' gpars .OUT('00000000') | .EMPTY
         .OUT('9680F000' , '58E.' R ,  A2 , '051E') .LABEL(R)
         .EXREF('ZBGENT') .OUT('00000000' , '00000000' ,
         '00000000')) ;

terminte  :=  .LABEL('@termi') .OUT('58E.@terms' , '1BE1' ,
         '58F.@finis' , '07DF' , '50E.@terms' , '58' +
         '.@extr' , '58F.' *1 , '18E' OF) .LABEL(*1)
         .OUT('181E' , '58EE0000' , '41EE0000' , '59E.@me' ,
         '077F' , 'D2031000E000' , '58F.' *2 , A2 , '051F')
         .EXREF('ZBFRWRK') .LABEL(*2) .OUT('58F10000' , '181E'
         , '58E.@dispt' , '07FF') .IGN(-) .LABEL('@finis')
         .OUT('58F.@away' , A2 , '051F') .EXREF('ZBOUT')
         .LABEL('@away') .OUT('58F10000' , '07EF' , '58DD0004'
         , '98ECD00C' , '1BFF' , '07FE') ;

seize  :=  .LABEL('@seiz') .OUT('58210004' , '583.@me' ,
         '50F30008' , '50130014' , '584.@extr' , '58F.' *1 ,
         '18E4') .LABEL(*1) .OUT('185E' , '58EE0000' ,
         '41EE0000' , '19E3' , '077F' , 'D2035000E000' ,
         '58F.' *2 , A2 , '05EF') .EXREF('ZBQSP') .LABEL(*2)
```

```
          .OUT('1813' , '58020004' , '58FE0000' , '58E.@dispt'
          , '07FF') ;

release  :=  .LABEL('@relse') .OUT('780.@now' , '7B010000' ,
          '58210004' , '7A020008' , '70020008' , '58320004' ,
          '1233' , '078E' , '58F.' *1 , '1B11') .LABEL(*1)
          .OUT('41110001' , '58330000' , '1233' , '077F' ,
          '58F.' *2 , '5912000C' , '07DF' , '5012000C')
          .LABEL(*2) .OUT('58320004' , 'D20320043000' ,
          '58F.@chain' , A2 , '051F') .EXREF('ZBDISP')
          .LABEL('@chain' .OUT('58F10000' , '182E' , '1813' ,
          '05EF' , '07F2') ;

prmprm  :=  prmflt .SAV('02') | prmint .SAV('01')  ;

prmflt  :=  'PF' prmpart  ;

prmint  :=  'P'  prmpart  ;

prmpart  :=  .INT .OUT('58' + '.@me' , '58E' .IGEN , '58' OF OF
          '0008' , '1AEE' , '1AEE' , '1A' OF 'E')  ;

prmasgn  :=  prmflt '=' expr .OUT('70' 0 OF '0000') .IGN(-2 -)
          | prmint '=' iexpr .OUT('50' OF P '0000') .IGN(- -) ;
```

APPENDIX F

ELEMENTS OF THE ISU META PI COMPILER-COMPILER

Meta languages such as Backus Normal Form (BNF) were the precursors of efforts to systematically produce compilers. The original purpose of these meta languages was to standardize the definition of programming languages and to provide a rigid structure for that definition. The extensions to this philosophy naturally evolved into the area of the compiler-compiler system. The assumption was made that if the language could be described in some form of meta language then a translator could be produced which would automatically produce a compiler for that language.

The original meta languages were primarily concerned with the syntactic qualities of the language, that is, those properties which define the validity of a language statement. A compiler must perform the function of syntax verification for the input statements. This verification may be defined by a meta language, therefore, the obvious process might involve a meta language translator for syntax checking.

The second major requirement of a compiler is not in general satisfied by the meta languages. The association of meaning (semantics) with a given statement is the phase which produces the necessary computer instructions. These instructions may be in the form of actual machine code or as an intermediate instruction set which may later be interpreted or converted to machine code. This part of a compiler is not described by the Backus Normal Form or other meta languages.

The two primary tasks of a compiler are to check the syntax of the input statements and then to produce the proper instructions according to the semantics of the language. A proper meta compiler-compiler language must provide facilities for both syntactic and semantic definitions. These basic facilities were designed into the META series of compiler-compilers described by D.V.Schorre and his associates at UCLA (23).

The basic parsing algorithm of the META type compiler is top-down left to right and deterministic. Top-down means the compiler first decides which rule should be satisfied next and then checks the input (or calls new rules) according to the alternatives of the rule. On the other hand, a bottom-up parser would check the input and then determine which rules may be used to describe it. The top-down parser has some advantages for a compiler-compiler. First, the compiler generates code immediately. This generation allows generality, in particular, for incremental compilation. Error detection is easily achieved because of the deterministic parser. Backup is only provided when explicitly requested. Thirdly, the deterministic parser has fewer choices to pick from, therefore, it is faster than the non-deterministic parser.

As previously stated, the first task of a compiler-compiler language is to provide a syntax checking capability. This syntactic capability has been provided by earlier meta languages such as BNF. Unfortunately, BNF was not designed with semantic operations in mind.

The META PI compiler-compiler as described by J.T. O'Neil (17) has attempted to remedy this situation. An extended BNF is used to contain both syntactic and semantic elements. The result of this compiler-compiler is machine code which is the compiler for the language described. This code consists primarily of a set of subroutine calls which perform a recursive left to right scan of the source statements of the particular compiler language it describes.

Four major extensions were made to the BNF form. These extensions were made to include semantic operations and to simplify the description of the language. These four extensions are:

1. The inclusion of factoring and the addition of an iterative operator. Two reasons motivated these changes. First, the use of a $ enables the compiler to identify an iterative operation immediately. This greatly simplifies the compilation process. Second, from a descriptive point of view, the iterative description simplifies the identification of proper strings defined by the statement. The $ is interpreted to mean

"followed by an arbitrary sequence of". Therefore, the BNF statement

<A> ::= <B> | <A><C> | <A><D>

becomes

A := B$(C | D)

2. The semantics are included within the syntax of a statement. This allows generation of object code as the scan of the statement proceeds. In many statements the generation of code and the input scan complete simultaneously. Both syntactic and semantic operations are aided by commands called primatives which provide standard actions and tests.

3. As previously noted, backup of the scan and code generation is explicitly controlled through special commands. This facility allows a retry with a different definition to resolve ambiguities.

4. The compiler writer is provided with the capability of generating compile time error comments with a special error command. This capability allows extensive error messages with a minimum of effort.

Minor differences in the writing of the statements also distinguish META PI from BNF. The following conventions will hold:

META PI	BNF
:=	::=
ABC	<ABC>

'XYZ' XYZ

In addition:

1. A ; (semi-colon) will terminate a META PI statement.

2. Parentheses will be used to simplify BNF and to provide an indication of factoring.

3. A $ replaces BNF finite state recursion.

META PI statements contain 3 types of elements:

1. The syntactic elements create code to test for syntactic expressions in the source input. These elements are used to generate the syntax verification phase of the compiler.

2. The semantic elements assign a given meaning to the input string. These elements produce the computer instructions for the execution of the program.

3. META syntactic elements are compiled into code which will allow the user's compiler to efficiently resolve possible conflicts (ambiguities) by the use of a backup facility.

These three element types are combined to produce the META PI input which will define a user compiler. The general form of a META PI statement is:

LABEL := expression ;

The left hand side is a unique identifier which serves as a reference to the expression on the right hand side (a META PI identifier is defined as a letter (A-Z) followed by an arbitrary sequence of letters or digits). The character pair := serves as a delimiter and separates META PI statements from user source statements. These statements are compiled into recursive code, therefore, the expression may contain either a direct or indirect reference to itself. One of three results is produced by these statements:

1. True. The input scan has satisfied the expression and the input pointer is updated past the correctly scanned data.

2. False. The input does not satisfy the expression. Therefore, the input pointer is left unaltered.

3. Error. A prefix of the expression is correctly identified but the remainder of the expression is not satisfied by the input string. The input pointer is partially updated and the error routine inserts a ? after the last character successfully scanned.

A detailed discussion of the various elements which comprise META PI expressions follows. These elements are grouped into like types for ease of reference.

Syntactic Elements

'XXXXX...X'	The X's represent any character string. This syntactic element creates code which tests the current input string for the sequence of characters contained within the apostrophes.
ABC	This produces a call to the routine or expression labeled ABC. The expression represented by ABC is the definition of the next part of the input string. This is the first of two forms of linkage to routines. The second form is written with a period preceding the name. In general, the distinction is that the rou-

tines with a preceding period are considered to be built-in functions while the ordinary call is generally to a function written in META PI. When the period notation is used META PI will assume that the call is not recursive.

.ID — This routine makes a test for a META PI identifier. Code is generated to link to the .ID routine. The period notation implies that this routine does not subsequently link to itself.

.EMPTY — A special syntactic operation which forces the true setting of the truth indicator regardless of the contents of the input string.

.NUM — A test for a numeric literal sequence which represents a fractional real number. This number is directly related to the floating point type of numbers and must contain a decimal point.

.INT — An integer number test on the input string.

.TYPE('NNYYY') — Actually a cross between syntax and semantics, this routine references the labels to find the label contained in the current string. When found, a comparison is made to determine if the type specified by YYY is correct. The flags NN determine if the type may be a default type and how much space is to be allocated for one element.

.TO('XXX') — A general search test which searches the input string until an XXX is found. If the input string runs out before the search is satisfied, a "false" return is made. This test is useful for such things as comments.

.ERR('NNXXX...X') — The primary method of producing error messages for the user language. If the previous test has failed, the string NNXXX...X is printed as an error message. The digits NN are retained as a completion code. The maximum NN is used as a return code at the end of the compile step.

.SR — A test for a character string enclosed in apostrophes. The test leaves the string pointer pointing at the terminating apostrophe.

Semantics for code generation

Two types of semantic functions are contained in META PI. These two elements are interconnected since the semantic operations are always contained within the semantic commands. Each semantic command generates an element of the final object program.

Semantic Commands

.OUT(...) This command causes the current contents of the output area (a temporary area where code is being created by the user's compiler) to be converted to internal form. This output area serves as a staging area for output in intermediate forms. The output is formed by the semantic operations contained within the parentheses. Three actions are possible depending on the structure of these operations.

1. If the first character is not a letter or a digit, then the rest of the output area is copied directly into the code area.
2. If the fourth character is a period or a blank, it produces actions which assume that an index register based address is required. The symbolic address following the period or blank will be looked up in a label table and an actual internal address generated for the variable.
3. If the above two cases fail, the character string is assumed to be machine code and it is converted directly into the code area.

A series of operations may be put into a single .OUT command by separating the operation sets with commas.

.LABEL(...) The current contents of the output area is used as a search argument in the label table. If the label already exists, the current core location is filled in. If the label does not exist in the table, it is entered. If the label is already defined an error results.

.IGN(...) The current contents of the output area are ignored. This command is necessary because many of the semantic operations have side effects such as releasing registers.

.NOP(...) This command produces the effect of the semantic operations but no more. The results of the semantic operations will be left in the output area.

.EXREF(...) The symbolic name in the output area is recognized as an external reference. A four-byte field is reserved for the address, and information is stored to produce the reference later.

.DECK A command which uses as input all of the information stored in 1) the label table, 2) the external reference table, and 3) the working core to produce an object module which represents the compiled program. Normal save area conventions are automatically generated at the beginning of the object module, but not at the return points.

.TSET A generalized declaration primitive, .TSET receives its input from 1) the top of the save stack, and 2) the output buffer. The output buffer must contain a type declaration of the same form as required by .TYPE. The top element of the save stack must contain the variable name followed by its array bounds, if any. The array bounds must be of the form:

,[lower bound:] upper bound

where the brackets indicate optional items.

Semantic Operations

The semantic operations are used to generate code in the output area. This code is in intermediate forms which must be converted to an internal form. These operations are not allowed to alter the input pointer or the truth indicator. A pointer is maintained to remember the next available location in the output area. This pointer is updated after some of the semantic commands. The semantic operations are:

'CCC...C'	Suffix the string between the apostrophes to the output area.
*	Suffix the current input string to the contents of the output area. This operation is usually used in conjunction with a successful .ID test.
S	Save the current contents of the output area in a pushdown list and push down the list.
R	Restore (suffix to the output area) the top of the pushdown list and pop up the list.
I	Ignore the top element of the pushdown list (pop it out).
X	Swap (exchange) the top two elements in the pushdown list.
*1	Generate a globally unique four byte character string beginning with the character #. This string will be locally constant and provides a convenient way to label and reference locations in the generated code.
*2	A second globally unique, locally constant variable like *1.
A2	An alignment operation which forces the next operation to occur on a half-word boundary (but not a full-word boundary) by filling in "no operation" codes.
A4	Same as A2, but for full-word boundaries.
W1,W4,W8	Work space operations to acquire space of length one, four, and eight bytes, respectively.
R1,R4,R8	Work space operations to release space of length one, four, and eight bytes, respectively.

A set of semantic routines exist for the use of the general purpose and floating point registers. A type of pushdown list is maintained at compile time for both types of registers. There are eight general purpose and four floating point registers available to the user. If more registers are needed, coding will be automatically generated to save and restore registers. This save and restore operation is a side effect of the following semantic operations.

OF	Output the current general purpose register.
O	Output the current floating point register.
P	Output the previous general purpose register.
P2	Output the previous floating point register.
+	Output the next free general purpose register and make it current.
+2	Output the next free floating point register and make it current.
-	Output two general purpose registers. The first one is the previous register, the second is the current register. Upon completion, the previous register is made current. This operation is to take advantage of the register to register operations.
-2	Output a pair of floating point registers. The action is the same as the semantic operation for general purpose registers.

META Syntactic Commands

A final class of commands, the META syntactic commands are added to control the internal operation of META PI. These commands aid the user in producing efficient compiler code.

.LATCH(name)	This command causes the routine in parentheses to be called. In addition, pointers are kept so that if an exit to the error routine occurs, backup will be affected.

C	This operator may occur anywhere a semantic operator may occur. It causes the last .LATCH operation to be ignored if an error occurs.
.CLAMP	This operator may occur anywhere a semantic operator may occur. It directs the compiler to ignore all preceding .LATCH operations.
.SAV(...)	The semantic operations represented by ... are performed and the output buffer is then entered into the pushdown list, and the list is pushed down.
.RES	The top element of the pushdown list is restored to the current string and the list is popped up.

The META PI compiler-compiler is sufficiently versatile to describe itself. As a final definition of the compiler-compiler, its META PI definition follows:

```
ccst   :=  .ID .LABEL(*) ':=' .NOP(C) ccx2 $('|' .OUT('078A')
           ccx2) ';' .OUT('07FA');

ccx2   :=  $cco ccx3e .OUT('58E.' *1) .OUT('077E') $( cco |
           ccx3e .OUT('477..ERR')) .LABEL(*1);

cco    :=  ('.OUT(' | '.IGN(' .OUT('92FF900A')) $cco1
           .OUT('05E9') $(',' $cco1 .OUT('05E9')) ')' |
           '.LABEL(' $cco1 ')' .OUT('45E..LABE') | '.DO(' $(.SR
           .OUT(*) ''' ) ')' | 'OPT(' ccx1 ')' .OUT(0420') |
           '.SAV(' $cco1 ')' .OUT('45E..SAV') | '.NOP(' $cco1
           ')';

cco1   :=  ccosub .OUT('45E..' *) | 'C' .OUT('45E..LATX') | .SR
           .OUT('05E4') .OUT('#' #* ':') ''' ;

ccosub :=  'R4' | 'R8' | 'W4' | 'W8' | 'R1' | 'W1' | 'A2' |
           'A4' | *1' | 'B' | 'I' | '+2' | '+' | 'S' | '-2' |
           '-4' | '-' | 'X' | '*2' | '0F' | '0' | '*' | '#' |
           '.'.ID ;

ccx3e  :=  ccx3 ('.ERR(' .SR .OUT('05E2') .OUT('#' #* ':' )
           ''' ')' | .EMPTY);
```

```
ccx3  :=  .ID .OUT('41E.' *) .OUT('0503') | .SR
          .OUT('45E..TEST') .OUT('#' #* ':') ''' | '(' ccx1
          ')' | '.EMPTY' .OUT('0420') | '$' .LABEL(*1) ccx3
          .OUT('58E.' *1) .OUT('078E') .OUT('0420') |
          '.LATCH(' .ID .OUT('41E.' * , '450..LATCH') ')' |
          '.TYPE(' .SR .OUT('45E..TYP' , '#' #* ':') ''' ')' |
          '.' .ID .SAV(*) ('(' $CC01 ')' | .EMPTY)
          .OUT('45E..' R);

ccx1  :=  ccx2 $('|' .OUT('58E.' *1 , '078E') ccx2)
          .LABEL(*1);
```