NGR-21-002-197

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND

A Report on Computer Performance Evaluation Techniques

by
James Yeh

ACKNOWLEDGEMENT

The author wishes to thank his advisor, Dr. Jack Minker, for his gentle criticism, encouragement, and continuing guidance in the development and execution of this paper, and for his patience in correcting the English.

The author wishes to make further acknowledgement to Mrs. Sarah Crooke for her many enlightening discussions and significant suggestions.

The author is also grateful for the financial assistance provided during the period of this research by the National Aeronautics and Space Administration under Grant NGR21-002-197.

ABSTRACT

This thesis is a report on performance evaluation techniques of computer based data processing systems. An overview of the analysis techniques and a review of some evaluation techniques are described first, followed by descriptions of three analysis techniques developed in this study. Experimental results using these techniques are given. Finally, a summary and a bibliography are provided.

TABLE OF CONTENTS

1. INTRODUCTION

The need for computer performance evaluation arises at the time that a computer is used for business and scientific data processing. As computer systems increase in size and complexity, it becomes not only more important but more difficult to measure performance. The original problem for implementing a computer system is: 'What configuration of hardware/software/personnel is required to perform the anticipated data processing tasks and to generate useful outputs within a required response time?'. Today a vast range of different hardware/software are available, with a large variety of internal capacities, capabilities, and features, and with a wide and complex range of peripheral functions. Most contemporary computers have the capability for a concurrent processing between peripherals and individual programs. The object then, is to determine which configuration is 'optimal' for a particular application. That is, the total capability of a computer must now be measured in terms of both time and space requirements. In order to make the evaluation meaningful, standard measures of system capabilities and techniques for analyzing systems and assigning weights to these measures must be employed. The major evaluation goals are: (1) to provide precise measurements of implementation costs, processing requirements, and response or turn-around times for feasibility analysis; (2) to provide the ability to accurately specify performance requirements for adequate and optimum computer system selection; (3) to provide a predictive tool to allow the programmer to optimize the capability and capacity utilization of the programs; (4) to provide the ADP Management with the ability to measure the change and/or peak workload.

Due to their complexity, computer systems are very difficult to measure. In many cases, computer systems may run each user job correctly, but may still be grossly inefficient in using the computing power of the system. Sometimes these 'performance bugs' are more frequent and more serious than logic bugs. The performance bugs have no obvious symptons, except that they decrease the efficiency of the system. A flow which degrades the average response time by 20% may not be recognized immediately. It frequently takes a factor of two before the user realizes something is wrong. Detailed measurements from a quantitative study on the behavior of that system are perhaps the only way to locate those bugs and to examine the inefficiencies that may exist in the structure and utilization of the system. Also, a good quantitative understanding of the performance of an existing system or program is necessary for designing a new and better system or program.

The available evaluation techniques are applicable as a function of the level of analysis required, so that no single technique can serve as a satisfactory performance measurement of a total computer system. An examination of the assumptions that underlie some of the presently available techniques is essential. It is also important to review some of the evaluation techniques with their results. The emphasis in this paper is on the software monitor technique. This technique provides a scheme for obtaining data from 'inside' the operating system as it is running. These internal performance data not only reveal the exact sequences and patterns of events which occurred during execution, but also assist in locating implementation problems. The performance data can also be used as feedback to a system designer and/or provide realistic calibration data input to simulation models.

2. OVERVIEW OF COMPUTER SYSTEMS PERFORMANCE ANALYSIS TECHNIQUE

In the classical scientific method, a complete analysis involves a combination of a theoretical approach and an empirical measurement. The theoretical analysis may be handled with a mathematical or simulation mode. The empirical experiment is designed to gather statistical data for testing the theory. Neither the theory nor the measurement alone is sufficient.

2.1. Theoretical Analysis Techniques

2.1.1. Analytical Modeling Technique. Analytical modeling techniques describe the general characteristics of a computer system (or subsystem) in terms of mathematics. A set of variables are defined to represent the inputs, outputs, and internal states of the system. A set of equations describing the relations between these variables are formulated. By varying the given inputs, one can predict the behavior of the computer system under different situations. Analytic models provide a means of thoroughly understanding specific critical aspects of a computer system. These results are generally applicable to system design and algorithm formulation. A limitation of analytic modeling is that the scope of the modeling is restricted to a subsystem of the total system. In general, attempts to describe a total system mathematically result in a complex unsolvable model or the design of a complete model with significant detail is not possible.

2.1.2. Simulation Model Analysis Technique. A simulation model may be used to represent some particular function of a computer system or subsystem. If constructed with sufficient accuracy the model can reflect the effects of various changes as if made in the original system. Thus, it enables the original system to be studied and analyzed by studying and analyzing the behavior of the simulation model. Today, there is perhaps no single technique more valuable than simulation for use in evaluating systems. Several simulation languages have been developed to facilitate the expression of the components and logic of complex systems to be simulated. Two general purpose system simulation languages are GPSS (General Purpose System Simulator) [58], and SIMSCRIP [50]. Languages developed especially for computer system simulation are CSS (Computer System Simulator) [46], and SSS (System and Software Simulator) [34]. Special purpose computer hardware simulation languages include CDL (Computer Design Language) [33], and HARGO (Hardware Oriented ALGOL Language) [39]. Much simulation work has also been done using general programming languages such as FORTRAN, ALGOL and PL/1. Perhaps the most critical factors in simulation are the unavoidable assumptions made concerning the behavior of variables within the real system. The results produced by simulation are no better than the assumptions underlying the construction of the model. Several simulation models and the results of computer systems have been reported. Two of the most interesting simulation models available are CASE and SCERT [32, 35, 40, 41, 47]. These models are reviewed in Section 3.1.

2.2. Empirical Performance Analysis Technique

Empirical measurement is used in an operating computer system to determine hardware, software, and user characteristics. It provides information about what goes on inside the system as well as the system throughput, capacity, and the characteristics of system load. There are two major empirical measurement techniques: Analytical Measurement and Benchmark Measurement. A comparison of these two techniques is shown in Figure 2-1.

2.2.1. Analytic Measurement Technique. The analytic approach to computer system measurement involves the insertion of hardware and/or software probes into the system to allow measurement and recording of the system's most subtle behavior. The application of analytic measurement can be divided into four general categories: Program Analysis, Supervisor Analysis, System Analysis, and System Research. Program Analysis may involve debugging and understanding inter-program relationships such as those found in the process of debugging a large data base system, program tracing to pinpoint performance bugs, uncovering communication problems, and performing introspective analysis. Supervisor Analysis falls into two classes: Assessment and Evaluation. In the first, the problem is to find and to measure the variables affecting the executive's environment. The second class involves evaluating the executive response. System analysis is concerned with the suitability of the system for fulfilling its intended purpose. Essentially, it provides an answer to the question: To what degree, and in what manner, has the man/hardware/software complex affected its environment?. System Research involves experimenting with a computer system by observing and measuring the effects on the system as a result of controlled changes deliberately induced. A comparison of the hardware monitor, the software monitor, and the instruction trace methods are shown in Figure 2-2.

2.2.1.1. Hardware Monitor Technique. Hardware instrumentation involves attaching electronic probes to components of the computer which are to be monitored. The probes are capable of generating a signal upon detection of any voltage change presumably caused by some known computer activity. The probes are attached to a hardware device that can logically combine the signals and record their frequency, value, and durations. Several papers have reported on the use of hardware monitors [60, 64, 65, 66]. Some of these techniques along with other commercial products are discussed in Section 3.2. In most cases, hardware monitoring is used to determine system operating characteristics such as I/O waiting time, overlap of activities, resource utilization, and idle time. The advantage of hardware monitoring is that it imposes no interference upon the object system. The disadvantage is that it needs a special hardware device, and only a limited subset of the total system data and relationships are accessible to the probes.

2.2.1.2. Software Monitor Technique. Software monitoring involves modifying the system software so that the system's operation may be interrupted at any point to permit access to pertinent data and intra-system relationships. The disadvantage of software monitoring is that it results in some system degradation as a function of the frequency of data collection and recording. To compromise between resolution and system degradation some design principles and implementation techniques have been given in [71, 73]. Some of the results obtained using software monitoring are reported in [12, 71, 72, 73, 75, 77].

The software measurement technique can be subdivided into the following areas of study: Instrumentation, Measurement, Recording and Reduction.

1. Instrumentation

   The software instrumentation is a scheme to access the internal data as well as intra-system relationships. There are two major techniques suitable for different purposes; the stand-alone package is applied on the sampling measurement, and the integrated system is used for continuous analysis.

2. Measurement

   A. Sampling Measurement

      This technique can provide a frequency distribution which describes the activity of a program. It is very useful in selecting areas of a large program for analysis and improvement. According to Cantrell and Ellison [72], 'If an executing program is frequently interrupted according to some random or periodic time schedule which is known to be statistically independent of any natural execution pattern in the program, then the frequency with which the interrupt location falls within a particular instruction sequence is proportional to the total time spent by the program in executing that instruction sequence.' The formula to compute the proper sampling rate is described in detail in [70] and the clock interrupt feature of the computer is used to control the sampling rate.

   B. Trace Measurement

      When the analyst is concerned with the identification and the order of the events in a system function, trace measurement is used. It results in a time-oriented listing of the occurrence of the selected events. This technique is particularly suited for the debugging and the analysis of intra-system behavior.

C. Accounting Measurement

The standard accounting data provides resource usage information such as CPU time, channel time, peripheral device time, memory usage, amount of terminal time, and volume of file storage. The data available from standard accounting files are frequently sufficient to determine the resource utilization.

D. Playback Measurement

This technique which provides the ability to recreate a system or subsystem's operation for interactive study and experimentation was successfully used in the sage system and has been well described by Sackman [78]. It is also in MULTICS, which was described by Saltzer in [79].

3. Recording

The recording component of the software monitor causes significant problems due to the large volume of data which must be transferred from main memory to secondary storage. Data compression, pre-analysis, data selection, and interleaving techniques are used for reducing the data volume and/or the time required for recording.

Reduction

Once the data selected for measurement have been recorded, reduction is necessary to make them legible and meaningful to a human analyst. The reduction operations may be required so that the data are time-sequenced or event sequenced, converted to meaningful units, and presented as summary counts, graphs or histograms.

2.2.2. Benchmark Measurement Technique. A benchmark is a routine which is run on a number of different computer configurations to obtain comparative throughput performance figures regarding the abilities of the various configurations to handle specific applications (Joslin, 1966). The benchmark methodology involves the specification and execution of instruction mixes, and kernels or tasks to provide the comparative measurement. An instruction mix is the weighting of each instruction execution time by a coefficient which represents the frequency of occurrence of the associated instruction. A kernel is a block of code which constitutes a basic function. A task is the type of work requested by the user. There are many measures of software capability that may be emphasized to varying degrees according to a specific user's need. These measures are basically concerned with time and utility to the user. They include programming time, checkout time, compilation time, execution time, I/O utilization, product economy, secondary storage utilization, hardware growth flexibility, I/O and CPU synchronization, facility maintenance cost, operator intervention, machine independence, documentation, and programmer training.

The stimulus measurement technique used in evaluating time-sharing systems is an outgrowth of the benchmark analysis technique. It involves applying a controlled set of stimuli to the black box so as to activate its functions and then observe its performance. The purpose of the measurement is to provide a measure of the throughput and the response time by measuring the effect of certain key functions upon the overall system's behavior. These functional variables must be stimulated in a controlled and measureable manner by the benchmark programs. Each of the programs provides one or more stimuli in controlled quantities and determines the effect of the stimuli upon the system in terms of its own performance. A total system (man/hardware/software) may be viewed as a 'black box' containing certain known functions which can be activated by external stimuli. The stimuli consist of computation, terminal interaction, paging, I/O, swapping and resource allocation activities. The effects are measured in terms of the throughput and the response time. The throughput is a measure of the volume of work performed by the system. The response time is the speed with which the system responds to an interactive user. By controlling the stimuli and observing their effects, inferences can be made about the behavior of both the system's functional components and the users characteristics.

The stimulus measurement technique may be used in three different environments in which the system's behavior is to be evaluated: A stand-alone environment, a benchmark environment, and a real world environment. A stand-alone environment is used to determine the best throughput and response time which a given configuration of hardware/software could ever deliver. This measure of maximum performance is used to evaluate the cost/effectiveness of a proposed modification to the system and to determine the performance dgradation introduced by a time-sharing system. A benchmark environment represents a typical user population which makes a typical set of demands upon the system. Modifications to system functions such as job scheduling, swapping and demand paging logic, which may affect more than one class of user, may be evaluated quickly both for effectiveness and for correctness of operation using benchmarks. A real-world environment is used to measure the service behavior given to one pseudo-user of known characteristics under real-world conditions. This technique involves running a benchmark program as the pseudo-user when the system has an almost full complement of real users. The major problem of the stimulus measurement technique lies in establishing equivalent environments within computer systems which are to be evaluated. Further, it does not provide sufficient data for a clear insight into the system's operation.

Factor	Analytic Measurement	Stimulus Measurement
Development cost	High. Requires personnel with sophisticated and detailed knowledge of executive routines. Testing requires stand-alone computer time. Errors may affect all users.	Low. Personnel with little experience can produce the benchmark programs. Testing can be done under time time-sharing. Errors affect no one else on the system.
Operating cost	Increase in system over-head.	Require some stand-alone time. Usurps a terminal and increases system load under time-sharing.
Measurement capability	Detailed Data on the system behavior and interactions. Measure-ment include sampling, accounting, tracing and playback.	All behavior is measured in terms of response time and throughput.
Knowledge of results	Usually require extensive offline analysis. Con-siderable statistical and analytic skill is requir-ed.	Result are online, simple and immediate. Extended analysis is usually not required.

Figure 2-1. Comparison of the Analytic and Stimulus Measurement Techniques.

Techniques / Attributes	Hardware Monitor	Performance Data Recording	Instruction TRACE
Degradation on measured system	None	Low	Very high
Level of detail recorded	Low	Medium	Very high
Special hardware required	Yes	No	No
Cost	High	Medium	Low
Flexibility	Very low	Medium	High
Purpose	Overall system analysis	Overall system analysis	Implementation analysis

Figure 2-2. Comparison of Measurement Techniques.

3. REVIEW OF SOME PERFORMANCE EVALUATION TECHNIQUES

3.1. Simulation Model Analysis Technique

SCERT (System and Computers Evaluation and Review Technique) and CASE (Computer-Aided System Evaluation) are simulation program packages. Both have been designed to accept the definition of a computer system's parameters and to build an application 'workload model' and a 'configuration model'. Both simulation packages maintain a library of hardware/software performance factors for a wide range of digital computers. The simulation can extract the appropriate hardware/software factors for all the components in any one configuration. With this information, configuration models are built which satisfy the performance requirements. Then during the simulation phase, it simulates the response of each of the 'workload models' against the 'configuration models' of each of the selected hardware/software complexes. The results of this simulation are projected in terms of cost, time, memory and manpower requirements. Since the functional structure of SCERT and CASE are very much alike. A block diagram is shown in Figure 3-1. Only SCERT was chosen and is described in detail in this report.

SCERT consists of four major components: Definition Language, a Factor Library, Simulation Programs, and Output Reports. The Definition Language is used to define the application system and the Hardware/Software complex to be simulated. The Factor Library contains the characteristics of the hardware/software items such as cost, performance, and technical specifications. The simulation programs perform the necessary processing to accept the input definition data and create the output reports. Output reports consist of several different types of reports. These may be broken down into four major categories: Summary Reports, Computer Complement Report, Real-Time and Multi-Programming Analysis, and Detailed Report.

From the acceptance of the input data to the preparation of the output report, a SCERT simulation involves five phases. Input to Phase 1 consists of a series of definitions outlining the workloads and computer processing requirements of the system to be simulated. The Output of Phase 1 is the model of the application system. Input to Phase 2 is a series of definitions which outline the hardware components, the software packages, and an environmental definition in which the hardware/software configuration is to be operated. The output of Phase 2 is a model of the hardware/software configuration complex. In Phase 3, the models created in Phase 1 and 2 are combined with each system/hardware/software combination, and the raw timing figures are computed. Phase 4 calculates the run time for each configuration combination by considering simultaneous operations allowed by the hardware, as well as any other constraints imposed upon such simultaneous operations. Results of the previous four phases are then accumulated, and in Phase 5 the output reports are created.

3.2. Hardware Measurement Techniques

Within the normal standard hardware features of a digital computer, such function as address stop switches, trap transfer modes, and normal error-faulting procedures are sometimes used for measurement purposes. In addition, some special hardware devices have also been developed and added to systems so as to perform hardware monitoring of a computer's performance. Devices can be attached to a central processor so as to passively examine each instruction as it is executed. Hardware monitor devices have built-in counters and self-contained output devices to record the occurrence of any given data pattern.

3.2.1. IBM 7094 Hardware Measurement Technique. This device is designed to record information from the 'CPU' while the 'CPU' is processing data. The recorded data is then used to analyze the basic nature of the program and to measure the performance of the hardware. The hardware measurement device consists of a control unit, a control panel, and an IBM 279 VI tape drive. There are three internal sections of the control unit: (1) an input unit, which contains 40 lines from the monitored 'CPU', six 24-bit data buffers, and one comparison unit. Of the 40 lines, there are 24 data lines which are used to transfer 20 bits of the contents of the instruction counter, and 4 bits specifying the channel in-use to one of the data buffers; 15 selector lines which transfer the 15-bit op-code to the comparison unit; and 1 stroke line which contains the status of the input lines. The comparison unit compares the 15 selector input lines with each of five sets of switches manually set by the operator from the control panel. Data are recorded if there is a match between the 15 selector lines and one of the five sets of switches; (2) an encoding unit and assembly register, which encodes the 24-bits of data to a variable length string, packs the string into 6-bit groups, and transfers the string to the output buffer one group at a time; (3) an output unit, which contains eight 6-bit output buffers and one tape controller. A block diagram of the operation of the device is shown in Figure 3-2.

3.2.2. IBM System/360 Hardware Measurement Technique. TS/SPAR (Time-Sharing System Performance Activity Recorder) is a hardware-measuring device used to collect performance data for measuring the dynamic operations of an information handling system. It can be used to measure the external effects of internal software and hardware operations, and to measure the internal operational characteristics of software or hardware units. It can also be used to count the frequency of an event, to clock its duration, and to record the gross time. A block diagram of TS/SPAR is shown in Figure 3-3. Electronic counters within the device provide accumulative storage for up to 48 measurable parameters of 3 decimal digit length.. Mechanical counters are activated when overflow occurs from the electronic counters. Comparators are used to dynamically monitor data paths in the interface and to compare them with fixed values indicated by switch settings. These switches are used to indicate to the monitor a unique address, an operation code, or some contiguous memory locations. The sequencer can be used to detect any three-event sequence. An event may be a reference to a real or virtual memory address, an instruction counter, an op-code, a control signal, etc. The time interval between the occurrence of events is not considered, only the event sequence is of interest. The plugboard receives the interface signals and transfers the data and control to the various functional areas in the recorder. The logical circuitry is accessible from the plugboard to logically combine interface signals so as to form complex events or to generate control signals. Input to TS/SPAR is through a specially engineered interface which can handle 256 predetermined signals and strokes. These interface signals reflect certain key states (internal or external) of the system to the recorder.

3.2.3. CDC 6600 Chippewa Hardware Measurement Technique. The Lawrence Radiation Laboratory uses a PPU (Peripheral Processor Unit) as a programmable hardware monitor to record and to analyze the activity in the CDC 6600 central processor and other peripheral processors. Two monitoring routines, Mr. See and Mr. Eye, are used. Mr. Eye gathers informations on 'CPU' activity, central memory utilizations, channel activity, PPU activity and control dispositions. Mr. See furnishes data on the disk utilization and the job profiles.

3.2.4. Univac 1108 Hardware Measurement Technique. A Univac 1108 is used to measure the performance of another 1108 system. The hardware measurement system uses a special hardware device interface as a recording processor to gather live data. (See Figure 3-4), it contains a hardware monitor, data collection software, and data reduction software. The monitor creates and records data each time a jump instruction is transferred to a drum via two large core storage buffers areas. When the drum is filled, the data are transferred to tape. A special data reduction software package reduces the data into either graphic or statistical form to provide a perspective of the performance analysis of the monitored equipment.

3.2.5. Some Commercial Hardware Monitors. CPM 11 (Computer Performance Monitor) [59], CPA 7700 (Computer Performance Analyzer) [61], and SUM (System Utilization Monitor) [62, 63] are some of the commercial hardware monitors. In general, the hardware monitor consists of three logical elements: probe lines to convey statistical data sensed in the computer being monitored, accumulators to temporarily store counts or timing signals, and a computer compatible tape transport to record system performance data for later analysis. Most of the commercial products also provide a data reduction and analysis program which reduces the accumulated data and prepares tabular and graphical reports representing system performance. The functional structures of the available commercial hardware monitors are very much alike.. CPM has been chosen as representative of these monitors and is described in detail here.

CPM consists of an operator console, a logical unit, a control panel, and a tape drive. On the operator console, there is one ten-position decimal visual register which is used to display any one of the sixteen counters or the clock, and ten function select switches that control the main function of the CPM. The logical unit consists of a real-time clock which records the time of day in 100 US, 20 measurement probes to sense the various functions throughout the monitored system, and 16 counters, each with 10 decimal place registers to measure the activity of the monitored functions. The measurement probes are attached to individual circuit pins in the computer system which are active when a particular event occurs. The counter may be used either to measure the length of time a function is active (Time Duration Mode) or to count the number of times an event occurred (Event Count Mode). The real-time clock is incremented every 100 US and overflows to zero at 24 hours. The clock is used to provide a measure of total elapsed time, as well as to allow direct correlation of the measure of total real-time, and console logs. The control panel provides the operator with control of the probe counter assignments, the counter operating mode, and the combinatorial logic functions. It consists of 26 and/or elements, 2 hexadecimal recorders, 16 fanouts, 8 latches, 16 inverters, 60 probe exit hubs (2 true exit and 1 false exit for each probe), 32 counter entry hubs (1 count entry and 1 time entry for each counter), 20 clock exits (each exit has the following different durations: 1 US, 10 US, 100 US, 1 MS, 10 MS, 100 MS, 1 SEC, 10 SEC, 1 MIN, 10 MIN), and 10 function hubs. A 1200 foot reel mounted on a tape drive provides synchronous recording on 9 tracks 800 BPI, with a minimum recording internal of 100 MS. Each record written on tape is 175 characters in length. Included in each record are the contents of the clock, the 16 counters, and the settings of the five data switches.

3.3. Software Measurement Techniques

There have been several developments in the field of applying software techniques to monitor systems. Some of these developments are described below.

3.3.1. GE GECOS II,III Software Measurement Technique. The overall performance of a computer system depends on the efficiency of both the hardware/software environment and the programs which operate in that environment. The software monitoring device used in 'GECOS II' is designed to permit analysis of the system performance and also of individual programs. The system analysis includes user program accounting analysis, overhead analysis, and trace analysis. To provide for individual program analysis, that is functional value analysis, high density sampling is used. By frequently interrupting the system at random or periodic times, the fraction of the total time spent in a particular instruction sequence is found to be proportional to the number of samples taken while in that sequence. The results of the periodic sampling are used as the basis of I/O and program execution time profiles. Several software measurement techniques were applied during the development of 'GECOS

III'. Software measurement of processes internal to the system were developed. Event counters were included in all functions of the system so that they could be analyzed and studied separately. Internal system auditing was provided to check on new entries in each of the system queues, to checksum critical tables each time they are referenced, and to checksum all system files as they are loaded into core for execution. Event tracing is used to detect the occurrence of important events. Decisions made within the system are monitored and made available for subsequent analysis by recording, in a circular list, each intermodule transfer. The total data collected on function usage, queue formation, table and file manipulation, and event occurrences is sufficient to summarize system operation and performance. The total analysis uses as input, standard system accounting data, the recorded trace entries, and other parameters made available from the system.

3.3.2. IBM TSS/360 Software Measurement Technique. SIPE (System Internal Performance Evaluation) is an on-line software recording technique used to collect the data necessary to measure and to evaluate the performance of the IBM System/360 Time-Sharing System (TSS/360). SIPE is a selective, event-driven recording mechanism that operates within TSS/360. The activating mechanism of SIPE is called a 'hook'. (See Figure 3-5). Hooks have been implemented at various points throughout the resident supervisor code. Each hook includes an identifier code. Based on this code, SIPE collects the applicable data. The degradation of the operating system with the SIPE monitor is proportional to the number of times SIPE hooks are activated. It is also affected to some degree by the volume of the output data. To compromise between resolution and degradation, a selective option function (Delta-

Data-Set) has been implemented. The Delta-Data-Set is input to SIPE as a parameter at the start of a run. The given Delta-Data-Set instructs SIPE to 'turn-off' any hook or group of hooks for that run. In order to derive meaningful information from the data collected by SIPE, a library of data reduction programs has been developed. These programs convert the SIPE data to a simple or elaborate form for use in performance evaluation, system analysis and debugging as requested by the analyst. A functional diagram of the interface between TSS/360 and SIPE is shown in Figure 3-6.

3.3.3. IBM OS/360 Software Measurement Technique. SMS/360 (Systems Measurement Software) is a software package developed by Boole and Babbage, Inc. Two componenets of the SMS/360 described below are the PPE-2 and the CUE-1 components.

The PPE-2 (Problem Program Efficiency) component is concerned with the efficiency of the user's problem program. The output of the PPE provides the distribution of CPU and I/O time spent by the user's program. The PPE consists of two elements: The Extractor Program and the Analyzer Program. The Extractor Program randomly samples the problem program during its execution and collects statistics for later analysis. Each time the extractor records a sample, one of two events has taken place, either the instruction address falls within sample bounds, or a SVC (Supervisor Call) has been invoked from within the sample bounds. The analyzer uses the collected data to generate reports which indicate where and how the program spends its time and how the program is balanced between being computer bound and being input/output bound. The reports generated include a number of tabular displays and one graphic display called the Histogram.

The CUE-1 (Configuration Utilization Efficiency) component is used to aid in maximizing system throughput by determining the configuration utilization and by showing specific hardware/software relationships which contribute to configuration utilization. CUE is also divided into two programs, the Extractor and the Analyzer. The Extractor collects data on hardware usage, disk head movement, data cells, and transient supervisor call routine usage. The Analyzer generates a configuration report, an equipment usage sub-report, a head movement sub-report, and a SVC sub-report. The quantitative information given in these reports can assist in locating bottlenecks in a configuration which might otherwise be overlooked.

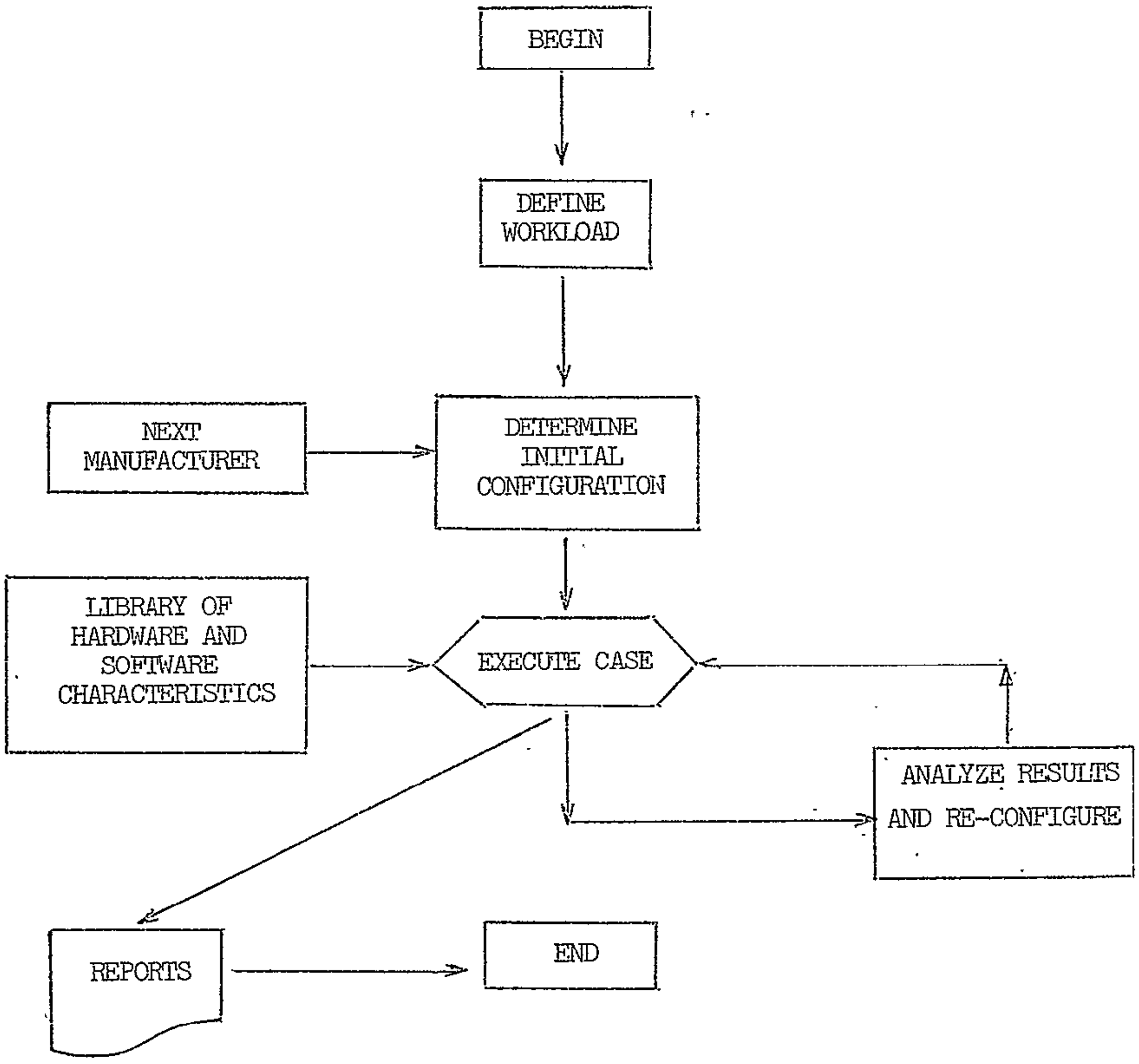
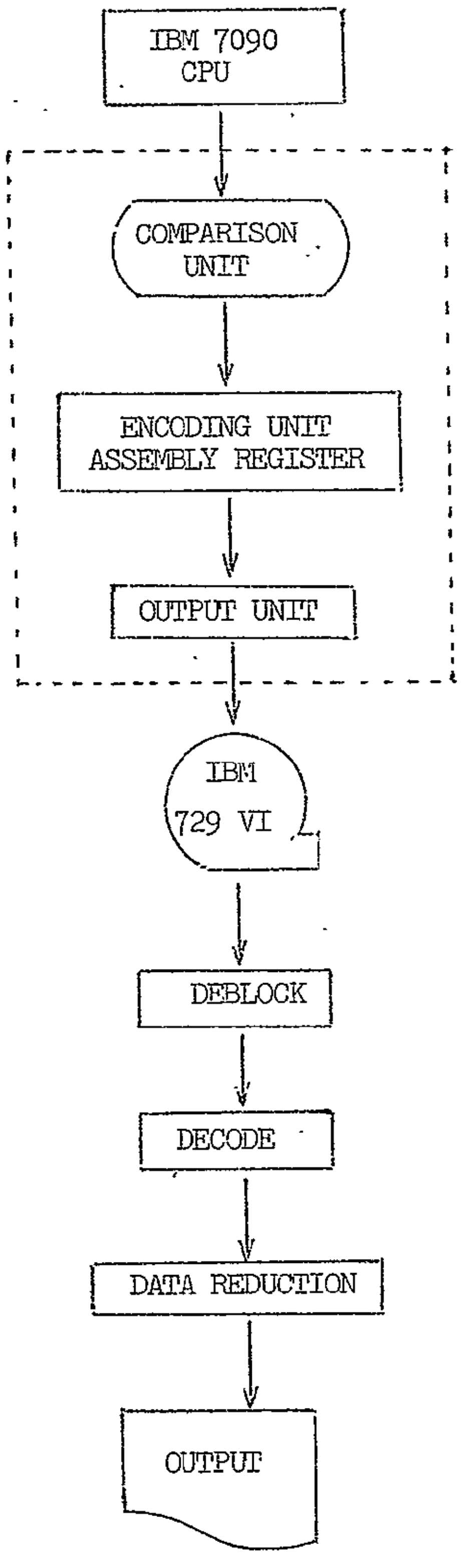

BEGIN
DEFINE WORKLOAD
NEXT MANUFACTURER
DETERMINE INITIAL CONFIGURATION
LIBRARY OF HARDWARE AND SOFTWARE CHARACTERISTICS
EXECUTE CASE
ANALYZE RESULTS AND RE-CONFIGURE
REPORTS
END

Figure 3-1. Block Diagram of the Functional Structure of CASE



Recording Machine

- Compare the 15 Selector Lines with the 5 Sets of Switches.
- Encodes the 24-bits of Data into a Variable-Length String and then Packs the String into 6-bit Words.
- Writes Inter-record Gaps between each Logical Record.
- Decodes the Variable-Length String to 24-bits of data and Adds Time Information.
- Produces a Trace-like Printout or Generates CRT Graphical Display.

Figure 3-2. Functional Diagram of the IBM 7090 Hardware Monitor Device.

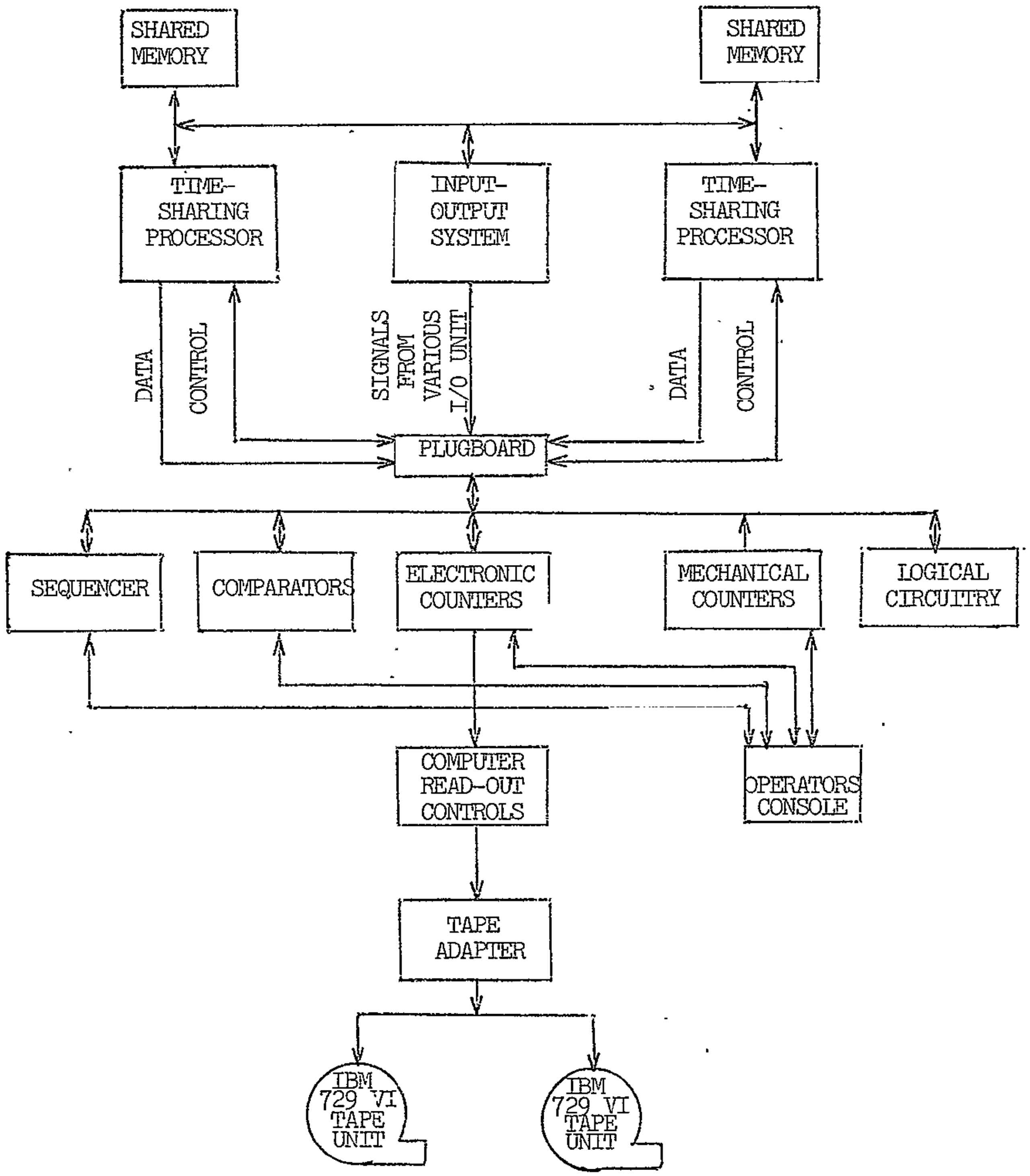


Figure 3-3. Functional Diagram of TS/SPAR

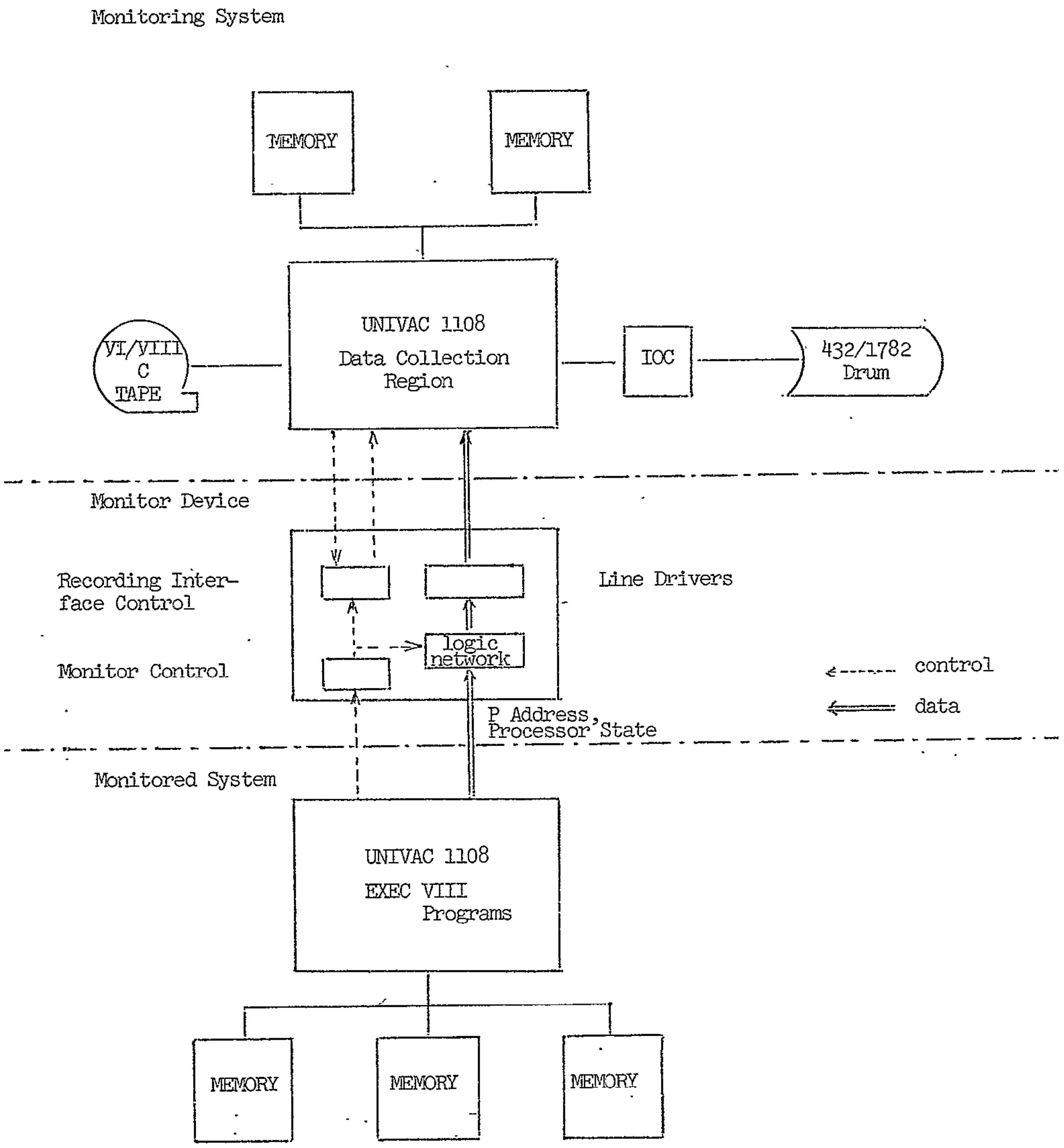
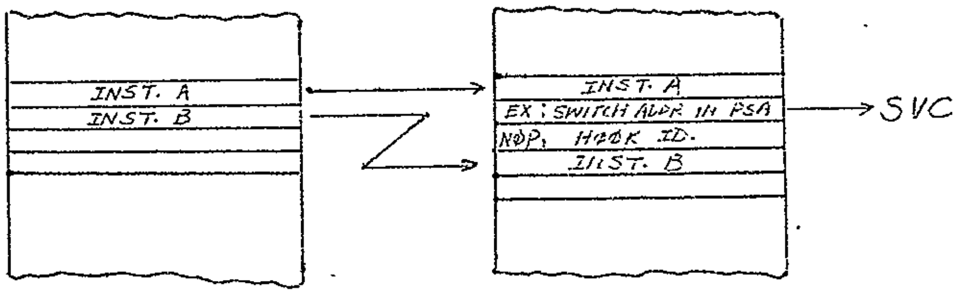


Figure 3-4. Block Diagram of the UNIVAC 1108 Hardware Monitor Device



The "Hook" Structure of SIPE

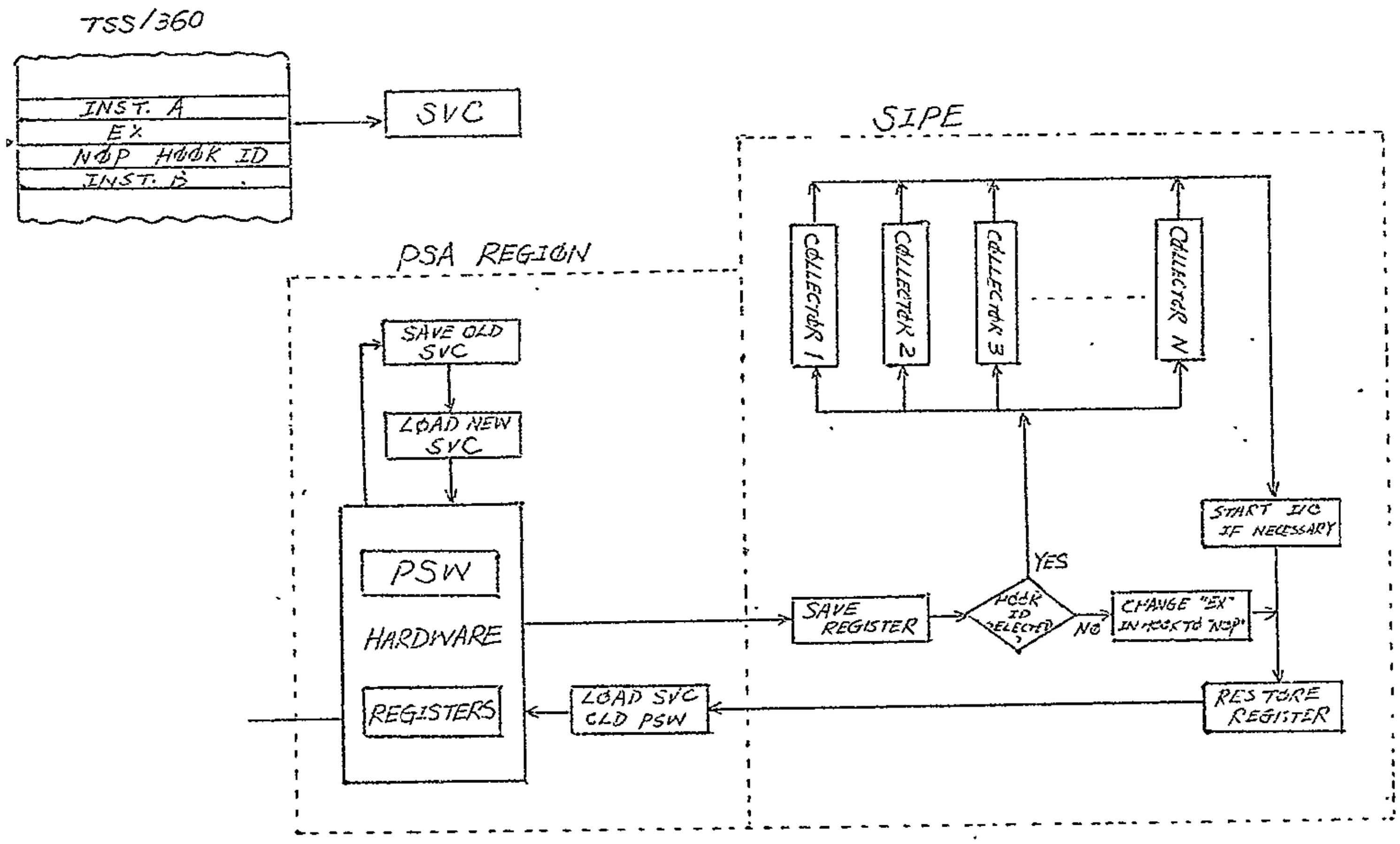


Figure 3-5. Functional Diagram of Interface between TSS/360 and SIPE

4. SYSTEM FUNCTION ANALYSIS USING SOFTWARE MONITOR TECHNIQUES

The objective of the software monitoring efforts conducted under this thesis was to develop techniques to permit the collection of data from the operating system as it was running. A quantitative study of an operating system using data on the behavior of that system is an effective approach to permit one to locate and to examine defects that may exist in the structure and utilization of the operating system. In the design of a system monitor technique, the following capabilities were desired: (1) to provide a technique that would permit one to study the logic and behavior of programs so as to define and locate significant events that occur within a program; (2) to provide a technique which would permit analysis and evaluation of the implementation of a program, so that local performance errors could be detected and possibly avoided; (3) to provide a technique to collect the applicable data of the total operating system in order that the interaction of system functions could be analyzed and evaluated; and (4) to provide a technique to continuously report the performance summary on a display or on an on-line printer at specified periods of time. To meet some of these objectives, several program were designed and implemented on the Univac 1108. These programs are described below.

4.1. Instruction Trace

TRACE is a special simulation tool which has the ability to simulate itself. It is written and developed for the purposes of studying the logic and behavior of a program. It is sometimes very difficult to obtain documentation and descriptions of system routines. This has been found to be the case with the 1108 executive routine. TRACE can provide useful information concerning the operation of a program, such as the location of the instruction, the data in the operands of the instruction itself, and the contents of all registers used by the instruction. The TRACE Routine records data at every instruction, or at selected instructions, and then prints out a step-by-step account of the behavior of the program. From the printout developed by TRACE, the programming technique of the traced program can be observed and evaluated.

In the TRACE Program, we contrive to let the machine execute most of the instructions as the instruction appears in the program. The exception is that TRACE modifies jump or conditional jump instructions before execution so as to insure that control will return to the TRACE Routine after the jump has taken place. Inside the TRACE Routine, a memory word is maintained to simulate the hardware instruction counter which points to the current instruction to be traced. TRACE copies the traced instruction into its own work area. Before execution of the instruction, a subfunction is called to analyze the op-code so as to identify whether this is an unconditional or conditional jump instruction. If the instruction is not a jump type instruction, the simulated instruction counter is increased by one and the traced instruction is executed. However, if the-instruction is a jump type instruction, the address field of the jump instruction is saved first and then replaced by a specified address. If a jump occurs, that is, the condition of the jump is satisifed, the control then goes to the specified location instead of to the successor instruction. In this fixed location, the simulated instruction counter is replaced by the saved address field. In this way the exact program instruction sequence can be traced. A general flow chart of the TRACE Program is shown in Figure 4-1. An output from the TRACE is also given in Figure 4-2.

4.2. Functional Value Analysis

The purpose of a functional value analysis is to try to improve the efficiency of a program. In analyzing a program to achieve this improvement, the payoff between the time spent in analysis, debugging, and the total possible machine time gained should be considered. A technique is described that will indicate to the user the most frequently executed code within his program. Since it is executed frequently there is a higher payoff if this portion of the code is improved.

Either in a high level language or in a machine language program, a jump instruction represents the end of a sequence of operation. Those contiguous sequential operations can be considered as a single macro-instruction. In this way, a program can be divided into several macros, each terminated by a jump instruction. By 'Kirchhoff's Current Law', the number of times the control flows out of a macro-instruction must equal the number of times control is transferred to the macro-instruction. Hence, if we record the information when a transfer is made to a special instruction (location), then we can get the exact number of times that the macro-instruction has been executed.

This functional value analysis program is formed by modifying the TRACE Routine described above by adding a sorted, linked list to record the transfer information. See Figure 4-1. After the recording is complete, another analysis routine is called to print the distribution of CPU time for each macro-instruction. An analysis of EXPOOL on the Univac 1108 that resulted from the use of ITFVA (Instruction Trace and Functional Value Analysis) is presented in Section 5.2 as a case example.

Another technique most frequently used for functional value analysis is the high dentisty sampling method which was described in Section 2.2.1.2. The advantages of using the TRACE Routine are: (1) the TRACE Routine is easily modified to permit recording information of every instruction traced or to record the trace data only when a jump occurs; (2) it provides a high level of information detail since the recorded data contains the exact number of instructions executed in each macro, and if desired, provides the exact sequence of each macro-instruction performed.

The disadvantage of using Trace is that it will greatly slow down the execution of a system. Hence, TRACE is best suited for the analysis of short input-data independent programs. An analysis of the ITFVA Routine indicates that the time required by using ITFVA within a system results in the need for an increase of 18 times the normal execution for a non-jump type of instruction, and an increase of 60 times for jump instruction.

The above disadvantage can be avoided to a certain extent by using the TRACE technique in conjunction with event counters. That is, set a count in every basic system function which is to be monitored. It is relatively simple and straight-forward to implement. According to the contents of these counters, the most frequently executed function can be detected. The procedure then is to analyze only frequently executed functions with the trace technique. This provides a very simple and useful tool to improve the implementation and efficiency of either a system routine or a user program.

4.3. System Performance Data Extract

The purpose of evaluating an operating system is to determine and to substantiate the capabilities and the limitations of that system. The problem is to find out what is going on inside the system and where the CPU spends the majority of its time. To solve this problem requires that data be obtained 'inside' the system as it is running. OSPDE (Operating System Performance Data Extractor) is developed so as to provide a software recording technique to extract internal system performance data. Such data provides the exact sequence and patterns of events that occurred during execution. It can be used as input to a simulation model to provide a realistic calibration and feedback to the system designer. This provides a good, quantitative measure of the existing system which permits pinpointing 'performance bugs' - the results of errors in programmer evaluation and judgment on performance optimization. Under this thesis, the program OSPDE has been designed, but has not yet been implemented. The structure of the data item and the data block of OSPDE is shown in Figure 4-3. The major objectives of the design were: (1) to minimize the system degradation by providing a selective option, which permits the user to be selective in the system events to be monitored at any given time; (2) to share a tape path with the system, use a variable data length structure and a data collection macro-instruction to get additional generatlity and flexibility; and (3) to use the mechanism of a double output buffer, that is, while one buffer is transferring data to tape, the other buffer is being filled with data. The CPU is forced to wait when the second buffer is full and the first buffer has not yet transferred data to tape. With this arrangement, the loss of data is possibly avoided.

4.4. Other Techniques Under Consideration

If the OSPDE recording rate is approximately one millisecond, there will be sixty thousand data items recorded every minute, and 3.6 million data items recorded every hour. It is obvious, from these huge volumes of data, that a process to reduce data must be done on a computer to give meaningful information to the user. Hence, a data-reduction and reporting routine is needed. This routine should have the capability to receive parameters from the user, to select any combination of events of the recorded data, and to output the analysis results in tables or graphs.

The Standard System Accounting Routine provides data concerning the resources and the elapsed time used by a program. The accounting data can be used to measure gross performance, and can be combined with OSPDE recorded data to summarize the overall system performance during long periods of computation time. As described above, such a technique is required to provide continuous measurement analysis to the user.

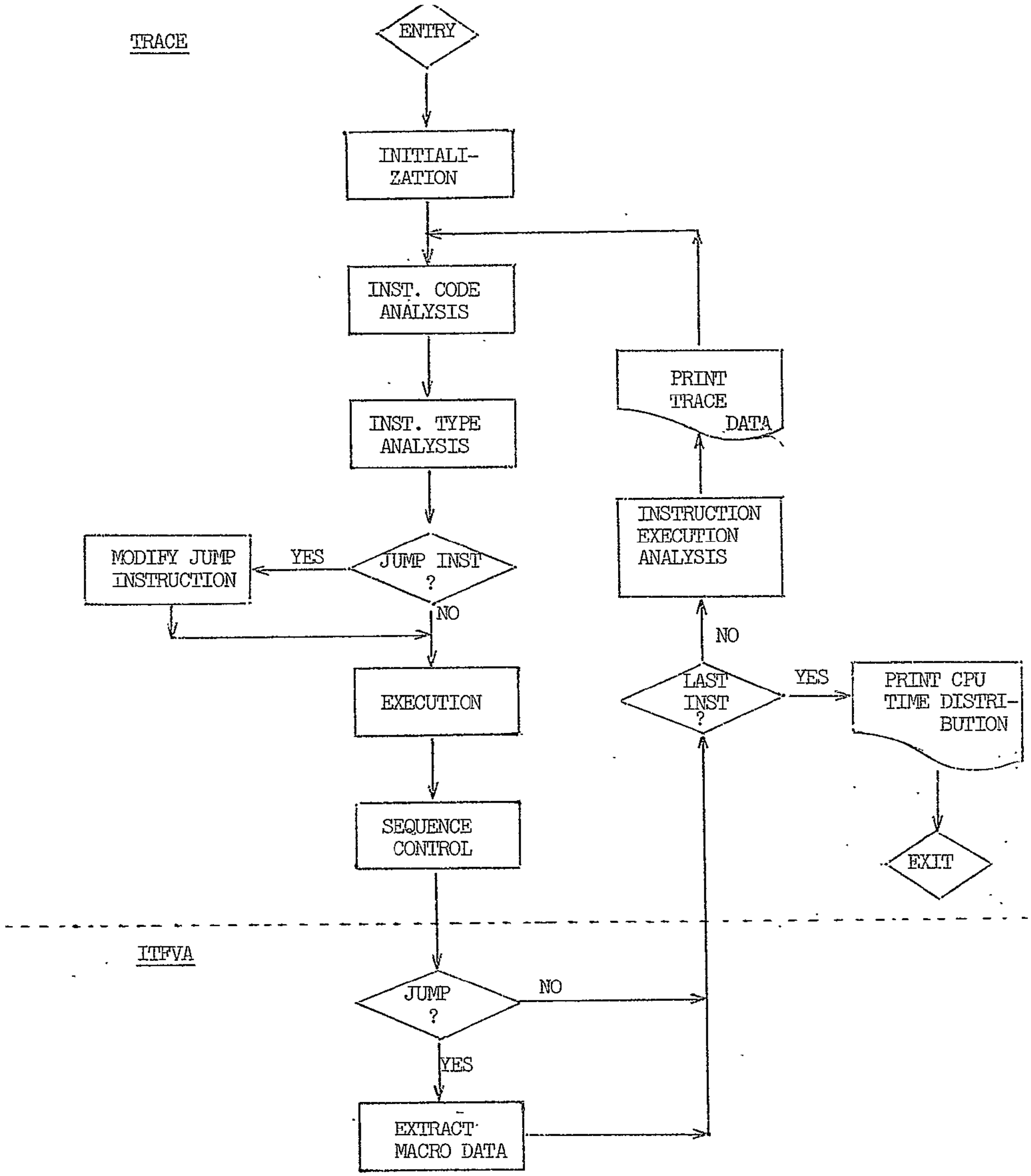


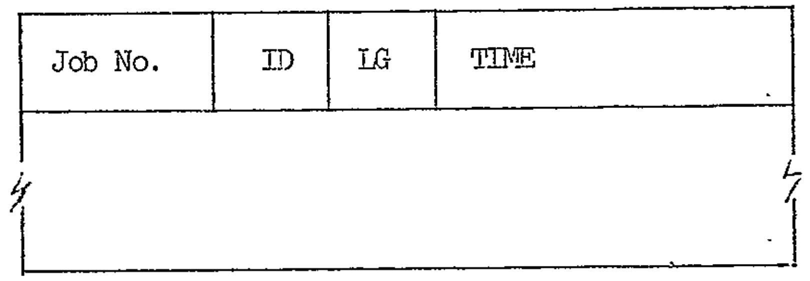
Figure 4-1. Functional Diagram of ITFVA and TRACE.

```
43115   53  02  04  14  0  043071   A  000000000017  000200000000  000000000012  777777777777  777777777777
43117   25  16  14  00  0  000001   X  000000000011  777777777777  777777777777  777777777777  777777777777
43120   72  02  05  00  0  043115      777777777777  777777777777  777777777777  777777777777  777777777777

43115   53  02  04  14  0  043071   A  000000000017  000400000000  000000000011  777777777777  777777777777
43117   25  16  14  00  0  000001   X  000000000010  777777777777  777777777777  777777777777  777777777777
43120   72  02  05  00  0  043115      777777777777  777777777777  777777777777  777777777777  777777777777

43115   53  02  04  14  0  043071   A  000000000017  001000000000  000000000010  777777777777  777777777777
43116   74  04  00  00  0  04312       777777777777  777777777777  777777777777  777777777777  777777777777

43122   27  01  14  14  0  043071   X  000000000004  777777777777  000000000004  777777777777  777777777777
43123   10  13  04  13  0  000000   A  000000000000  001000000000  011530057010  777777777777  777777777777
43124   10  16  05  13  0  000000   A  000000057010  000000011400  011530057010  777777777777  777777777777
43125   74  04  00  00  0  043130      777777777777  777777777777  777777777777  777777777777  777777777777

43130   74  13  13  00  0  043234      777777777777  777777777777  777777777777  777777777777  777777777777

43234   46  16  14  00  0  000000   X  000000000004  777777777777  777777777777  777777777777  777777777777
43235   50  13  00  00  0  043706   A  000000000004  000000011530  777777777777  000000000000  777777777777
43237   53  16  00  00  0  000011   A  000000000004  000000011530  777777777777  777777777777  777777777777
43241   27  01  15  00  0  043713   X  000000000000  777777777777  777777777777  000000000000  777777777777
43242   54  01  01  00  0  043712   A  000000000000  040075413506  777777777777  022000033000  777777777777
43243   74  04  00  00  0  043246      777777777777  777777777777  777777777777  777777777777  777777777777
 (1)             (2)               (3)     (4)           (5)           (6)           (7)           (8)
```

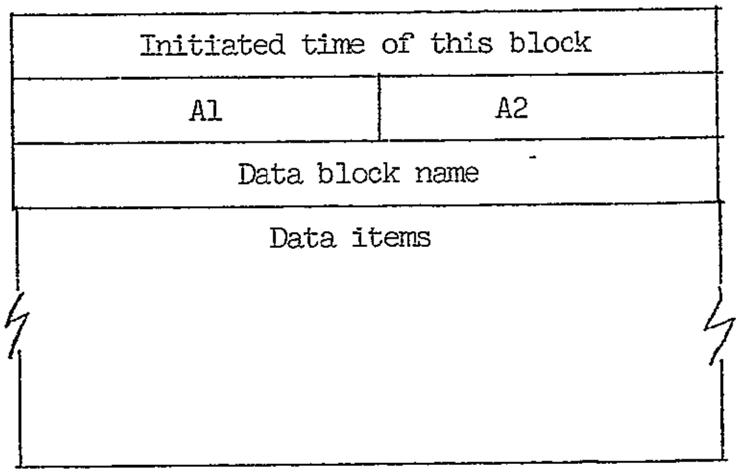
Figure 4-2. Sample Output from the TRACE Program.

1 The absolute address of the traced instruction.
2 The instruction code being traced.
3 An indicator of what type of control register is being used by the traced instruction, i.e., A, X, or R register.
4 The content of the register referenced or a code of 777777777777.
5 The contents of the next sequential register or a code of 777777777777.
6 The contents of the index register referenced or a code of 777777777777.
7 The contents of the operand of the traced instruction before execution or a code of 777777777777.
8 The contents of the operand of the traced instruction after execution or a code of 777777777777.

DATA ITEM



DATA BLOCK



ID:	Data Id
LG:	Data Length
A1:	Number of items lost in previous block
A2:	Number of words in previous block

Figure 4-3. A Data Item and Data Block of OSPDE.

5. EMPIRICAL STUDY OF THE EVALUATION TECHNIQUE OF EXPOOL

5.1. The Central Role of EXPOOL

EXPOOL is a core resident element within the EXEC VIII operating system that contains a buffer pool and two routines to maintain this pool. EXPOOL is one of the most active elements in the EXEC VIII supervisor. All system tables, queues, and control words are located in the EXPOOL buffer pool. Because of its central role, the frequency of use within the system, it was chosen for detailed analysis using the techniques developed during this study.

5.1.1. The Buffer Pool. The common buffer pool within EXPOOL is maintained in order to provide a maximum number of buffers with a minimum amount of overhead. The 'Buddy' System Storage Allocation technique is used here with permissible buffer sizes of 2**N-1 words, where $2 \leq N \leq 9$. The structure of a buffer is shown in Figure 5-1.

The EXPOOL Buffer Pool initially contains 27 blocks of 2**9 words each as implemented in the University of Maryland EXEC VIII operating system. Of the 27 blocks, 10 blocks are generated at assembly time and 17 blocks are given to the EXPOOL Buffer Pool by linking 17 blocks of no-longer-needed core to the end of the available chain upon termination of system initialization. When all space within EXPOOL has been allocated, the Buffer Pool may be expanded by calling CRQED (Core Request for One Block EXEC D-Bank) to get a block of 2**9 words from System D-Bank. The borrowed core space will be released as soon as it is no longer needed in the Buffer Pool. When the total unused space is less than 4000 (octal) memory words, the Buffer Pool is set to a tight mode. In the tight mode, only critical requests, that is, those with the flag set, can be allocated space. All other requests are linked to the EXPOOL request chain and the requestor is deactivated by EXPOOL.

5.1.2. Request for a Buffer from EXPOOL. To request a buffer storage area from EXPOOL, the following calling sequence is used:

```
LXI,U        X11,P
LMJ          X11,EXPOOL
```

On exit from the request, the program leaves the external buffer address in the AO Register, the return address in the Index Register 11 (X11), and the address of the word that contain the user specified parameters, P, in the A1 Register. The information indicating the exact nature of the buffer request is made available to EXPOOL in the following format:

P:

SIZE	N---FC	ADDRESS

where:

SIZE = Number of words in the buffer desired.

N=0 : Needs a buffer when it becomes available.

N=1 : Must receive the buffer immediately to continue processing.

F=D : Add to the end of chain.

F=1 : Add to the front of chain.

C=0 : No chaining.

C=1 : Chain as specified in F.

ADDRESS = A pointer to the control word if C=1; or the address of the buffer to be assigned if C=0.

The Buffer Allocation Algorithm is as follows.

(1) Check if this is a legal buffer request size.

(2) Call request to request internal buffer.

  (A) If buffer of size 2**K is not available, then go to (C).

  (B) Remove the first buffer of size 2**K from available chain and go to (D).

  (C) K=K+1, if K>9, terminates unsuccessfully, otherwise, recursively call request.

  (D) If 2**K is the request size, then return to request, otherwise K=K-1, break buffer into two equal pieces.

  (E) Chain one piece into available chain and recursively return to request.

(3) Update size indicator, tight mode indicator.

(4) Save switch list ID or function ID.

(5) Chain to the control word if it is so requested.

(6) Return to the requestor.

5.1.3. <u>Release of a Buffer from EXPOOL</u>. To release a buffer storage area from EXPOOL, the routine EXREL (EXPOOL Buffer Release) is initiated by proving the following calling sequence:

```
LA          AO,P
LMJ         X11,EXREL
```

Where P has the following format:

```
P:          ..........................................
            :     SIZE     :-------:     ADDRESS     :
            ..........................................
```

The Buffer Release Algorithm is as follows.

(1) Check if this is a legal buffer release size.

(2) Update size indicator, tight mode indicator.

(3) Call release to release internal buffer.
   (A) If the buddy of this buffer is free, go to (C).
   (B) Chain the buffer to available chain and return to release.
   (C) Remove the buddy from available chain.
   (D) Combine with the buddy, set K=K+1, and recursively call release.
(4) Return to the requestor.

5.2. Preliminary Results of an Analysis of EXPOOL

The efficiency of a function or program depends both on the algorithm used, and the effectiveness of the code used to implement the algorithm. In evaluating EXPOOL, both the algorithm and the implementation have been analyzed. As described in Section 3 of [12], a simulation model of the buddy system storage allocation technique, as well as several other allocation schemes have been constructed and run on the Univac 1108.

Several core memory dumps of the EXPOOL buffer pool have been taken. The distribution of used buffer size was calculated according to the results obtained from the memory dumps, and has been used as the input source to ITFVA (Instruction Trace and Functional Value Analysis) described in Section 4.2. The time interval between a buffer being allocated and released is assumed to be an exponential distribution. Under ITFVA requests and releases are called. Figures 5-2 and 5-3 show the analysis results of the original EXPOOL program. We see 23.7 percent of the allocation time has been spent in looking through the table, TAB2, to convert the external request size into the internal buffer size index. It is interesting to note that within EXPOOL, the table, TAB2 is ordered randomly as shown in Figure 5-4. That is, there is no rationale for the sequence of entries in the table. It is of interest to calculate the average time required to search for an entry in the table. If we let E be the average search time to find a matching entry in TAB2, N(I) be the number of instructions needed to access the ith entry in the table, and P(I) be the probability that the ith entry in the table is requested. Then

$$E = N(1)*P(1) + N(2)*P(2) + - - - +N(12)*P(12).$$

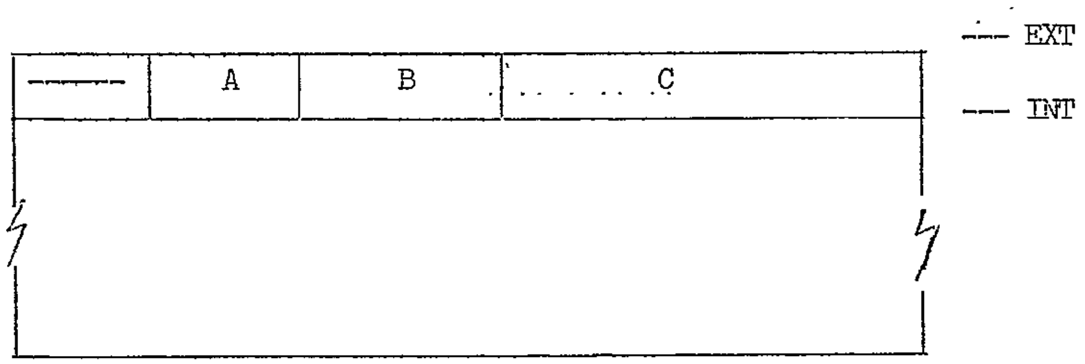
if N(I)=N*I, where N is a constant, the value of E is minimized if $P(I) \leq P(J)$ for all $J \geq I$. That is, a minimum search time can be obtained if the table entry is given indecreasing order according to its probability of occurrence. In Figures 5-5 and 5-6, the result of reordering the table, TAB2, according to the size usage distribution obtained from the memory dumps is shown. The percentage of CPU time spent in this table_lookup is still high, but, an average of 15.5 percent of allocation time has already been saved.

An additional saving in time may be obtained by recalling that the buddy system storage allocation technique is so defined because each buffer request made for a block of size N, where $2**K \leq N \leq 2**(K+1)$, is allocated a block of exactly 2**(K+1) words providing 2**(K+1) is less than or equal to the maximum block size permitted. In most allocation schemes, to convert an external request length to the internal size index, a table lookup is used. Actually, the feature of the buddy system provides a very easy way to handle the conversion. The simple formula is that the internal buffer size index K equals the number of bits in the machine word minus the number of bits with leading zeros. For this, a single shift and count instruction can get the index size immediately. Now the average search time E is decreased substantially. For, in this case, N(1) becomes a constant, C, the time to perform the shift count instruction. Hence, E = C. Figure 5-7 shows the result of the above change in the time required to access the appropriate word. An average of 29.1 percent saving for each request (or release) is gained over the code currently implemented in EXEC VIII.

In the 1108 Executive System, there will be essentially the same number of releases as requests for buffer storage after the system stabilizes, so that, in the following discussion, no attempt is made to distinguish the type of action requested in the allocation process. In the EXEC VIII version of the allocation routine, by using the TRACE routine it was found that the average number of instructions required for an allocation was 103. In the 1108, the average time per instruction is 1.12 μsec. Therefore, the time spent in one allocation process is 1.12 μsec times 103 instructions or .116 msec.

By reordering the table, TAB2, so that the order of the entries in TAB2 are given in decreasing order according to their probability of occurrence, the average number of instructions required for an allocation was found to be 87. The time spent in the allocation process is then .097 msec, a reduction of .019 msec per allocation. By introducing a shift and count instruction to replace the table lookup process, the average number of instructions was reduced to 74. The time spent in the allocation process is then .083 msec. This represents a reduction of .033 msec over the EXEC VIII version or a reduction of .014

over the version with a reordered TAB2.



EXT: External Buffer Address
INT: Internal Buffer Address

A = 0 If the buffer is used.
    B If the buffer is free.

B = The internal size index.

C = The link to the next buffer if the buffer is free.
    The function ID if the buffer is used by a function.
    The switch ID if the buffer is not used by a function.
    The return point if the buffer is used by the EXEC main interlock code.

Figure 5-1. Structure of a One Block Buffer of Size 2**B.

CODE EXECUTION FREQUENCY FOR EACH INTERVAL

LABEL	RELATIVE START	LOCATION END	TOTAL INST. EXECUTED	PERCENT OF RUN TIME
EXPOOL	0015	0030	3141	16.87
EXP2	0031	0037	500	2.69
EXPEXT	0040	0070	1300	6.98
INLK	0071	0142	1300	6.98
REQUES	0143	0154	976	5.24
NOMORE	0155	0171	1441	7.74
REQ23	0172	0204	630	3.38
MCORE	0205	0277	3	.02
EXREL	0300	0316	3341	17.94
ER22	0317	0330	800	4.30
EXREXT	0331	0343	500	2.69
ER23A	0344	0356	0	.00
RELEAS	0357	0363	1680	9.02
REL1.1	0364	0413	1378	7.40
REL1.2	0414	0434	811	4.36
REL2	0435	0442	900	4.83
REL3	0443	0446	0	.00
REL56	0447	0473	0	.00
OTHER	0000	0000	2	.01
1	2	3	4	5

TOTAL 18619 INSTRUCTION EXECUTED DURING THIS ANALYSIS.

Figure 5-2. Code Execution Frequency for each Labeled Block of the Accessing Routines (EXPOOL/EXREL) as Implemented in EXEC VIII.

1. The block symbolic name, that is the label.
2. The relative location of the label to the start of the routine.
3. The relative location of the instruction preceding the next label.
4. The total number of executed instructions within each labeled block of the routine.
5. The percentage of total run time spent in each labeled block of the routine.

THE MOXT FREQUENTLY EXECUTED INTERVALS

LABEL (EXREL) TOTAL 3341 INSTRUCTION EXECUTED.

MACRO INST. START	LOCATION END	MACRO INST. LENGTH	EXECUTION FREQUENCY	TOTAL INST. EXECUTED	PERCENT
0300	0311	9	100	900	26.94
0312	0313	2	100	200	5.99
0312	0315	3	747	2241	67.08

LABEL (EXPOOL) TOTAL 3141 INSTRUCTION EXECUTED.

MACRO INST. START	LOCATION END	MACRO INST. LENGTH	EXECUTION FREQUENCY	TOTAL INST. EXECUTED	PERCENT
0015	0023	7	100	700	22.29
0024	0025	2	100	200	6.37
0024	0027	3	747	2241	71.35

LABEL (RELEAS) TOTAL 1680 INSTRUCTION EXECUTED.

MACRO INST. START	LOCATION END	MACRO INST. LENGTH	EXECUTION FREQUENCY	TOTAL INST. EXECUTED	PERCENT
0357	0403	16	60	960	57.14
0357	0405	18	40	720	42.86
1	2	3	4	5	6

Figure 5-3. Analysis of Most Frequently Executed Labeled Blocks of the Accessing Routines (EXPOOL/EXREL) as Implemented in EXEC VIII.

1. The relative location of the first word of each macro-instruction to the start of the routine.
2. The relative location of the last word of each macro-instruction to the start of the routine.
3. The number of instructions in each macro-instruction.
4. The number of times the macro-instruction was executed.
5. Total instruction executed in each macro-instruction.
6. The percentage of labeled block execution time spent in the macro-instruction.

```
TAB2 AS IMPLEMENTED IN THE EXEC VII

 .  TABLE OF EXTERNAL AND INTERNAL BUFFER SIZES
 .     +   EXTERNAL SIZE, INTERNAL SIZE

 TAB:

                 +             3,2
                 +             6,3
                 +            28,5
                 +            56,6
                 +           224,8
                 +           127,7
                 +            15,4
                 +             7,3
                 +            31,5
                 +            63,6
                 +           255,8
                 +           511,9
```

```
TAB2 REORDERED TO OPTIMIZE TABLE LOOKUP PROCESS

    TABLE OF EXTERNAL AND INTERNAL BUFFER SIZES
      +   EXTERNAL SIZE, INTERNAL SIZE

TAB2.

                 +           511,9
                 +           127,7
                 +           224,8
                 +           255,8
                 +            56,6
                 +            63,6
                 +             6,3
                 +             7,3
                 +            15,4
                 +            28,5
                 +            31,5
                 +             3,2
```

Figure 5-4. Structure of TAB2 as used in EXEC VIII and Structure of Reordered TAB2.

CODE EXECUTION FREQUENCY FOR EACH INTERVAL

LABEL	RELATIVE START	LOCATION END	TOTAL INST. EXECUTED	PERCENT OF RUN TIME
EXPOOL	0015	0030	1485	9.70
EXP2	0031	0037	500	3.27
EXPEXT	0040	0070	1300	8.49
INLK	0071	0142	1300	8.49
REQUES	0143	0154	976	6.38
NOMORE	0155	0171	1441	9.41
REQ28	0172	0204	630	4.12
MCORE	0205	0277	3	.02
EXREL	0300	0316	1685	11.01
ER22	0317	0330	800	5.23
EXREXT	0331	0343	500	3.27
ER23A	0344	0356	0	.00
RELEAS	0357	0363	1680	10.98
REL1.1	0364	0413	1378	9.00
REL1.2	0414	0434	811	5.30
REL2	0435	0442	900	5.88
REL3	0443	0446	0	.00
REL56	0447	0473	0	.00
OTHER	0000	0000	4	.03
1	2	3	4	5

TOTAL 15307 INSTRUCTION EXECUTED DURING THIS ANALYSIS.

Figure 5-5. Code Execution Frequency for each Labeled Block of the Accessing Routines (EXPOOL/EXREL) as Implemented in EXEC VIII.

1. The block symbolic name, that is the label.
2. The relative location of the label to the start of the routine.
3. The relative location of the instruction preceding the next label.
4. The total number of executed instructions within each labeled block of the routine.
5. The percentage of total run time spent in each labeled block of the routine.

THE MOST FREQUENTLY EXECUTED INTERVALS

LABEL (EXREL) TOTAL 1685 INSTRUCTION EXECUTED.

MACRO INST. START	LOCATION END	MACRO INST. LENGTH	EXECUTION FREQUENCY	TOTAL INST. EXECUTED	PERCENT
0300	0311	9	100	900	53.41
0312	0313	2	100	200	11.87
0312	0315	3	195	585	34.72

LABEL (RELEAS) TOTAL 1680 INSTRUCTION EXECUTED.

MACRO INST. START	LOCATION END	MACRO INST. LENGTH	EXECUTION FREQUENCY	TOTAL INST. EXECUTED	PERCENT
0357	0403	16	60	960	57.14
0357	0405	18	40	720	42.86

LABEL (EXPOOL) TOTAL 1485 INSTRUCTION EXECUTED.

MACRO INST. START	LOCATION END	MACRO INST. LENGTH	EXECUTION FREQUENCY	TOTAL INST. EXECUTED	PERCENT
0015	0023	7	100	700	47.14
0024	0025	2	100	200	13.47
0024	0027	3	195	585	39.39
1	2	3	4	5	6

Figure 5-6. Analysis of Most Frequency Executed Labeled Blocks of the Accessing Routines (EXPOOL/EXREL) as Implemented in EXEC VIII.

The relative location of the first word of each macro-instruction to the start of the routine.

2. The relative location of the last word of each macro-instruction to the start of the routine.
3. The number of instructions in each macro-instruction.
4. The number of times the macro-instruction was executed.
5. Total instruction executed in each macro-instruction.
6. The percentage of labeled block execution time spent in the macro-instruction.

CODE EXECUTION FREQUENCY FOR EACH INTERVAL

LABEL	RELATIVE START	LOCATION END	TOTAL INST. EXECUTED	PERCENT OF RUN TIME
EXPOOL	0000	0004	900	7.12
EXP2	0005	0013	100	.79
EXPEX1	0014	0044	1300	10.29
INLK	0045	0112	900	7.12
REQUES	0113	0124	976	7.72
NOMORE	0125	0141	1441	11.40
REQ2B	0142	0154	630	4.99
MCORE	0155	0247	3	.02
EXREL	0250	0254	800	6.33
ER22	0255	0266	600	4.75
EXREXT	0267	0302	300	2.37
ER23A	0303	0311	0	.00
RELEAS	0312	0316	1680	13.29
REL1.1	0317	0347	1378	10.90
REL1.2	0350	0367	811	6.42
REL2	0370	0375	900	7.12
REL3	0376	0401	0	.00
REL56	0402	0426	0	.00
OTHER	0000	0000	6	.05
1	2	3	4	5

TOTAL 12637-INSTRUCTION EXECUTED DURING THIS ANALYSIS.

Figure 5-7. Code Execution Frequency for each Labelled Block of the Accessing Routines (EXPOOL-EXREL) as Implemented in EXEC VIII.

1. The block symbolic name, that is the label.
2. The relative location of the label to the start of the routine.
3. The relative location of the IN.
4. The total number of executed instructions within each labeled block of the routine.
5. The percentage of total run time spent in each labeled block of the routine.

6. SUMMARY

The measurement and evaluation of computer systems has finally been recognized as a significant field of endeavor for computer professionals. This recognition is evidenced by an increasing flow of literature. There is a dearth of available tools and techniques which are capable of measuring the large man-hardware-software complexes that are presently being developed. This report has attempted to describe various techniques for the measurement and analysis of system behavior, with emphasis being placed on the empirical performance analysis technique. The software monitoring of an existing system's executive system is a difficult and costly process. The tight design constraints imposed on an executive make it less amenable to inserting data recording devices than the typical user's program. The acquisition of performance data by well-designed benchmarks can provide useful measures of the system performance at a much lower developmental cost than a software recording capability. However, it is strongly recommended that a software recording utility be an early design requirement for any new system. Benchmarks cannot be substituted for comprehensive recording. There are three major concepts concerning system design constraints and requirements which affect a recording utility. These include interface and internal recordability and recording selectivity. Interface recordability consists of the ability to record the occurrence of any event that involves the interface between a user object program and the computer used. This concept forces the interface activity of a system to be clearly defined and standardized. Internal recordability is concerned solely with the internal behavior of the object program, that is, the ability to record program-generated data which is never transferred to another component of the user's computer. This concept requires that the program at any level should be able to initiate the operation of the recording function. Recording selectivity states that the user can specify any subset of the set of recordable data for actual recording. The ideal goal then is to design a language which will permit procedural-like statements which can be used to describe the recording operation. The language would permit logical and conditional as well as declarative statements. The logical and conditional statements would specify conditions under which recording is to take place, while the declarative statements would specify the data to be recorded.

7. BIBLIOGRAPHY

The following paper are grouped in accordance with the topics covered by this thesis; for a more complete bibliography, see [6]

General

1. Arbuckle, R. A. Computer Analysis and Throughput Evaluation. Comput.Autom., Vol. 15, No. 1 (January 1966) pp. 12-15, 19.

2. Arden, B. W., Boettner, D. Measurement and Performance of a Multi-Programming System. Proc. ACM 2nd SYMP on O/S Principles, October 1969, 130-146.

3. Bonner, A. J. Using System Monitor Output to Improve Performance. IBM System, J. 8, 4 (1969) 290-297.

4. Buchholz, W. A Selected Bibliography on Computer System Performance Evaluation. Computer Group News (New York) 2, 8 (Dec. 1968), 21-22.

5. Calingaert, P. System Performance Evaluation: Survey and Appraisal. (CR 11661) CACM 10, 1 (January 1967) pp. 12-18.

6. Crooke, S., Minker, J. Key Word in Context Index and Bibliography on Computer System Evaluation. TR-69-100, University of Maryland. December 1969.

7. Drummond, M. E., Jr. A Perspective on System Performance Evaluation. IBM System, J. 8, 4 (1969) 252-263.

8. Fife, D. W. Alternatives in Evaluation of Computer Systems. (AD 683693) Mitre Corporation, Bedford, Mass. Report MTR-413, December 1968.

9. Gosden, J. A., Sisson, R. L. Standardized Comparisons of Computer Performance. Information Processing 1962 (PROC. IFIP Congress 1962), pp. 57-61.

10. Joslin, E. O. Cost-Value Technique for Evaluation of Computer System Proposals. (CR 6166) PROC. AFIPS SJCC 1964, Volume 25, pp. 367-381.

11. Knight, K. E. Changes in Computer Performance. Datamation, Vol. 12, No. 9 (September 1966), pp. 40-54.

12. Minker, J., Crooke, S., Yeh, J. Analysis of Data Processing Systems. TR-69-99, University of Maryland, December 1969.

13. Opler, A. Measurement of Software Characteristics. Datamation, Vol. 10, No. 7 (July 1964), pp. 27-30.

14. Patrick, R. L. Measuring Performance. Datamation, Vol. 10, No. 7 (July 1964), pp. 24-27.

15. Smith, J. M. A Review and Comparison of Certain Methods of Computer Performance Evaluation. (CR 15293) Computer Bull 12, 1 (May 1968), pp. 13-18.

16. Statland, N. Methods of Evaluating Computer Systems Performance. Comput. Autom., Vol. 13, No. 2 (February 1964), pp. 18-23.

ANALYTIC MODEL ANALYSIS TECHNIQUE

17. Bryan, G. E., Shemer, J. E. The UTS Time-Sharing System - Performance Analysis and Instrumentation. Proc. ACM 2nd Symp on O/S Principles, October 1969, 147-158.

18. Coffman, E. G., Jr. Stochastic Models of Multiple and Time-Shared Computer Operations. Ph.D. Dissertation. Report No. 66-38, Department of English, U.C.L.A. 1966.

19. Coffman, E. G., Jr. Markov Chain Analysis of Multiprogrammed Computer Systems. (AD 692004), Naval Res. Log. Qtrly., 16, 2 (June 1969), 175-197.

20. Coffman, E. G., Jr., Kleinrock, L. Feedback Queueing Models for Time-Shared Systems. (CR 16432) JACM 15, 4 (October 1968), pp. 549-576.

21. Demeis, W. M., Weizer, N. Measurement and Analysis of a Demand Paging Time-Sharing System. Proc. ACM 24th National Conf., August 1969, 201-216.

22. Gaver, D. P., Jr. Probability Models for Multiprogramming Computer Systems. (CR 13459) J. ACM 14 (July 1967), pp. 423-438.

23. Gurk, H. M., Minker, J. Storage Requirements for Informance Handling Centers. JACM (January 1970).

24. McKinney, J. M. A Survey of Analytical Time-Sharing Models. Computing Surveys 1, 2 (June 1969), 105-116.

25. Minker, J. A Stochastic Model of an Information Center. TR-69-90, University of Maryland, 1969.

26. Patel, N. R. Mathematical Analysis of Computer Time-Sharing Systems. MS Thesis - E. E. Department (AD 605 825) M.I.T. Technical Report 20, MIT. Cambridge, Mass. July 1964.

27. Pinkerton, T. B. Program Behavior and Control in Virtual Storage Computer Systems. (Ph.D. Dissertation) Tech. Rpt. 4,

University of Michigan, Ann Arbor, Mich., 1968.

28. Rasch, P. J. A Queueing Theory Study of Time-Shared Computer Systems. N6920120. Ph.D. Thesis, Southern Methodist University, Dallas, Texas. 1967, 95P.

29. Smith, J. L. An Analysis of Time-Sharing Computer Systems Using Markov Models. (CR 10835) Proc. AFIPS SJCC 1966, pp. 87-95.

30. Wulf, W. A. Performance Monitors for Multiprogramming Systems. Proc. ACM 2nd Symp on O/S Principles, October 1969, 175-181.

SIMULATION MODEL ANALYSIS TECHNIQUE

31. Braddock, D. M., Dowling, C. B. Simulation, Evaluation, and Analysis Language: Seal. IBM Program Lib., IBM, Hawthorne, New York.

32. Canning, R. G., Edit. Data Processing Planning via Simulation. EDP Analyzer, Vol. 6, No. 4 (April 1968).

33. Chu, Y. An Algol Like Computer Design Language. CACM 8, 10, (October 1965), pp. 607-615.

34. Cohen, L. J. Associates. System and Software Simulator: S3, Technical Manual. AD679-269 - AD679-272.

35. Compress. A Technical Description of SCERT. Compress, Rockville, Maryland.

36. Estrin, G., Kleinrock, L. Measures, Models and Measurements for Time-Shared Computer Utilities. (CR 13642) Proc. ACM 22nd National Conf., (1967) pp. 85-96.

37. Fine, G. H., McIssac, P. V. Simulation of a Time-Sharing System. (CR 11118) Mgt. Science 12, 6 (February 1966), pp. B180-194.

38. Goodman, R. M., Pivonka, L. M. A Simulation Study of the Time-Sharing Computer System at the Naval Postgraduate School. (AD 692447) Masters Thesis, Nav. Postgrad. School, Monterrey, California, June 1969, 150P.

39. Grice, A. HARGOL - A Hardware Oriented Algol Language. Internal Report No. VA5, August 1966, A/S Regnecentralen, Copenhagen, Denmark.

40. Herman, D. J., Ihrer, F. C. The Use of a Computer to Evaluate Computers. (CR6167) Proc. AFIPS SJCC 1964, Vol. 25, pp.383-395.

41. Herman, D. J. SCERT: A Computer Evaluation Tool. Datamation, Vol. 13, No. 2 (February 1967).

42. Holland, F. C., Merikallio, R. A. Simulation of a Multiprocessing System Using GPSS. IEEE Trans. Syst. Sci. Cyb. Vol. SSC-4 (November 1968), pp. 395-400.

43. Huesmann, L. R., Goldberg, R. P. Evaluating Computer Systems Through Simulation. (CR 13526) Computer J. 10, 2 (Aug 67) pp. 150-156.

44. Hutchinson, G. K., Maguire, J. N. Computer Systems Design and Analysis Through Simulation. (CR 9939) Simulation Univac 1107. Proc. AFIPS FJCC 1965, pp.161-167.

45. Hutchinson, G. K. Some Problems in the Simulation of Multiprocessor Computer Systems. (CR 15578) Proc. IFIP Working Conf, Oslo, 1967.

46. IBM Computer System Simulator/360 Program Description and Operations Manual. IBM Form No. Y20-0130.

47. Ihrer, F. C. Computer Performance Projected Through Simulation. Comput. Autom., Vol. 17, No. 4 (April 1967), pp. 22-27.

48. Jacobson, R. V. Digital Simulation of Large-Scale Systems. (CR 10843) Proc. AFIPS 1966 SJCC 159-164.

49. Lehman, M. M., Rosenfeld, J. L. Performance of a Simulated Multiprogramming System. Proc. AFIPS FJCC 1968, Vol. 33, PT2, pp. 1431-1442.

50. Markowitz, H. M., Hausner, B., Karr, H. W. SIMSCRIPT: A Simulation Programming Language. Prentice Hall, Inc. Englewood Cliffs, N. J. 63.

51. Nielsen, N. R. An Approach to the Simulation of a Time-Sharing System. (CR 14066) Proc. AFIPS FJCC 1967, pp. 419-428.

52. Nielsen, N. R. Simulation of Time-Sharing Systems. (CR 12769) CACM 10 (July 1967) pp. 397-412.

53. Nielsen, N. R. Computer Simulation of Computer System Performance. (CR 13525) Proc. ACM 22nd National Conf. (1967) pp. 581-590.

54. Rehmann, S. L., Gangwere, S. G., Jr. A Simulation Study of Resource Management in a Time-Sharing Environment. Proc. AFIPS FJCC 1968, Vol. 33, PT2, pp. 1411-1430.

55. Scherr, A. L. An Analysis of Time-Shared Computer Systems. (CR 14068) Ph.D. Dissertation (AD 470 715) M.I.T. Cambridge, Mass., June 1965.

56. Scherr, A. L. Analysis of Time-Shared Computer Systems - Simulation of CTSS. (CR 12369) M.I.T. Res. Monograph No. 36, MIT

Press, Cambridge, Mass. 1967.

57. Seaman, P. H., Soucy, R. C. Simulating Operating Systems. IBM Sys. J. 8, 4 (1969) 264-279.

58. UNIVAC. General Purpose System Simulator II (GPSS II) 'Reference Manual'. UNIVAC Manual UP-4129.

HARDWARE MONITOR ANALYSIS TECHNIQUE

59. Allied Computer Technology, Inc. Computer Performance Monitor System Summary Manual.

60. Apple, C. T. The Program Monitor - A Device For Program Performance Measurement. Proc. ACM 20th National Conf. (August 1965), pp. 66-75.

61. Computer and Programming Analysis, Inc. Computer Performance Analyzer Series 7700 Description.

62. Computer Synectics, Inc. System Utilization Monitor Model SM-416 Specification. Form No. A-416-4, CSI, Santa Clara, California.

63. Computer Synectics, Inc. System Utilization Monitor Evaluation Technique. Form No. 4-416-1, CSI, Santa Clara, California.

64. Estrin, G., Hopkins, D., Crocker, S. D. SNUPER Computer - A Computer In Instrumentation Automaton. (CR 13296) Proc. AFIPS SJCC 1967, pp. 645-656.

65. Roek, D. J., Emerson, W. C. A Hardware Instrumentation Approach to Evaluation of Large Scale System. Proc. ACM 24th National Conf., August 1969, 351-367.

66. Schulman, F. D. Hardware Measurement Device for IBM System/360 Time-Sharing Evaluation. (CR13298) Proc. ACM 22nd National Conf., (1967), pp. 103-109.

67. Stevens, D. F. System Evaluation on the Control Data 6600. Proc. IFIP Congress 68 (August 1968), P.C34-38.

SOFTWARE MONITOR ANALYSIS TECHNIQUE

68. Bemer, R. W., Ellison, A. L. Software Instrumentation Systems for Optimum Performance. Proc. IFIP Congress 68 (August 1968), pp. 39-42.

69. Boole and Babbage, Assoc. Systems Measurement Software (SMS/360) Users Guide For CUE-1. Boole and Babbage Report 135, February 1969.

70. Boole and Babbage, Assoc. Systems Measurement Software (SMS/360) Users Guide For PPE. Boole and Babbage Report 41, May 1969.

71. Campbell, D. J., Heffner, W. J. Measurement and Analysis of Large Operating Systems During System Development. (CR 16874) Proc. AFIPS FJCC 1968, Vol. 33, pp. 903-914.

72. Cantrell, H. N., Ellison, A. L. Multiprogramming System Performance Measurement and Analysis. Proc. AFIPS SJCC 1968, Vol. 30, pp. 213-221.

73. Deniston, W. R. SIPE - A TSS/360 Software Measurement Technique. Proc. ACM 24th National Conf., August 1969, pp. 229-245.

74. Hornbuckle, G. D. A Multiprogramming Monitor for Small Machines. (CR 16431) Comm. ACM 10,5 (May 1967), pp. 273-278.

75. Karush, A. D. Two Approaches for Measuring the Performance of Time-Sharing Systems. Proc. ACM 2nd Symp. on O/S Principles, October 1968, pp. 159-166.

76. Karush, A. D. The Computer System Recording Utility: Application and Theory. SP-3033, SDC, Santa Monica, California, February 1969.

77. Pinkerton, T. B. Performance Monitoring in a Time-Sharing System. CACM 12, 11 (November 1969) 608-610.

78. Sackman, H. Computers, System Science, and Evolving Society. John Wiley and Sons, Inc., New York, 1967.

79. Saltzer, J. H., Gintell, J. W. The Instrumentation of MULTICS. Proc. ACM 2nd Symp. on O/S Principles, October 1969, pp. 167-174.

80. Stanley, W. I., Hertel, H. F. Statistics Gaithering and Simulation for the APOLLO Real-Time Operating System. IBM System J. No. 2 68, pp. 85-102.

81. Van Horn, E. C. Three Criteria for Designing Computer Systems to Facilitate Debugging. CACM, Vol. 11, No. 5, (May 1968) pp. 360-365.

BENCHMARK ANALYSIS TECHNIQUE

82. Arbuckle, R. A. Computer Analysis and Throughput Evaluation. Comput Autom., (January 1966), pp. 12-15.

83. Budd, A. E. A Method for the Evaluation of Software: Executive Operating or Monitor Systems. Mitre Corporation, Bedford, Mass. MTR-197.

84. Joslin, E. O. Application Benchmark: The Key to Meaningful Computer Evaluations. Proc. ACM 20th National Conf. (1965), pp. 27-37.

85. Joslin, E. O., Aiken, J. J. The Validity of Basing Computer Selections on Benchmark Results. Comput Autom., (January 1966), pp. 22-23.

86. Karush, A. D. Benchmark Analysis of Time-Sharing Systems. (AD 689781) Report SP-3347, SDC, Santa Monica, California June 1969, 40P.

87. Karush, A. D. Benchmark Measurement of the Adept-50 Time-Sharing System. Report TM-4324, SDC, Santa Monica, California, (June 1969), 44P.

88. Hillegass, J. R. Standardized Benchmark Problems Measure Computer Performance. Comput Autom., (January 1966), pp. 16-19.

89. Oppenheimer, G., Weizer, N. Resource Management for Medium Scale Time-Sharing Operating Systems. CACM 11,5 (May 1968), pp. 313-322.

90. Rubey, R. J. A Comparative Evaluation of PL/1. Datamation, Vol. 14, No. 12 (December 1968), pp. 22-25.

91. Totar, J. B. Real-Time Processing Power: A Standardized Evaluation. Comput Autom. (Apr. 67), pp. 16-19.

UNIVERSITY OF MARYLAND
THE GRADUATE SCHOOL
COLLEGE PARK, MARYLAND 20742

TO: The Graduate School

FROM: Jack Minker
Advisor

SUBJECT: Certification of Completion of Master's Degree Without Thesis.

Please check appropriate Master's degree.

(✓) Master of Arts (Without Thesis)
( ) Master of Business Administration
( ) Master of Education
( ) Master of Library Science
( ) Master of Music
( ) Master of Science (Without Thesis)
( ) Master of Social Work

I certify that Mr. James Yeh candidate for a Master
Name of Candidate
of Science degree, who seeks the degree at the commencement
Degree
of June, 1970 has met all the requirements of the department or program for the degree including:

✓ Seminar or Research papers

✓ Comprehensive Examinations (written or oral)

Signed,

Jack Minker
Name of Professor (Advisor)

Associate Professor
Title

May 19, 1970
Date

(Form G.S. 1-Jan. 1969)