DEPT. OF COMPUTER SCIENCE,
UNIVERSITY OF WATERLOO,
WATERLOO, ONTARIO N2L 3G1

THE MONITORING OF COMPUTER SYSTEMS & NETWORKS
A SUMMARY & PROPOSAL

by

D.A. Sutton & D.E. Morgan

Research Report CS-74-06

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

May 1974

THE MONITORING OF COMPUTER SYSTEMS & NETWORKS
A SUMMARY & PROPOSAL

by

D.A. Sutton & D.E. Morgan

Research Report CS-74-06

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

May 1974

THE MONITORING OF COMPUTER SYSTEMS AND NETWORKS: A SUMMARY AND A PROPOSAL

by

D.A. SUTTON

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS OF THE

DEGREE OF

MASTER OF MATHEMATICS

AT THE

UNIVERSITY OF WATERLOO

WATERLOO, ONTARIO

DEPARTMENT OF APPLIED ANALYSIS

AND COMPUTER SCIENCE

APRIL 1974

ABSTRACT

The monitoring of a computer system involves the detection of electrical signals within a computer and the use of these signals for making performance measurements or for detecting specific events. Monitoring must now be applied to computer networks, as well as to single computer systems. The purpose of this thesis is to discuss the monitoring of computer systems and networks, and then to present a computer network monitor.

The discussion first considers those people and organizations who would be interested in monitoring a computer and then considers why they would want to monitor.

The many computer events which can be monitored are presented in several formats, but the main presentation utilizes a subclassification of the parameters into two parts--those for the computer and those for the workload. Once these parameters are listed, a formal definition of a monitor event is given.

The means of accomplishing monitoring at present consists of a discussion of three types of monitors, a description of some of the monitors which already exist and a detailed description of the limits and characteristics of each type of monitor.

Chapters II, III, V, and VI are based primarily on documentation of and experience with, the monitoring of computer systems. Even though the monitoring of computer networks is considered throughout, it is still necessary to consider those monitor problems which are specific to computer networks. Once this is summarized, a unique monitor is described and a technique for the monitor's evaluation is presented.

ACKNOWLEDGEMENT

The author would like to sincerely thank his friend and advisor Dr. D. E. Morgan for his stimulating advice, his suggestions and his criticisms. His cooperation was very beneficial.

The author is grateful to his sister, Donna, for her aid in the preparation of the manuscript.

The work described here was performed at the Department of Applied Analysis and Computer Science, at the University of Waterloo.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1.1 PURPOSE

Computer development has progressed to a stage whereby the operation of a total computer system is extremely difficult to understand. Besides single computers with their peripherals and operating systems, there now exist networks of computers (at least two computers connected together) and computer networks (a network of computers or several terminals connected to at least one computer).

The monitoring of a computer system involves the detection of electrical signals within a computer and the use of these signals for making performance measurements or for detecting specific events. Monitoring must now be applied to computer networks, as well as to single computer systems. Thus the purpose of this thesis is to discuss the problems of monitoring computer systems and networks, and then to present a computer network monitor.

This discussion first considers those people and organizations who would be interested in monitoring a computer and then considers why they would want to monitor.

The many computer events for monitoring can be presented in several formats, but our main presentation utilizes a subclassification of the parameters into two parts--those for the computer and those for the workload. Once these parameters are listed, a formal definition of a monitor event is given.

In order to understand how the above monitoring is being accomplished, we present a discussion on the three types of monitors, a description of some of the monitors which already exist and a detailed description of the limits and characteristics of each of these types of monitors.

This discussion is based primarily on documentation of, and experience with, the monitoring of computer systems. Even though the monitoring of computer networks is considered throughout, it is still necessary to consider those monitor problems which are specific to computer networks. Once this is summarized, a new monitor is described and a technique for the monitor's evaluation is presented.

1.2 ORGANIZATION

The following chapters are organized to study and comment on the present monitor tools and techniques as applied to both computer systems and networks, and then to describe a proposed monitor system capable of monitoring computer networks. Thus the chapters and their content are:

A. Introduction;

B. Who monitors and why do they monitor?

C. What would we like to monitor?

D. How can we formally represent that which we wish to monitor?

E. What tools and techniques have been available for monitoring?

F. What are the characteristics and limitations of these tools?

G. A summary of the monitor concepts which apply to the monitoring of computer networks, a description of a new network monitor, and a technique for performing its preliminary evaluation.

MONITORING AND MANAGEMENT

2.1 INTRODUCTION

Systems must be managed. There are three basic characteristics of any system:

A. Each system is part of a still larger system;

B . Each system has a specific purpose to which all its parts are designed to contribute, and

C. Each system is complex in the sense that a change in one variable will cause changes in others (KOON680).

The complexity of these three characteristics implies that every system must be controlled and its processes coordinated. Thus, the control and coordination of systems are the two basic tasks of management.

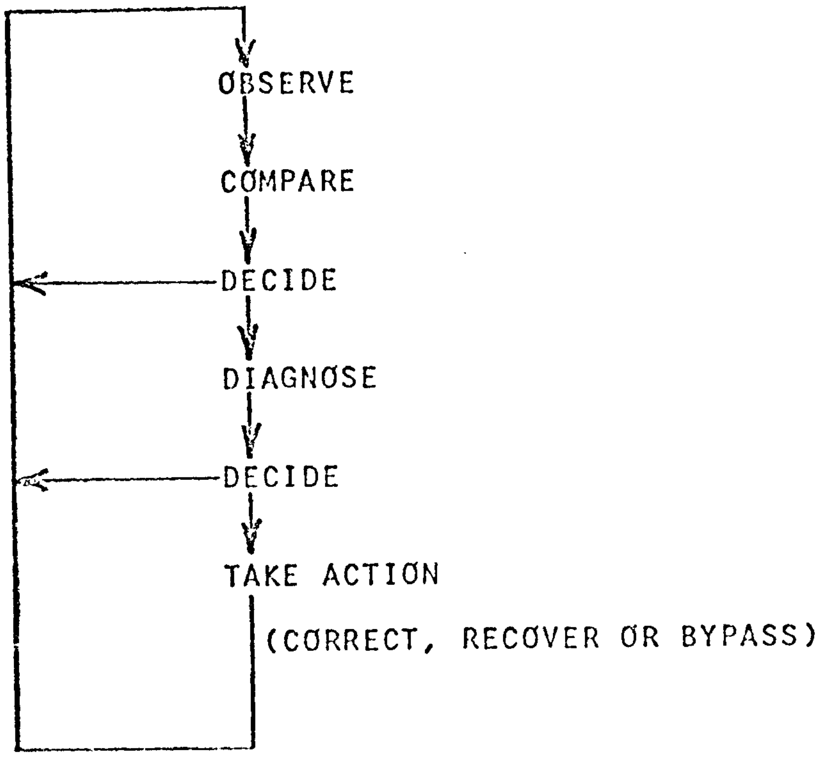
By the above characteristics, management itself is a system or a systematic process, applied by both humans or computers. Management is "the effective coordination and utilization of available human and non-human resources to achieve the objectives of the organization" (PIGO690). The objectives may be to maximize profit, to improve efficiency, to expand the scope of services, to do research, to improve the quality of a product or service, or to attract more consumers. Generally the objective for any managerial system can be stated as the desire to perform two functions:

A. To provide a product or service cost-effectively, and

B. To charge for the consumer's use of the product or service.

Thus management involves the coordination of planning objectives, policies, programs, and procedures; of organizing a structure of roles with intent to achieve the system's goals; of supplying resources; of directing the subsystems; and of controlling the system process through the measurement of performance, the correction of negative deviations and the assurance of the accomplishment of goals.

Management (whether human or computer) involves the execution of a simple cycle:



This cycle is a basic thought process very common to any system evaluation, whereby a continual observation of the system behaviour is performed. Managers use the observed performance and malfunction indicators to help them understand the system better and to detect any trouble.

Managers compare the observed behaviour with an expected behaviour pattern or system standard. Both the expected behaviour pattern and the system standard are based on a model of the system or on redundant information which is contained either within the system or within the system's environment.

Next managers must decide if a discrepancy exists between the real and the expected behaviour. If no discrepancy occurs, the manager can continue his observation of the system on a gross level; otherwise, a diagnosis of the cause must be performed on areas relevant to the problem and at a level detailed enough to locate the fault without incurring undue cost.

The next decision is to determine what action is to be taken. When the cause of the problem has been diagnosed, management can either correct or bypass the cause of the problem, recover the system, or modify the system to improve the performance. However, the expected behaviour may also be incorrect or incomplete, such that the model, from which the expected behaviour is taken, may have to be modified.

In most cases we assume that the model is correct in order to prevent the system (The system in this case, consists of both the object (*) and the monitor systems.) from being unstable.

The manager's last function (consumer charging) involves only three steps: the observation and the recording of the resources consumed and the computation of a charge as a function of these quantities.

All configurations of computing hardware or software are systems and thus must be managed in a manner similar to the general systems discussed above. Therefore, since the computer is a system requiring manangement, and management is a systematic process which can be described as the occurrence of the two basic functions discussed above, the nature of these two functions indicates that management has four reasons for monitoring the performance of a computer system or network:

A. To observe performance;

B. To detect malfunctions;

C. To diagnose specific problems, and

D. To provide accounting.

(*) The object system is the system which is being monitored.

Monitoring is performed in many ways, depending on the complexity of the system of interest. For a simple minicomputer, which the user controls, an experienced user can simply watch the indicator lights for unusual behaviour (eg. the tell-tale flicker of an infinite loop), then halt the process and step through instructions one at a time to locate where the program is executing and what the cause of the loop may be. Special tools and techniques must be provided to aid the human being in monitoring more complex systems, such as multiprogrammed computer systems and computer networks. Monitoring of network activity can frequently by very difficult. Some of the characteristics which cause problems are the number of subsystems of the network, the topology of the network, and the network message control. One of the tasks which is made difficult is time synchronization between monitor events. These problems will be discussed in Section 7.1.

Thus, when considering the complexity of computer systems and networks, special tools and techniques are necessary to aid managing and understanding systems.

2.2 ORGANIZATIONS AND MONITORING

The first function of management (providing a product or service) implies that both the product and the means of production must be dependable. The requirement which any organization has for a dependable computer system or computer network is determined by the degree to which the quality and availability of their product or service is sensitive to the dependability and performance of their computer system or network. In short, what is the cost of an inefficient or unreliable computer? Is it enough to justify the cost of an automated monizoring system that helps management and systems experts find inefficiencies and detect malfunctions?

Some organizations, such as military or communications organizations, find that dependability is essential in order to supply their service, so they must afford such automated monitoring systems. The work of other organizations is so independent of their computers that they can afford to wait for computer services which are slow because of inefficiencies or malfunctions. The latter organizations do not need an elaborate monitoring system. Organizations between these two extremes will use such systems only when they become cost effective as compared with the cost of manual detection and recovery from failure, or of degraded performance.

But many organizations would prefer to have a monitoring system which was sufficiently flexible to handle requirements ranging from routine observation of performance to sophisticated on-line diagnosis. A monitoring system must also be dependable so that its users can have confidence in its results.

2.3 PEOPLE AND MONITORING

In general, everyone uses monitors of varying sophistication. Some people require only their five senses for observing, while others supplement their natural sensors (eg. eyes) with special tools, such as microscopes or binoculars. Let us give some thought to who will be interested in monitoring computers:

A. Managers who have computer responsibilities will want a monitor system that is easy to use and does not require that they possess detailed knowledge of hardware, software and statistics, in order to learn if their system is behaving well or not.

B. Systems maintenance people (both engineers and programmers) want a monitor to give them access to detailed aspects of the object system. They also want a tool which is easy to use, so that they are not constantly fighting the tool to get results. Moreover, they need a tool by which they can gain an understanding of the nature of the parameters that characterize abnormal and normal behaviour of the system.

C. Designers are obvious users of monitors for computers. As the new system or network progresses and is in its trial runs, a flexible, multi-purpose monitor would be extremely useful to help debug both hardware and software modules.

D. Researchers want a monitor system to be flexible enough to permit measurement of the simplest event or a complex logical or sequencial combination of events which they deem to be of interest. These results will be used in projects like model validation.

2.4 CONCLUSIONS

The conclusions reached during the above discussion are:

A. Special tools and techniques are needed to aid in managing and understanding systems;

B. Organizations would prefer that one system could accomplish all the goals, rather than to require a wide range of specialized tools. Thus, there is a need for a flexible system to handle a range of experiments from routine observation of performance for managers to a sophisticated diagnostic tool for systems experts.

C. The monitor should be cost-effective when compared with the cost of not having it and of failing to provide the service or product within a reasonable time and at a reasonable level of quality.

D. The monitor system should be easy to learn to use as well as easy to use.

WHAT SHOULD A MONITOR DO?

3.1 INTRODUCTION

During the design of a computer monitor, it is important to have a set of specific, well defined aims. At this point, we wish to discuss what the users of monitoring systems may have for their aims. There are three levels which this discussion must occupy. The questions of concern to us are:

A. What are the overall goals of a monitor's users? For instance, do they wish to improve performance?

B. What are the objectives as they try to achieve their goal? For instance, in order to improve performance, are they seeking to discover what and where system bottlenecks are?

C. What are the events to be monitored, so that each objective is achieved? For instance, to determine whether the disk is a bottleneck, are they seeking to measure disk and seek time?

By asking and answering these questions, we will be able to obtain a comprehensive list of the events which should be detectable by a monitor.

3.2 WHAT GENERAL GOALS DO USERS HAVE?

The managers, system personnel and researchers mentioned in Chapter II have general goals in mind such as increasing throughput, reducing response time, or hypothesis testing, as they observe a computer system. To achieve these goals, the user should be able:

A. To obtain enough information about the system and its parts:

1. To permit an understanding of its performance;
2. To provide data for simulation and modelling;
3. To determine if tuning of the hardware or software is needed, and
4. To determine what the characteristic parameters are.

B. To measure the characteristic parameters of a system:

1. To determine if a dependency exists between these parameters, and
2. To provide data for use in a comparison of systems, and of a real system to an existing standard or model.

C. To obtain enough information about the workload:

1. To provide data for modelling, simulation, benchmarking, and other system analysis programs;

2. To determine if the users are making efficient use of their resources or if they need consultation, and

3. To determine the parameters of the workload.

D. To measure the characteristic parameters of a workload:

1. To determine if a dependency exists between these parameters, and

2. To provide data for the accounting of consumed resources per user.

E. To measure the attributes of special events(An attribute of an event is defined as a quality of the event which can be detected and used for measurement);

F. To detect specific events such as malfunctions, rare occurrences, and overload conditions, and

G. To obtain enough information about the system to do adequate diagnosis of a malfunction.

3.3 WHAT OBJECTIVES ARE ATTACKED?

As we pursue knowledge of a computer system, we must of course analyse the system with respect to all factors. An appropriate way to analyse a system is to discuss it in sections. For example, an industrial system consists of two logically separate sections. The industry itself is the section which utilizes the resources to produce the final product. The second is the consumer's demands on industry--What do the people want from the industry? In our case, the industry corresponds with the computer system and the consumer's demands with the workload (reference FERR727). The following discussion is organized according to this division; the computer system, then the workload.

3.3.1 COMPUTER SYSTEM'S PARAMETERS

The study of operational computer systems can be discussed in three parts. First to be considered is the utilization or activity of all the resources. The second is throughput, the measure of power of the system and its resources--how much work does it process per unit of time? The last concerns how well the user (or a task) is serviced when any request for action is presented to the system, for which the performance standard is response time. These terms are discussed in the following sections.

3.3.1.1 RESOURCE UTILITY (*)

We are concerned only with the broad definition of a resource as .being anything which a process can request or await, be it a physical device, a software program or a signal. This general definition is used in SHAW72C, but for a complete definition of resource types, see HØLT711 or HØLT726. This broad terminology is appropriate here since we wish to discuss the activity of any resource and the amount of management required for each.

The knowledge of the distribution function of the utilization of a resource can help managers coordinate their system in order to increase concurrency. That is, to accomplish concurrency, a manager must keep as many resources as possible busy doing useful work. This can be accomplished by:

A. Balancing the number of active compute bound and I/Ø bound tasks, and

---

(*) Resource utility is defined as the frequency of access and the quantity of usage for each resource.

B. Balancing the requests for a resource by:

1. Distributing the over-active datasets across the vast majority of the disks or drums and not overloading one or two storage units with all the active files, and

2. Distributing the devices on the controllers such that one controller is not a bottleneck. (Similarly for controllers on channels.)

The observation of resource utility also helps to determine what "size" (including speed, word size, amount of storage, etc.) of processors or storage devices would be needed by an installation. It can also help the system's operators to decrease the access time of datasets (reference DYER72B, DYER734) by storing the active files of the random access devices close together to decrease the seek time, and by using the access frequency of a dataset to determine what type of storage (classified by speed) would be appropriate. Besides using the monitoring of resource utility for observation of performance, it is used by the accounting routines for charging the users.

3.3.1.2 THROUGHPUT

Throughput is the prevalent indicator of computer system and network efficiency, and is the comparison standard for one computer to another, or to a previously defined standard or model. Throughput is broadly defined as a measure of the amount of work performed per unit time. It can be approximated by the number of processes reaching completion or leaving the system per unit of time. For a detailed definition see reference STIM690.

Throughput appears to be an appropriate way of measuring computing efficiency, but the broad definition given above and even the definitions by Stimler (in reference STIM690) have permitted its use in too many unrelated areas--jobs per hour or day, messages per minute, service calls per second, bits per second, etc. Thus it is a general term to which several particular definitions can be applied for a specific purpose, but the standard for efficiency should be a parameter which characterizes the system.

Dr.K.Kolence in his latest plea (reference KØLE732) for structure within and communications between those groups of people who are concerned with computer system measurement, discussed the need for a basic measurement unit for software units (*). He stated that "In our field we must admit that, except for intuitive feeling, we have no reason to believe any of our measures are deeply characteristic of the properties of software units. It is this unfortunate state of affairs which must be rectified by the development of valid theory, as it already has in the world of physical reality." (Page 71 of reference KØLE732)

Naturally we concur with him, but we are concerned with more: the development of a measurement system which includes subsystems of both hardware and software--the whole computer system, not just a theory for software. Consider the mechanical unit of power--ergs per second and foot-pounds per minute--or the electrical unit of power--joules per second. These are units of power expressed as units of mechancial or electrical energy used per unit of time. A similar basic standard is required for computer energy used per unit of time. What is the unit of computer energy? Is it bits processed per second or perhaps Binary Units of Work (reference RØZW732)?

(*) Dr. K.Kolence defines a 'software unit' to be an arbitrarily large or small grouping of code.

Since computers began, there have been several attempts at defining gross computer measurement standards. Computers were compared first by raw speed--memory, clock, arithmetic and I/O--then by raw speed and word size; followed by listings of instructions sets, then the Adam's chart, which contained all the characteristics of the computers marketed or announced at the time of publication. The next attempt involved numerical standards--the Gibson Mix (JOHN70C, ARBU661), and the Knight Figure of Merit (SMIT685). The main failings of these measurement standards were that the performance could not be predicted from the presented data, that no single system could be fully described and that they did not reflect the effects of multi-programming or multi-processing.

As computer systems progressed and were no longer single user systems, a need arose for a software manager--the operating system--and a need developed to measure the effects of the operating system on computer performance. Special programs (reference CAMP732) were developed to accomplish this. The three main classes of these special programs were the Nucleus or Kernel programs(JOHN70C, LUCA719), the Synthetic programs (LUCA719) and Benchmarks (JOHN70C).

Benchmarking is the only one of the above standards still widely applied. This entails a selection of programs from a computer system's workload to provide a standard workload. Each job is run on a stand alone basis and then in the multi-programming or multi-processing environment. The throughput and turnaround (*) are measured as the benchmarks are run and the results for each separate test can be used to discuss multi-programming or multi-processing as compared to a single user system. This is the best

existing technique for a comparison of systems, but we must accept that the chosen programs are representative of the workload and that turnaround and throughput are adequate indicators of efficiency. Thus Benchmarks and the present definition of throughput provide a highly controlled test and are sufficient standards for some managers, but not enough detail is provided for designers and researchers.

3.3.1.3 RESPONSE

Response is a measure of how fast a request for service is performed. L.Kleinrock (reference KLEI001) uses the general definition of response or turnaround to be the total time a customer spends in the system. The start time and stop time of the period are flexible and depend on what information the experimenter wishes to include in his measurement. The most commonly used period (used in

---

(*) Turnaround is the time that a job or message is in the system. It is used as an indicator of the system response.

reference STIM690 and several others) is the time between entering of the last character of a request for service and receiving of the first character of the reply. However, the basic parts of the response period are entering, transmitting and processing the request, then transmitting and receiving the response.

Kleinrock thinks that a measurement of the response time should provide both the first and second moments of the distribution, but he acknowledges that the average may suffice in some cases. Since serving involves waiting in queues and wait time can dominate the actual service, queueing theory is widely applied in this area, especially by Kleinrock (references KLEI001, and KLEI718 for computer systems; references KLEI640 and KLEI729 for computer networks) and by S.Stimler (reference STIM690) (**).

Stimler uses queueing theory to demonstrate a relation between throughput and response for several different situations of both computer systems and computer networks. Similarly Kleinrock has been studying many models for both systems and networks. Reference KLEI729 gives a good summary of work on the ARPA network with respect to models, actual measurement, queueing techniques applied, routing algorithms and general response considerations. One

---

(**) S.Stimler uses results from P.M.Morse, Queues, Inventories and Maintenance, John Wiley and Sons, Inc., New York, 1968, while L. Kleinrock shows the development of the majority of his results.

F. System software utilized;

G. Distribution of instruction usage;

H. Size, number and type of I/O initiated by the user's program;

J. Size, number and type of I/O initiated by the system, as the system processes the user's program, and

K. Quantity of main memory requested and used.

To perform a monitor experiment on the workload, the experimenter is well advised to consider the following questions--

What resources are requested?
What resources are needed?
When are they requested?
When are they used?
How long are they used?
When are they released?
How are the resources used?

By answering the above questions for every job step or message, the experimenter is certain to obtain at least the data desired--if not too much. The workload is, of course, a function of time. Because each user or even group of users can never be expected to follow a rigorous pattern when computing, we must consider a certain group of jobs which they apply to the system. these jobs can be classed according to their general characteristics, such as the pattern of their resource requests.

Several measurement groups have classified workload grossly under scientific, mathematical, and text manipulation. AUERBACH Information Incorporated is at present using General File Processing, Sorting, four subcategories under Matrix Inversion and seven subcategories under General Mathematical Processing. Each group was created by considering some average measures of workload characteristics.

However, even the AUERBACH classification is only an aid for finding and applying a test workload to a computer system or network. Even in the AUERBACH reports, we can read warnings about the validity of their workload classification and their benchmark results. To attack the problem of testing a system with a standard workload, there are three different kinds of workloads which could be applied:

A. An artificially generated, standard load permits reproduction of the test. This gives an excellent comparison for the before and after tests, when system modifications are being made;

B. A real load provides no ability to reproduce the test, but it is a test of reality--an experiment using a workload which the actual users supplied. Comparison is possible, but difficult mathematical techniques like clustering and regression analysis are needed to extract meaningful results.

C. A combination of both the real and the artificially generated load permit some degree of the reproduction of a previous workload, and of reality. This is a workload which represents the actual load better than the artificial load being used alone.

Thus, an experimenter should be able to understand, design, implement and apply the above three kinds of workload or at least be aware of the degree of reproducability and reality, and of the difficulties and validity of result comparison for the above types of workload.

3.4 CONCLUSION

This concludes the general discussion of objects which monitor systems should be able to observe. For completeness, a list of specific items is given in Table #3.1. The format of the list parallels the above discussion, but another technique for listing these items is to group them according to which measurement is made on each item. A list in this format is given in Appendix "A". The categories used are time, frequency, count, length, and space and time.

TABLE #3.1.

CHARACTERISTIC PARAMETERS

I COMPUTER SYSTEM PARAMETERS

UTILIZATION OF RESOURCES

A. Frequency of

1. Specific software activity. This includes system software, utilities, and a part or whole of the operating systems of nodes or hosts.
2. Processor activity;
3. Line or link activity;
4. Channel or controller activity;
5. Auxiliary or main storage device activity;
6. Data set activity;
7. Data set structure activity;
8. Processor states, and
9. Instruction execution.

B. Quantity of auxiliary or main storage space requested or used.

C. Quantity of data moved to or from specific devices.

THROUGHPUT

A. Time required to transmit/handle a message/packet through a net, node or other specific resource;

B. Number of messages, packets or jobs handled by a node, net or host;

C. Number of bits transmitted or received by a link, line, node, net or host;

D. Raw speed of a resource, and

E. Time between dispatch of packets, messages or jobs.

RESPONSE

A. Time to set-up or disconnect a logical or physical path through a net or node, and

B. Time required to respond to a call for service.

II <u>WORKLOAD PARAMETERS</u>

A. User response time (or think time);

B. Time between arrivals of packets, messages or jobs;

C. Frequency and types of requests for service;

D. Reference pattern of software;

E. Size of pac6et, message or job in characters, lines or cards;

F. Real time on the system, and

G. Quantities and types of storage requested and used.

III <u>MISCELLANEOUS ITEMS</u>

A. Time for the object system to detect, correct or recover from trouble with data transmission; lines, nodes, hosts or specific devices out of service; software errors, and link problems.

B. Time for the object system to detect saturation of lines, links, nodes, hosts or other devices.

C. Number of packets, messages or jobs within the system and the number of jobs active.

D. Size of a queue.

E. Logical or sequential combinations of the above events.

F. The identification and status of messages, packets or jobs.

MONITOR EVENTS

4.1 THE DEFINITION OF MONITOR EVENTS

The occurrences which either stimulate or require monitoring can be discussed individually, or as classes of occurrences. The occurrences and the actions stimulated by the occurrences can be described simply by creating a structured definition of these occurrences.

While creating the definition, the aims were to be able to obtain:

A. A simple, clear definition which would be easy to use and understand;

B. A logically structured definition which would contain information on the relationship between events and the structure of each event;

C. A structured approach indicating what a monitor should do when the event occurs. This would facilitate the implementation of the seven basic monitoring actions--registering the time of occurrence, timing the duration, counting the occurrences, reading and recording some value, and initiating recovery, correction or diagnosis of a system.

D. A basis for a generalized measurement language.

E. Some insight into the structure of a Hardware-Software Monitor System.

F. A tool permitting easy discussion and presentations of all the monitor events.

G. A definition applicable to all the types of monitors(the monitors are defined in Chapter V). Note that each type of monitor can only observe a subset of all the events defined and each type may consider the events for either measurement information or to stimulate other action or both.

The technique used to define the monitor event is a language grammar represented in BNF (*). The grammar is displayed in Tables #4.1 and #4.2 and is described below. The description of the grammar is presented in three levels (Basic, Intermediate, and Final). Each level of the definition represents a specific function. The existence of these levels clarifies the relationships between the productions.

(*) Naur, P. (ed.); "Revised Report on the Algorithmic Language Algol 60"; Comm. ACM, Vol.6, No.1, pp.1-17.

TABLE #4.1

ALPHABET OF THE EVENT GRAMMAR

TERMINALS

s	-The basic digital signal.
d	-An input standard, provided by the experimenter and used in comparisons.
.	-Boolean binary operator of intersection
+	-Boolean binary operator of union.
¬	-Boolean unary operator of complementation.
=	-Comparison operators; indicating the
<	-respective performance of a comparison
>	-of one value for being equal, below, and
:	-above a standard, or within a range.
( ) ,	-Delimiters.

NONTERMINALS

<N>

<ABOVE>

<BELOW>

<EQUAL>

<EVENT>

<WITHIN>

<OCCURRENCE>

<BOOLEAN_TERM>

<BOOLEAN_ELEMENT>

<BOOLEAN_COMBINATION>

<SEQUENTIAL_COMBINATION>

TABLE #4.2

PRODUCTIONS OF THE EVENT GRAMMAR

```
<GOAL> ::= <EVENT>

<EVENT> ::= ( <SEQUENTIAL_COMBINATION> )
          | ¬ ( <SEQUENTIAL_COMBINATION> )
          |   ( <BOOLEAN_COMBINATION> )
          | ¬ ( <BOOLEAN_COMBINATION> )
          |   <ABOVE>
          | ¬ <ABOVE>
          |   <BELOW>
          | ¬ <BELOW>
          |   <EQUAL>
          | ¬ <EQUAL>
          |   <WITHIN>
          | ¬ <WITHIN>
          |   <OCCURRENCE>

<SEQUENTIAL_COMBINATION> ::= <EVENT> , <EVENT>
                           | <SEQUENTIAL_COMBINATION> , <EVENT>

<BOOLEAN_COMBINATION> ::= <BOOLEAN_ELEMENT> + <BOOLEAN_TERM>
                        | <BOOLEAN_TERM> ∘ <EVENT>

<BOOLEAN_ELEMENT> ::= <BOOLEAN_TERM>
                    | <BOOLEAN_ELEMENT> + <BOOLEAN_TERM>

<BOOLEAN_TERM> ::= <EVENT>
                 | <BOOLEAN_TERM> . <EVENT>

<ABOVE> ::= <N> > d

<BELOW> ::= <N> < d

<EQUAL> ::= <N> = d

<WITHIN> ::= <N> : d,d

<N> ::= <OCCURRENCE> <OCCURRENCE>
      | <N> <OCCURRENCE>

<OCCURRENCE> ::= s
               | ¬ s
```

4.1.1 THE BASIC LEVEL

This first level provides the basic information or stimuli to be used in the combinations or comparisons of the next level.

s Is the terminal symbol representing a single binary signal. This signal is either transient (travelling on a transmission line) or stationary (in a known location, whether fixed or variable). Monitors either detect 's' or use 's' as an interrupt stimulus.

'<OCCURRENCE>' -Is the non-terminal representing the state of 's', either high or low.

```
        <OCCURRENCE> ::= s
                  |   ¬ s ;
```

'<N>' -Is the non-terminal representing the concatenation of the non-terminal <OCCURRENCE> into a string with a length of at least two, independent of how the string is transmitted or stored.

```
    <N> ::= <OCCURRENCE> <OCCURRENCE>
        |   <N> <OCCURRENCE>
```

4.1.2 THE INTERMEDIATE LEVEL

This level represents the use of the basic level to represent further operations which provide a binary output. This entails both comparisons and combinations.

The comparisons are:

EQUAL Is <N> equal to 'd'?

```
     <EQUAL>  ::= <N> = d
```

ABOVE Is <N> above 'd'?

```
     <ABOVE>  ::= <N> > d
```

BELOW Is <N> below 'd'?

```
     <BELOW>  ::= <N> < d
```

WITHIN Is <N> with the two 'd's'?

```
     <WITHIN>  ::= <N> : d,d
```

The logical combination is defined to provide a boolean combination of at least two events, using the two basic binary operators of boolean algebra, and giving priority to intersection over union. At this point, the boolean expression has no length limit, and it represents only a sum of products expression.

```
<BOOLEAN_COMBINATION> ::= <BOOLEAN_ELEMENT> + <BOOLEAN_TERM>
              | <BOOLEAN_TERM> . <EVENT>
```

```
<BOOLEAN_ELEMENT>  ::= <BOOLEAN_TERM>
            | <BOOLEAN_ELEMENT> + <BOOLEAN_TERM>

<BOOLEAN_TERM>  ::= <EVENT>
            | <BOOLEAN_TERM> . <EVENT>
```

The sequential combination is defined to detect a sequence with a length of at least two. The minimum of two for both types of combinations is practical since a length of one is not a combination.

```
<SEQUENTIAL_COMBINATION>  ::= <EVENT> , <EVENT>
            | <SEQUENTIAL_COMBINATION> , <EVENT>
```

4.1.3 THE FINAL LEVEL

Now the event can be defined from the previous productions, as being the state of each of the above. That is, either the True or False state of an occurrence, a comparison or the boolean or sequential combinations.

```
<EVENT>  ::= ( <SEQUENTIAL_COMBINATION> )
           | ¬ ( <SEQUENTIAL_COMBINATION> ) ;
           | ( <BOOLEAN_COMBINATION> )
           | ¬ ( <BOOLEAN_COMBINATION> ) ;
           | <ABOVE>
           | ¬ <ABOVE> ;
           | <BELOW>
           | ¬ <BELOW> ;
           | <EQUAL>
           | ¬ <EQUAL> ;
           | <WITHIN>
           | ¬ <WITHIN> ;
           | <OCCURRENCE>
```

4.2 PROPERTIES OF THE GRAMMAR

Using results developed in (*) "Formal Languages and Their Relation to Automata" and the program developed by W.R.Lalonde (**), several properties of the grammar can be derived. These properties are:

A. Context-free;

B. Self-embedding;

C. Non-linear;

D. Not sequential, and

E. LR(1) (***) grammar (as proven by using the program 'LALR' developed by W.R.Lalonde).

Since the grammar is LR(1), the grammar is also LR(K) and unambiguous, and generates a deterministic language.

(*) J.E.Hopcroft and J.D.Ullman, Addison-Wesley Publishing Co., Toronto, 1969
(**) The program used is documented in W.R.Lalonde; "An Efficient LALR Parser Generator"; University of Toronto, Masters Thesis, Technical Report CSRG-2, February, 1971.

(***) LR(1) means a grammar which is left to right parsable, with one symbol look ahead.

4.3 OBSERVATIONS

Several intuitive observations can be obtained from the grammar. The grammar implies that a monitoring system needs certain functional units:

A. To obtain the state of each 's';

B. To make each of the comparisons for some string of a given length;

C. To permit the experimenter to input the standard 'd' and its mask, since it is assumed that the comparators would be of a fixed length and the desired information may be a substring of the detected string (eg. the op code within an instruction.)

D. To calculate a boolean function;

E. To follow a sequential pattern;

F. To complement the signals, and

G. To control the passing of the information from functional unit to functional unit.

*The techniques available to accomplish the functions listed above, excluding #C, are of three types--either performed by the hardware monitor section (of a hardware-software monitor system), by the monitor's software or by the object system's software. To minimize interference by the monitor with the object system, it is beneficial to avoid the last, but the decision between the hardware and software sections of the monitors, depends on factors like cost, resolution needed, frequency and ease of use, and ease of implementation. For #C, there is a choice, dependent on how the comparators are implemented, between having the experimenter use his control program or use some manual switches on the hardware monitor section.

4.4 APPLICATION

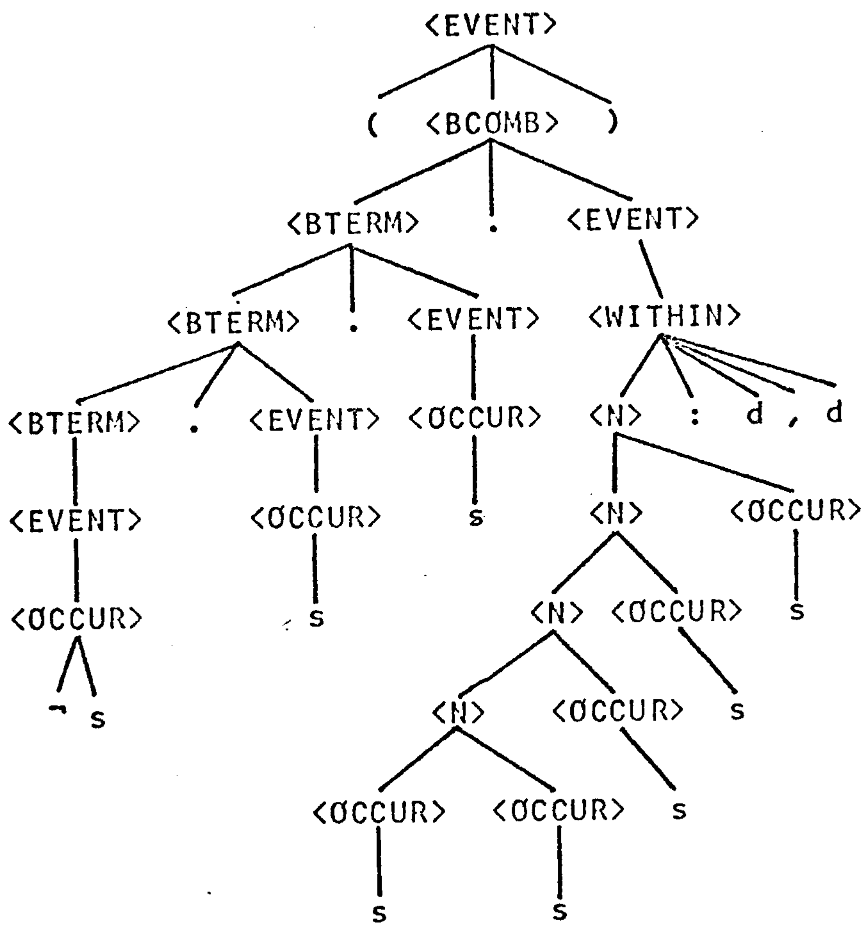
Using the above definition, it is now possible to discuss the items for monitoring as events. Thus Table #4.3 is presented and it contains a list of common items for monitoring, listed as events with the types and attributes of each event.

Consider a hypothetical computer having two channels, a wait light, and an instruction address of four bits. Place a probe on the wait light (label it "WAIT"), one on each channel (label them "CH1" and "CH2") and one on each of the four address bits (label them "S1","S2", "S3", and "S4"). The state when the processor and both channels are busy and when the instruction address is within two limits, say d1 and d2, can be represented by the boolean function:

$[\neg \text{WAIT} \circ \text{CH1} \cdot \text{CH2} \cdot \text{B}]$ where B represents the state when the address is between d1 and d2. The derivation tree of the terminal string representing this event is shown in Figure #4.0.

FIGURE #4.0

SAMPLE DERIVATION TREE



The terminal string would be:

(¬ s . s . s . ssss : d , d)

After replacing each terminal with its appropriate label, the string would be:

(¬ WAIT . CH1 . CH2 . S1 S2 S3 S4 : d1 , d2)

It should be noted that when using the grammar to define an event, the definition depends on the techniques available to detect the event. Thus the implementation will be both object system and monitor dependent. Consider the LDAQ instruction of the Honeywell 6000 (*). Neglect the operands for the instruction; then the combination of events which indicates the execution of the instruction is (**):

(*) Honeywell 6000 Processor Manual, Vol.2; SPEC Chart, Drawing #43D222768, Sheet #2.

(**) The symbols represent the sensing (¢) or strobing ($) of specific registers or logic components. The two registers, RMU/ZJ and RML/ZJ, contain the two pieces of information which are to be loaded into the accumulator ("A") and the quotient ("Q") registers, respectively. This sequence of electrical events is specific to the LDAQ instruction and represents the sequence of events which moves the contents of register RMU/ZJ to A and of register RML/ZJ to Q.

A. ¢RMU/ZJ and ¢RML/ZJ

B. ¢ZJ/ZH

C. $RAS

D. ¢RAS/ZL

E. $A and $Q

The derivation tree of the terminal string representing the event, is shown in Figure #4.1.

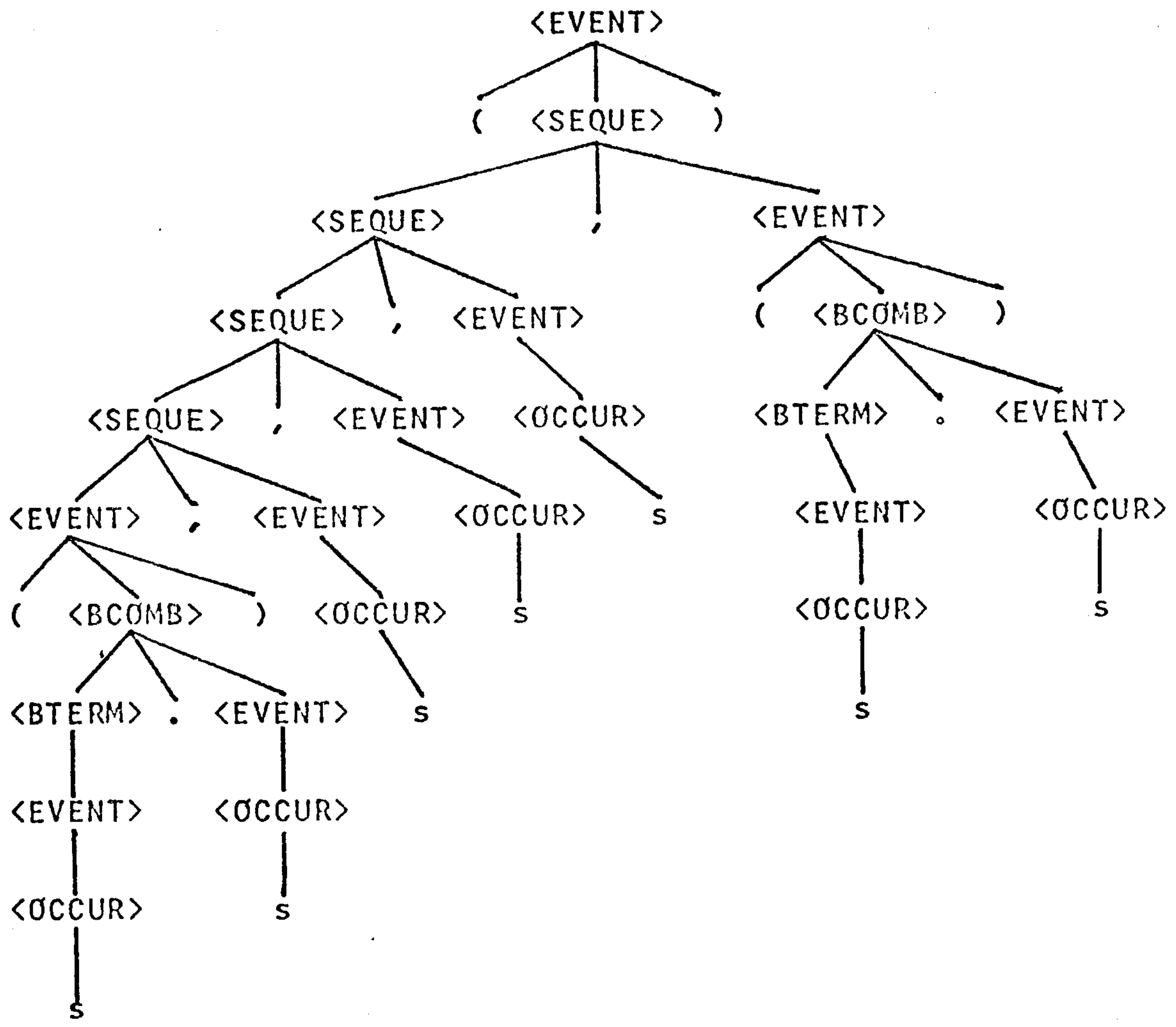
However, if the execution of this instruction is frequently monitored then some extra hardwired logic could be added to the H6000, to make this event available at a test pin. This procedure of providing test pins for a limited number of selected events is followed by Honeywell and a manual which lists the test pins, is available. If detection of the execution of the LDAQ instruction was reduced to observing one pin, then the derivation tree would be simply:

```
  <EVENT>
     |
<OCCURRENCE>
     |
     s
```

4.5 FURTHER WORK

These events all have attributes which require special action to be performed by the monitor. Attempts will be made to enlarge the grammar such that it includes the actions which the monitor should take. The addition of a level to include a monitor's control functions is being considered, and finally the grammar will include features common to Fortran or Algol, such that the language becomes capable of permitting mathematical analysis, flow control, monitor set-up and monitor control. For example, productions will be added such that "s" and "<EVENT>" will generate identifiers. This will permit the user to name his probe points and his events, as well as providing a tool which the system can use to maintain a library of definitions for commonly used events.

FIGURE #4.1
DERIVATION TREE
FOR THE
EXECUTION OF THE H6000 LDAQ INSTRUCTION



The terminal string would be:

((s . s), s , s , s ,(s . s))

If each "s" were replaced by its respective probe label, the string would be:

((¢RMU/ZJ . ¢RML/ZJ), ¢ZJ/ZH , $RAS , ¢RAS/ZL ,($A . $Q))

TABLE #4.3

EVENTS FOR MONITORING

1. Error Occurs

   I TYPE
      A. Data transmission error;
      B. Peripheral device out of service;
      C. Host out of service;
      D. Node out of service;
      E. Line out of service;
      F. Link problem;
      G. Software error in node or host, and
      H. Miscellaneous(eg. transient).

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Count;
      C. Duration;
      D. Notification.

2. Error is Detected by object system

   I TYPE--same as (1).

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Count.

3. Recovery or Corrective Action Commences

   I TYPE--same as (1).

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Duration;
      C. Count.

4. Saturation of a Network Resource

   I TYPE
      A. Line;
      B. Link;
      C. Node;
      D. Host;
      E. Storage (Auxiliary or Main).

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Count;
      C. Duration;
      D. Record a Value;
      E. Notification.

5. Request for a Software Resource

   I TYPE
      A. Accounting;
      B. Message Assembly and Disassembly;
      C. Service request;
      D. System modules;
      E. File System;
      F. Primitives, or Semaphores, and
      G. Link

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Count;
      C. Record a Value;

6. Allocation of a Software Resource

   I TYPE--same as (5)

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Duration;

7. Rejection of Request for a Software Resource

   I TYPE--same as (5)

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Count;

8. Use of a Software Resource

   I TYPE--same as (5)

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Duration;
      C. Count;

9. Request for a Hardware Resource

   I TYPE
      A. Processor;
      B. Peripheral;
      C. Line, and
      D. Memory.

   II ATTRIBUTES
      A. Time of Occurrence;
      B. Count;
      C. Record a Value;

10. Allocation of a Hardware Resource

I TYPE--same as (9)

II ATTRIBUTES
   A. Time of Occurrence;
   B. Duration;

11. Rejection of a Request for a Hardware Resource.

I TYPE--same as (9).

II ATTRIBUTES
   A. Time of Occurrence;
   B. Count;

12. Use of a Hardware Resource

I TYPE--same as (9).

II ATTRIBUTES
   A. Time of Occurrence;
   B. Duration;
   C. Count;
   D. Record a Value;

13. Request for Storage

I TYPE
   A. Main
   B. Auxiliary.

II ATTRIBUTES
   A. Time of Occurrence;
   B. Count;

14. Allocation of Storage

I TYPE--same as (13).

II ATTRIBUTES
   A. Time of Occurrence;
   B. Duration;
   C. Record a Value;

15. Rejection of a Request for Storage Allocation

I TYPE--same as (13).

II ATTRIBUTES
   A. Time of Occurrence;
   B. Count;

16. Use of Storage

I TYPE--same as (13)

II ATTRIBUTES
   A. .Time of Occurrence;
   B. Count;
   C. Duration;
   D. Record a Value;

17. Processor State

I TYPE
   A. Supervisor;
   B. User, and
   C. Nucleus.

II ATTRIBUTES
   A. Time of Occurrence;
   B. Count;
   C. Duration;

18. Instruction Execution

I TYPE--many.

II ATTRIBUTES
   A. Count;
   B. Duration;

19. Boolean Combination of the Above.

20. Sequential Combination of the Above.

21. Process Initiation

I TYPE
   A. Job, and
   B. Message.

II ATTRIBUTES
   A. Time of Occurrence;
   B. Count;
   C. Duration;
   D. Record a Value;

TOOLS AND TECHNIQUES OF MONITORING

5.1 INTRODUCTION

Once experimenters know why and what they wish to observe, they need tools to make the observations. With the tools, they need an instruction manual or some techniques for using the tool and for evaluating the results. People who monitor computers have had many tools built to help them. For obvious reasons, most hardware monitors to date have been built to observe the performance of IBM computers, and only a few monitors have been built to observe the behaviour of equipment from other manufacturers. Many of these monitors were designed and built by IBM, in fact. The inventors of monitor systems cannot be blamed for favouring such a large company, for this insures a large market and it is very difficult to construct computer monitoring tools which apply to a large number of different types of hardware. The number of configurations, the types of tools, the resources needed for monitoring and the design complexity of monitoring systems are just a few characteristics of computer monitoring tools and techniques which are discussed below. In this chapter, the types of monitors are defined and examples are presented, while in Chapter VI the characteristics and limitations of the various types are analysed.

5.2 HARDWARE MONITORS

Hardware monitors are the class of monitoring tools which use strictly a hardware system or a computer system which is completely separate from the object system. There are a great many characteristics of these tools, but one of them--the intelligence--provides a basic division for classification purposes. The level of intelligence of a monitor is dependent on the level of self-control which the monitor has. The first hardware monitors were unintelligent devices being controlled manually by human intervention or at most by timed interrupts or overflow signals. Soon the need was recognized for a monitor which could control itself, react to electrical stimuli and make decisions on a dynamic level, so hardware monitors were designed with patch-board programmable logic, and recently with the capability of being run under minicomputer control.

Table #5.1 is a summary of all the hardware monitors about which information has been published in English. They are listed by the date of design, if known, or by the date of the earliest documentation. All the available documentation which either describes a monitor or the use of a monitor in an experiment is listed for the appropriate monitor. Also listed is a summary of each monitor's characteristics. Since the ultimate hardware monitor should be able to observe computer networks, an attempt has been made to determine how much effort would be required to modify the documented monitors to make them useful for monitoring computer networks. Our estimate is given with each monitor.

The estimation follows a hierarchical pattern. There are four groups, and each group needs specific changes plus all the changes for the groups listed before it. The first group is denoted as needing "TRIVIAL" (*) modifications. These modifications involve software changes, which include the writing of software monitors for execution within the object system, the possible writing of an improved operating system for the monitor and the writing of data reduction routines. This group also needs a technique for synchronizing the clock at each network node. The second group is denoted as requiring the "MINOR" addition of some real time response capabilities, such as programmable logic. It also requires the modifications described under the "TRIVIAL" class. "MAJOR" modifications required by the third class includes the modifications described above, the addition of intelligence such as a controlling processor, and at least one of: better storage, improved resolution, comparators, and a clock. The last group requires "COMPLETE" redesign.

(*) Note that the keyword for each class only denotes the work required with respect to the other groups, but in no manner does it indicate the actual amount of work required.

NAME	REFERENCES	YEAR	AFFILIATION	COUNTERS	COMPARATORS	INTERFACE	RESOLUTION	LOGIC	OUTPUT RECORDING	INTERACTION WITH MONITORED SYSTEM AND CONROL.	REVISIONS TO BECOME A NETWORK MONITOR
HARDWARE MONITOR	PATR647 WARN708	1961	IBM	6 high speed, 6 mechanical		hardware	1 ms.	wired	camera or by hand		MAJOR
CHANNEL ANALYZER	WARN708	1962	IBM	0	0	HARDWARE	?	?	?		MAJOR
PROGRAM MONITOR	WARN708 APPL658	1962	IBM System Developement Division, Poughkeepsie	0	0	HARDWARE	?	?	magnetic tape		COMPLETE
POEM	WARN708	1963	IBM	0	0	?	?		magnetic tape		COMPLETE
PEC	WARN708	1964	IBM	High speed		Hardware	?	plugboard	magnetic tape		MAJOR
EXECUTION PLOTTER	WARN708	1965	IBM	0	0	Hardware	?	none	CRT		COMPLETE
SAMI	WARN708	1967	IBM	64	32	probes	?	plugboard	matnetic tape		MAJOR
SNUPER PHASE 2.	ESTR670	1967		variable	variable	probes	?	Software	display and communications	SIGMA-7 CPU	TRIVIAL
TS/SPAR	SCHU678	1967	IBM	48 electric	some	256 probes	?	plugboard logic cct.		operators console and digital signal	MAJOR
BCU	WARN708	1968	IBM	64	32	probes	?	plugboard	magnetic tape		MAJOR
CPM	CARL718 KOLF680	1968	ALLIED COMPUTER TECHNOLOGY	16-20 12 digit	0	20 probes	1us.	Plugboard	Visual and magnetic tape	logical signals from Host.	COMPLETE
CPM II	MILL717 HART718			32 of 10 digit	0	40 probes	50ns.	450 hub plugboard	Magnetic tape	"	COMPLETE
CPM III	HART718			16 of 10 digit	0		50ns.	300 hub plugboard	Magnetic tape Plotter	"	COMPLETE
CPA	CANN729 KOLE690 CARL718	1968	COMPUTER AND PROGRAM ANALYSIS INC.	6-18 mechanical	0	?	?		Visual		COMPLETE
SYSTEM MONITOR	BONN690	1968	IBM	16	0	probes	1us.	plugboard logic panel	display and card punch		MAJOR
SLUR	MURP690	1968		variable	variable	probes		Associative Memory and Secondary Storage	to monitored system	program in A.M.	TRIVIAL
SUM	CANN729 KOLE690 CARL718 WARN708 HART719	1968	COMPUTER SYNECTICS INC.	16 of 9 digit.	0	20 probes	50ns.	300 hub plugboard	magnetic tape		MAJOR
UNIVAC INSTRUMENT	GORD714 KOLC690	1968		1	0	4registers probes	0.5us.	wired	drums and tapes		MAJOR
CDM	GRCC690 SALT69A	1968	MULTICS	0	0	GIOC of MULTICS	0.05s	software	CRT or printer	Communicates with GIOC of MULTICS.	MAJOR
EVENT MONITOR	HART718	1970	[illegible] AND	6 of	0	32 probes	30ns.	148 hub	Plotter, tapes, ?		[illegible]

MARDAC.	BOYL735 KOHN717	1970	U. OF WATERLOO ONTARIO,CAN	0	3	..obes as needed	samples	logic board	metered and strip chart		MAJOR
XRAY	CARL718 HART718	1970	APPLIED SYSTEMS	16 of 9 bits	0.	96 probes	50ns.	480 hub plugboard	magnetic tape		MAJOR
A COUNTING MONITOR	KLAR700	1970	U. OF ERLANGEN GERMANY	four of* 18 bit	0	probes	?	OR multi-event control unit	variable	variable	MAJOR
CMSM	HART718	1970	COPAC INC.	32 of 9 digit	?	48 probes	30 ns.	240 hub plugboard	Tapes,printer		COMPLETE
DYNAPROBE	CANN729	1970	COMRESS								
7700	DYNA734 DYNA730 DYNA732 DYNA735			6 to 18 electrical and electro-mechanical 12 digit	Modules of 2 compara-tors at 32 bits.	6 to 18 probes 30 ns.	100 to 171 ns.	plugboards on most modules for 7700,7900	Buffered printer with Real Time Clock.	Manual control module.	MAJOR
7900	DYNA733			16 of 10 digit counters	"	0 to 32 probes 30ns.	100 ns.	"	Magnetic tape. Printer. Single counter display.	Manual contol module.	MAJOR
8000				?	?	48probes 30ns.	30ns..	logic board with 60 hubs.	65K Memory. Tapes.	Minicomupter with TTY.	MINOR
NEURCTION	AMIO729 ASCH710 ASCH710	1970	STANFORD UNIVERSITY CALIFORNIA	?	?	?	?	program logic	display and magnetic tape	minicomputer controlled	MINOR
DSP-1		1972		?	?	probes and I/O controller	?	?	display and to monitored computer	monitored system controlled	MINOR
ADAM	HUGH731	1972		128-32bit	?	100 probes	?	?	stored or printed.	monitored system controlled.	MINOR
TESDATA 1010	SYST730	1972	TESDATA (SUM)	16 to 20	?	10 to 20 probes	50ns.	optional plugboard	CRT,line printer,or	Operator controlled.	MAJOR
TESDATA 1155	SYST720 SYST730	1972	TESDATA (XRAY)	8 to 32	?	32 to 96 probes	1us.	plugboard	matnetic tape CRT	Manual control panel.	MAJOR
TESDATA 1185	SYST720 SYST730	1972	TESDATA	0 to 32	?	36 to 144 probes	100 ns.	Collector Groups	magnetic tape, CRT,Line	keyboard, control panel.	MINOR
TESDATA 1200	SYST730	1972	TESDATA	0 to 32	?	36 to 144	100 ns.	Collector Groups	magnetic tape, CRT,Line printer,disk.	keyboard and control panel.	MINOR
NAMELESS	BURK732	1972	MITRE Corp.	Basic Components are an Event Detector, Comparator, Interpretor and o/p Buffer. Utilizes micro-programming and Host's storage. Meant to be designed into the Host computer as a user resource.							
INSTRUMENTATION for C.mmp	FULL732	1972	CARNEGIE-MELLON UNIVERSITY	4 counters	4 groups of comparators	probes and unibus	?	None.	Interfaced to controlling CPU.	Minicomputer controlled.	MINOR

5.3 SOFTWARE MONITORS

Software Monitors are programs designed to execute and utilize the resources available on the object computer. There are three general classes of software monitors: accounting routines; malfunction detection, diagnostic, and recovery routines; and performance monitors.

Accounting routines perform the simple task of recording what a consumer uses so that he can be charged for it. Some accounting routines only charge for elapsed time, while others perform a more detailed analysis permitting charging by processor time, by lines printed or per access to a data base(eg. Account PAK of Systems Dimensions Ltd., Ottawa, Canada).

The software monitors concerned with malfunctions, control the detection of faults and errors, and initiate any of the correction, recovery or diagnostic routines which may be needed. These monitors normally do not perform any measurement monitoring, but they may count the number of occurrences of each error or fault, or do some accounting, as does No.1 ESS (reference DOWN650). Note that aberrations of performance indicators(eg. poor turnaround) can also indicate malfunctions.

Performance monitors are designed to collect performance data similar to that collected by the accounting routines, but more kinds of data are of interest. For example, performance monitors may measure the number of times a software unit or instruction is executed while accounting routines usually are not concerned with such data.

It would be a difficult task to create a chart, similar to Table #5.1, for software monitors since the number of accounting routines is quite large, but few are documented. However a description of twelve monitors is provided in volume eight of reference AUER736. This description includes for each routine, its cost, the output provided, the object systems on which it will execute, a basic description of operation and a brief list of data items collected. An example of the use or implementation of each of the three types follows, but for other discussions on software monitors see references (ARDE70A, BARB708, CANN729, COOP700, COOP729, DENN690, DEUT710, HERM640, HERM672, KEEF680, KOLE690, PINK690, SCHE664, SCHW720 and SEDG70B).

5.3.1 ACCOUNTING

Accounting systems are the most common software monitors available. The main difference between these and other software monitors is the set of data which they measure. Accounting systems restrict their data to items concerning resource usage by a consumer. The attributes of the monitored event are:

A. The access frequency(ie. accesses per unit time) of each resource used;

B. The identifier data of a programmer ID or project number;

C. The quantity of the computer system resource used; and

D. The initiation and termination times of the resource used.

Considering that the accounting systems measure a subset of the data gathered by the main class of software monitors, the produced statistics can actually be very complete. Some of the specific statistics derivable are the start and stop time of each job step; the core requested, and the core used; processor time used; the quantity of I/O performed in units of the number of cards read or punched, lines printed, and the number of datasets accessed on disk, tape, drum or cell; the number of I/O calls per dataset; processor wait time; the number of jobs or job steps processed; elapsed time per job step; job step cost; and various combinations of the above data, such as process time per elapsed time or histograms of core requests and accesses, as well as averages of the above.

From the above data, much can be learned, but the analysis is horrendous. In references WATS715 and LEHM735, experiments are discussed which use only accounting routines to provide the data for measurement and evaluation of their computer systems. In WATS715 the only modification required was the saving of all the accounting data on magnetic tape for later reduction. The extra items determinable from the data were processor idle time or down time, number of jobs in memory, quantity of memory allocated and occupied, processor activity, the average number of jobs on the computer at any time and finally their revenue, response time and throughput. The analysis task was described thoroughly and confirmed one of the major disadvantages of software monitors--the analysis used regression techniques and cluster analysis as well as data selection packages and statistical packages, and required four to six man months for the total analysis. This quantity of analysis and data massaging could be expected to be smaller if a hardware monitor had been used.

Some of the conclusions which can be drawn from reference WATS715 and which concern accounting systems in general are that accounting systems are useful for:

A. Measuring the effect of system modifications on system performance;

B. Providing intricate statistics on the workload of an experiment;

C. Formulating or revising charging schemes;

D. Providing the installation's management with a report on computer utility and workload characteristics;

E. Providing measurements and descriptions of workloads processed by the system in the design of typical job streams (ie. input for benchmark, simulation and modelling studies), and

F. Showing the trends of past usage and performance.

5.3.2 MALFUNCTION DETECTION

Software monitors can perform malfunction detection, diagnosis, correction and recovery. They are normally used alone, but special hardware devices are commonly used to aid the malfunction detection. The special hardware detects the state of various modules such as the 'Host ready' and 'Node ready' flags of the ARPA network, or does simple state calculations such as parity checks on a transmission line. The software monitor samples the status of these hardware units, and the other hardware units, such as processors or peripherals, and compares the status to known standards. If the status indicates trouble, the software monitor performing the check will initiate some predetermined action. The action chosen depends of course on the system under consideration.

There are two good examples of software monitor systems for malfunction control in use at present. One is part of the network control built into the ARPA network and the other is NO.1 ESS, which is used by the Bell System to switch telephone calls in many areas of the United States and a few areas of Canada (*). Monitoring for malfunctions of the ARPA network entails two simple functions. First is parity checking on transmission lines. The second is a watch-dog timer which times out if not reset by the normal packet handling routines and passes control to a routine in protected memory which assumes a software error, attempts to reload a new copy of the node's operating system and if successful, restarts the proper routine. The ARPA network is a good example of software monitoring for performance purposes and is discussed in more detail later.

(*) Reference Bell Systems Technical Journals of 1965 and 1970.

NØ.1 ESS, however, is an elaborate system designed not only to service or handle the telephone calls made, but also to maintain an efficient working configuration of the hardware which it controls. As a functional unit servicing the public, NØ.1 ESS controls or performs the detection of a service request, the interpretation of the digits dialed, the alerting of the called customer, the establishing of a talking connection, the detecting and performing of a disconnection and the accounting for the call, if required. As well as the call processing and accounting, it performs traffic measurements, and accepts data from a control Teletype, which permits changes to the classes of service and the directory numbers assigned to the lines.

However, the point of interest is the malfunction control of the NØ.1 ESS Maintenance Plan. The principal features of the maintenance plan include:

A. The conservative circuit designs and long-life components which are used to obtain reliable units.

B. The redundant units which are used to provide service in the presence of failures and during routine preventive maintenance. All the hardware equipment is duplicated, including the central processor. One part is considered active and its duplicate is on standby status. While on standby, if operational, it is functioning in parallel with its active duplicate and provides another level of status checking; otherwise it is labelled inactive and is awaiting diagnosis or repair.

C. The rapid detection of faults or errors by continuous hardware and software checks. This is necessary to permit restoration of service and to prevent incorrect information from propagating into other units of the system. The checks are provided in many ways. For example, the buffer store and buffer controllers have parity checks during transmission, status reports from each unit, match checks (similar to techniques described above) at the buffer control, and an "all-seems-well" response from each unit to the buffer controls. Besides the many checks, the detection system is designed for speed by having hardware circuitry perform the long time, continuous functions and the functions easily implemented in hardware, such as continuous trouble detection, automatic retrials, administration of coarse program priority, switching of duplicate units and accessing hardware status flags (probes or hardwired interfaces), while software performs the rest.

D. The recovery procedures by fault recognition programs which are designed to preserve message information while testing and configuring the system around faulty units. The Maintenance Plan works with three levels of programs, of which the recovery programs are the first level and have priority over the other two.

E. The error analysis programs which are used to distinguish between occasional errors and marginal or intermittent faults. Every time a fault or error occurs, a count for the specific occurrence is incremented. This provides data on the frequency of occurrence of specific troubles and pinpoints specific areas for further maintenance or even replacement.

F. The diagnostic programs are automatically interleaved with message processing programs to isolate faults to replaceable, plug-in, circuit packages. Once the detection circuitry has established with some certainty that trouble exists, control of the system is transferred, to programs that analyze the problem, determine an operational configuration and control the switching to establish the new configurations. This analysis is logically complex, but occurs infrequently, and the low overhead permits a software solution. The diagnostic routines which perform the tests and interpret the results, constitute the second level of the maintenance programs.

G. The in-service checks provide rapid detection of faults and marginal troubles. Exercise programs work the infrequently used systems and run periodic line tests. These exercise programs are the third level of the maintenance plan and execute only when the system workload is very small.

In summary, the system follows a chain of events like this: first the monitor scans status indicators at rates from 100 to 10 cps. or irregularly when needed; secondly the interrupt caused by a mismatch gives control to a recovery routine; thirdly control goes to a detection routine which does at least one of: repeat the failed operation, increment the error count, report the trouble to operational programs via software queues or interrupt the normal processing with a maintenance interrupt. Depending on the results of the recovery routine, diagnostic routines may or may not be queued to execute. The flow of control can be found in more detail on page 2016 of reference DOWN650. Another reference for NO.1 ESS Maintenance plan is AITC707.

5.3.3 ARPA NETWORK MEASUREMENT SYSTEM

The following is quoted from the abstract of reference COLE71A and is an excellent summary of the involvement of the ARPA network with software monitors.

> "The ARPA (Advanced Research Projects Agency) computer network involves the interconnection of about twenty (as of 1971)"--forty as of August 1972--"different research computers across the country by means of a store--and forward message switching net....Since there has been little prior experience which directly relates to the design of such a network, an extensive measurement and evaluation capability was included in the message switching computers (the Interface Message Processors or IMPS). UCLA was designated to be the Network Measurement Centre with the responsiblity of defining the measurements that were necessary for the support of the analytic and simulation model activities, and to determine the performance of the network by the use of these measurement facilities. The primary concern of this report, has been with the development of such a measurement capability and the utilization of this capability to create (and iteratively improve) analytic models of the network behaviour as well as the true system parameters.
>
> The measurement facilities which were designed into the network included: accumulated data such as histograms and totals; snap shot data relating to queue lengths and routing information; and traces of message flow through the network. Any of these measurement routines can be selectively enabled at one or more of the network IMPS to avoid an excessive data collection and artifact problem. In addition to the selective control over the measurements, artifact was further reduced by the careful selection of the variables to be measured, and by the development of measurement techniques such as the use of non-uniform (logarithmic) scale histograms for data which was expected to have an exponential-like distribution."

The above reference also includes reasonable discussions on experimental technique, output format of results, the effect of measurements on network performance. Using mainly this reference, but with aid from other references (BØLT711, BØLT71A, BØLT71C, MCQU72C, HEAR705, KLEI640, KLEI725, KLEI001, KLEI729) the four measurement routines--Accumulated Statistics Snap Shots, Traces, Arrival Time Data, Ten Second Summary--and the Status Reports are described here.

All of the measurement tools are implemented in software, are remotely controlled from a Host on the network (usually UCLA, which is Network Measurement Centre (NMC)), are activiated by either a hardware clock, a packet departure, or a packet arrival, and are activated only at specific nodes; thus, the experimenter must specify which measurement routines are to be active at which nodes.

The Accumulated Statistics program takes data over 12.8 second intervals and sends the data to the NMC. The specific data collected are the lengths of messages and packets on Host-to-Imp, Imp-to-Host and Imp-to-Imp lines, plus the numbers of ACKs, RFNMS, input errors, retransmissions and total words sent. The reduced data output at the NMC includes histograms of message size status, round trip statistics, message totals, channel activity and the packet size statistics. This routine runs only if specified for a specific node, takes 10 to 20% of the node capacity (MCQU72C) and generates a noticeable amount of communications traffic.

The Snap Shot Statistics routine (one per node) runs continuously, but only sends messages to a specific Host on request. The routine is activated by a timer interrupt every half second to take summaries of the internal queue lengths and routing information. The queues measured are the Task, Output, Sent and Reassembly queues(see ARPA references COLE71A, HEAR705, MCQU72C). This information provides an instantaneous picture of the node status and is used to update the node's own routing table or is combined with snap shot data from other nodes to give an overall picture of the network's state. However, there is no attempt to synchronize the snap shots, and thus the combined data is only approximate for any time instant.

The next two tools are the Arrival Time Data and the Ten Second Summary. The Arrival Time Data is collected by saving the time of arrival of a packet from any source ARPA experimenters use only the first sixty arrivals within each 1.6 second measurement interval. The Ten Second Summary records only the number of processed packets of each type, number of retransmissions per output line, and the quantity of local Host traffic.

The purpose of the last measurement routine--the Tracing package--is to permit extensive tracing and timing of a packet's flow through the network. When a node is activated to record tracing data and to send the data to a specific Host, and when a packet arrives with its special trace bit set on, the node records the arrival time, the departure time, which queues the packet is put on, time on each queue, reception time of the acknowledgement message, the output channel and the header of the packet. This routine is quite useful for gaining insight into the workings of the ARPA routing algorithms and the effects of the load on the routing.

The Status Reports are not directly concerned with the network measurement, but are implemented to make the Network Control Center (NCC) at Bolt, Beranek and Newman Inc. (BBN) aware of the status of the lines, Nodes and Hosts on the network. The periodicity of the status messages being sent to the NCC depends on whether a significant change has occurred at the specific node in question. If the status is unchanged, then the ten word binary status packet is sent every fourteen minutes, otherwise, with a change, the report is sent at 52 second intervals. BBN keeps a light panel reflecting IMP, Host and Line status on sight, and records a written copy of the status message which is used to influence maintenance and repair needs.

There is a second status message which is sent by a Node to another Node if it either detects a Node or Host status change or if it is warned by a message from a host of a future status change. This Inter-Imp communication permits special control of messages destined for an inactive Node or Host. This permits a Node to discard packets enroute to downed IMPs or Hosts; to reject the receipt of messages from a Host, when the messages are destined for an inoperative Host or NØDE; and to permit messages to enter the network when the status has returned to normal.

Thus, it is obvious that the implementers of the ARPA network have a considerable amount of monitoring software within their network, and they can obtain an impressive amount of data. The disappointing features are the time resolution--message delay must be measured as round trip time with a resolution of 0.1 ms. and the arrival times are restricted to a resolution of 0.1 second--and the artifact generated by the monitoring routines. A great deal of effort has gone into reducing the artifact, but it also involves reduction of the quantity and types of data measured. It would be a difficult task to monitor message delay and its dependancy on other factors such as the load, time of day, and message length, when the system is near saturation or only lightly loaded!

5.4 HARDWARE-SOFTWARE MONITOR SYSTEMS

A Hardware-software monitor system is a tool which should be developed to overcome some of the shortcomings (explained in Chapter VI) of using either a hardware or software monitor alone. Such a tool should also include many of the advantages of the two, but gains made by this union should become clear in the discussion of monitor characteristics and limits. However, there are some design guidelines (HOLT714) which should be stated:

A. The monitor system must produce accurate, significant measurements which provide system and applications programmers with an objective basis for improving system throughput.

B. Measurements must be output in such a manner as to be both legible and interpretable.

C. The monitor system must be able to monitor any job in the computer, at the request of the monitor operator. The monitor system must supply performance data about the object system and user program to the operator, as requested, without requiring any modifications to the program to be measured.

D. The monitor system's design must allow for easy installation by people not necessarily skilled in the use of the monitor, but of course knowing the object system.

E. Data collection must be thorough, but minimized to reduce any monitoring effects and any unnecessary data reduction or analysis.

F. Monitor initiated data transfers should be minimized, especially within the object system.

CHARACTERISTICS AND LIMITATIONS OF MONITORING TOOLS

6.1 INTRODUCTION

The following discussion should help to clarify the advantages and disadvantages of the three types of monitors, as well as elaborate on the differences and the similarities of the different monitor systems. Where possible, the discussion will elaborate according to the subclassifications presented in Chapter V, but primarily only the basic three categories are involved.

6.2 COST

Software monitors are the least expensive. Accounting routines are purchasable at prices from $2,000 to $5,000 and performance monitor packages range between $3,000 and $25,000(CARL718). Software packages are cheap enough to be purchased by most installations, but they may also be leased. The majority of installations will find that most hardware monitors are too expensive to be purchased. Some 1971 purchase prices for hardware monitors, as listed in reference HART71B and CARL718 are:

'DOLBY'	$8-9,000
'EVENT MONITOR'	$8,000
'CPA'	$6-15,000
'DYNAPROBE' 7900	$27,000
'SUM'	$35,000 & up.
'CPM II'	$35,000 & up.
'XRAY'	$66,800

These purchase prices can be compared with a hardware monitor, equipped with a magnetic tape, rented from IBM at $18,000.00 per week(CARL718). Examination of Table #5.1 shows that the power of the above hardware monitors varies with their costs.

6.3 CONTROL AND INTELLIGENCE

Since software monitors are programs, they are as intelligent as their designers, but they have restricted means of gaining control. Software monitors are either part of the event being monitored (coded within a software event and the monitor code is executed as part of the event) or they are event driven, which means they gain control after interrupts such as a timer interrupt, (*) hardware interrupt, or software call.

Similarly, hardware monitors use interrupts, but primarily for timing purposes--usually not to be able to obtain data. Hardware monitors which have some internal control, use clock pulses for master controls such as time to record, turn-on, turn-off, change experiment set-up, etc. The resolution necessary for this is certainly dependent on the experiment and the monitor's storage capacity and counter magnitude. Resolution is discussed later under Data Collection in Section 6.9.

(*) The use of a timer permits software monitors to sample an event's status.

Besides timing pulses, the means of controlling hardware monitors are varied. 'SUM' is manually operated. The state of 'A Counting Monitor' is changed manually or by digital signals. 'ADAM' is loaded with a program from the object system and then operates independently. 'GDM' has its own special operational mode for the writing of control programs, while those monitors controlled by minicomputers usually permit use of a terminal device for program creation and monitor control.

Hardware monitors may have software, software controlled hardware or manually operated logic units. Logic is discussed in depth in Section 6.9.3.

6.4 FLEXIBILITY

Flexibility involves both the ease of modification of the monitor and its ability to measure various object systems. Software monitors are very poor in this area. They are normally designed for a specific release of an operating system and a specific hardware configuration, making them "one-system" monitors. Some companies, such as Boole and Babbage or Comress have written monitors for many configurations, but the emphasis has been on IBM installations.

Dependable modification of software monitors is as difficult or as easy as the modification of any software, but again the configuration which it measures strongly affects the monitor's design. Thus software monitor modification requires complete knowledge of the object system and the monitor system before changes can be designed, implemented and debugged. Hopefully, the original designer of the monitor is present (which is almost never) or else the monitor system was left in a well-documented form and was written in a language that is easy to use.

In general though, the control software of a software monitor, if written in a high level language, can be transported from machine to machine. If the machines are not indentical then the modifications may only involve rewriting the data gathering routines.

The modification of an hardware monitor is an electronic problem and the degree of complexity varies from monitor to monitor. Some, like the 'DYNAPROBE' (CANN729) line, are very modular, while others like 'SAMI' (WARN708) are huge, awkward, and possess outdated electronic components. The majority of hardware monitors, like 'SAMI' are single units, which require an electrical expert to perform rewiring tasks when modification is needed, while Dynaprobe's monitors can be plugged together to form a variety of monitor configurations. Note, however, that any monitor may require some rewiring sooner or later during its use.

Most hardware monitors, though, do have the advantage of adaptability to many computers. The replacement of the hardwired interface by the sensor-probe in 1966 greatly aided this. Once appropriate signal conditioning have been chosen, all that need be done is to test the object system with the monitor hooked on and operating, then execute an experiment to validate the data obtained from the probes.

However, after reading references BURK732 and FULL732, and after studying the monitors 'GDM' and 'ADAM', it appears that some designers consider the creation of a monitor specifically designed for a given object computer, to be a worthwhile concept. This is only reasonable, if the function of the monitor is simple and the cost of the monitor is almost negligible when compared with the cost of the computer.

6.5 SELECTIVITY

Data selectivity is an important feature. It is a major problem with software monitors and not minor for hardware monitors. Changing the data collected by a software monitor normally entails modification of the monitor, and the obvious thing would be to make the software monitor modular, such that each module collects a certain set of data. Then the experimenter can indicate to the supervisory monitor which set of modules to activate when measurement is needed, and add more modules if the set of collected data es to be enlarged. Recall from Chapter V that the ARPA software monitor consists of four sections in each node and that each section per node and each node could be activated individually.

However, it appears that few monitors were designed in this way, thus modification of data collection becomes concerned with how data is obtained from the operating system. Table driven operating systems permit easy tracing of the active software units and thus are the easiest for monitoring, but when the need for software hooks within the operating system arises, due to the difficulty of modifying and debugging operating systems, it becomes a very complicated procedure.

The use of probes grants some advantage to the hardware monitor. Moving a probe from point to point is rather easy, if a skilled technician is available. Probes can now automatically adjust (within limits) to their electrical environment (DYNA733) and decrease the need for the experimenter to display his received pulses on an oscilloscope. The choice of measurement points is discussed under Data Validation (Section 6.8).

Since this is not the only means of selectivity and since probe set-up can be a problem, it would be nice to have all the needed probe points hooked onto probes. The experimenter could then be selective after he had received the information, but before he had done very much analysis and any storage. A prime example of a hardware-software monitor designed for selectivity is 'SNUPER', Phase II. The first data selection occurs in a module designed to pass the object system's state only when a certain event(s) occurs. This data is buffered, then a primary event table is used to determine if this data is of interest and what to do with it if it is. Thus 'SNUPER', would use two stages of selection before data storage, counting or timing.

6.6 FACILITY

The ease of use of a monitor concerns:

1. The amount of experience which an experimenter needs with the object system;
2. The complexity of operation of the monitor;
3. The number of systems personnel needed;
4. The experimental set-up, and
5. The data analysis required.

Data analysis is discussed below (Section 6.10), so that only the first four items are discussed here.

A software monitor, once debugged, is relatively easy to use; it involves the execution of a program which either does the monitoring or controls the monitor programs which already exist. The most complicated example would be for the case in which the object computer's operating system executes as a job under control of the monitor system. This would probably require a system shut-down, bootstrap of the monitor system, and then restarting the operating system as a task on the monitor.

The use of hardware monitors is different though, and the gain of adaptability corresponds to a loss of facility, for the probes must be connected at the initial set-up and perhaps even reconnected during an experiment. This requires knowledge of the probe points and a system engineer would be very useful, even if he were nervous about the extra electrical connections. Thus, there may be a cost for each experimental set-up. However, if enough probes were available, only the initial probe connection would be required.

Hardware monitors usually require a considerable amount of learning time. Monitor manufacturers have been willing to provide courses (usually about three weeks in duration) and continuous consultation. However, G. Carlson of Brigham Young University mentions that only one operator per installation is normally sufficient for full operation. However, a high level language like the use of an extended Cobol for 'ADAM' (HUGH731), probe point listings and manuals help considerably.

6.7 OBJECT SYSTEM-MONITOR INTERACTION

There are several degrees of object system-monitor interaction, but software monitors are normally the worst degraders of the object system's performance. This is natural, since the software monitors execute solely within the object system and utilize only the object system's resources, such as core, drums, disks, tapes and the processor. The monitor overhead averages 1-5%(CARL718), but has been observed to be as high as 45%. The interference is of course dependent on the monitor's rate of sampling, amount of data gathering at each interval, reduction of data occurring per sample, and storage of raw data.

Hardware monitors have generally been designed not to interfere with the object system, however, systems like 'TS/SPAR', 'CPM', and 'A Counting Monitor' were designed to receive digital control signals. These signals naturally can originate from the object system and thus the experimenter can utilize the object system's timer and language facilities to control his experiment.

A unique example is described in reference MILL717 whereby, with some rewiring of a CDC 6400, the contents of a particular register could be changed and used to feed control information to a 'CPM II'. A reverse situation can also occur with 'TS/SPAR' and 'A Counting Monitor'. These monitors look like terminals to the object system and can send digital signals to the object system, causing it to perform some predesignated tasks.

The degree of interference for 'SLUR', 'DSP-1', and the 'Instrumentation of C.mmp' also involves the object system's storage facilities. These three monitors collect their data and periodically transfer data blocks to the object system. The object system can either examine and perform some reduction or just store the data. This technique of sending data back to the object system may cause some degradation of the object system, but it has the excellent advantage of permitting the object system to accomplish some real-time system tuning as a result of the measured data. Proper programming within the object system can utilize this feedback and accomplish substantial gains in performance.

Naturally we have for hardware-software monitor systems a combination of the interactions previously discussed--the object system and the monitor system both exchange information. Thus there exists a level of communications protocol between the monitor and the object system. This may entail, as for 'GDM' and 'DSP-1', software within the object system to collect the data and initiate the data transfer. 'GDM' is attached to the GIOC (*) of MULTICS such that the GIOC thinks that it is just writing to a terminal. When the 'GDM' desires data, it interrupts the GIOC which accesses the non-transient system data base modules (They claim that about 80% of MULTICS, that is of measurement interest, is non-transient) (GROC690) the GIOC feeds a data block to 'GDM' which selects and reduces the data. This is actually a good technique, for it provides a maximum sampling rate of 20 per second, with at most a 0.1% degradation of the GIOC, and little effect on MULTICS.

Similarly, the hardware monitors 'ADAM' and 'SLUR' receive data from their object systems, but this data is in the form of programs to be executed by the monitor. This permits the user to write the experiment utilizing the object system's resources, then to transfer his program to the monitor, causing no interference with the object system while the experiment is in progress.

The monitor's effects upon the object system's internal operations have been discussed, but the object system also has personnel who are affected by a monitor's presence. In general, any of the main frame salesmen who are present could be made nervous when any measurement tools, with which

---

* GIOC is the General I/O Controller of the MULTICS System.

they are not accustomed, are used. However, we are more concerned about the effect on the system programmers, system engineers, operators and the users. System engineers do not worry about software monitors, but many have appeared nervous about the probes of hardware monitors. In a discussion with Honeywell engineers, working at the University of Waterloo, we discovered that they were strictly against any attachment of hardware probes to anything but the special test pins already provided, and these test pins only permit access to a few of the events listed in Chapter IV, meaning that special wiring would have to be done to make the missing events accessible at a test pin. However, T.E. Bell of Rand writes (BELL72G) that he sees no need for all the hysteria. He attached poorly designed probes to an IBM 1800 and only experienced minor hardware crashes, but never any permanent hardware damage.

Next the experimenter must consider the effect which monitoring may have on the object system's operators and programmers. The experimenter must realize that he is also dealing with humans, that the monitor may crash the object system, and that his conclusions may result in some suggestions to improve the efficiency of the work of the operators and programmers. The experimenter may have to apply some psychology when dealing with these people, but this is beyond the scope of this thesis.

6.8 DATA VALIDATION

Data validation implies for software monitors the verification of the software "hooks" and for hardware monitors the verification of the probe locations. Since software monitors are normally tools for one configuration and are modified very little, verification of the "hooks" need only occur once, until a new version of the object system is released. However, many hardware monitors have fewer than 50 probes and some relocation of probes is required for many experiments. only when using those hardware monitors fortunate enough to have a probe for every point of interest, would only a single probe validation experiment be possible.

The 'HARDWARE MONITOR', 'CHANNEL ANALYZER', 'PROGRAM MONITOR', 'POEM', 'PEC' and the 'EXECUTION PLOTTER', the first documented hardware monitors, had hardwired interfaces to their object systems and thus were not very adaptable and normally needed very little data validation. When 'SAMI', the first monitor to use probes, was built, new problems arose; such as the locating of probe points for each experiment or the electrical compatibility between a probe and any probe point.

A hardware monitor presently in the implementation stage at the University of Guelph, Ontario, Canada, has 120 probes for connection to an IBM 370/155. This number was chosen to minimize any need for probe relocation. (*) The 'TESDATA' models 1185 and 1200 provide up to 144 probes.

However, since most computer centres can only afford to rent hardware monitors, the number of set-ups generates a need for a standard probe validation experiment. the use of a real-time display would greatly facilitate this procedure, and also be useful for refreshing confidence by giving periodic data presentation during the long experiments. At any rate, there does exist the need to be able to read and analyze input data on a real-time basis, in order to conduct an easy, short experiment for probe validation.

6.9 DATA COLLECTION

The techniques used by monitors for data gathering can be discussed under the five subheadings of:

A. Resolution;
B. Input Sensitivity;
C. Accumulators, Comparators and Logic;
D. Continuity of Data Collection, and
E. Type of Data Collected.

(*) (Private communication from A.Dyer, Research and Development, University of Guelph)

6.9.1 RESOLUTION

As computer hardware becomes faster, the required resolution of a monitor system is being forced higher. The 'PROGRAM MONITOR' needed only a resolution of one millisecond while modern monitors range from 30ns. ('CMSM') to 171 ns. ('DYNAPROBE' 7700). This problem is mainly relevant to monitors using hardware, since the aim of software monitors is to provide a gross overview of information on the overall system activity for long periods of time and software monitors do not perform precise timing measurements. Nevertheless, the resolution of software monitors can be measured. The factors which effect the resolution are the speed of the object system, the resolution of the object system's clock, the sampling rate, and the processor time consumed by the monitor system.

Thus, as the object systems operate at higher speeds, the monitoring problem increases. At present, it appears that an overall resolution of 20ns is necessary to just register events from a high speed computer, but a monitor must normally perform several actions for each event; implying that the monitor must run fast enough to finish its reactions, before the next event occurs. This timing problem limits the number of actions which a monitor can perform, and for the events of very high frequency (over 20 Mhz.) and short duration (under 30 ns.), the monitor is restricted to using indirect techniques, such as saving the required action, perhaps data analysis or a cry for help, until later, when the monitor is not so busy.

The above discussion was concerned with the speed of an entire monitor system. In Sections 6.9.2 and 6.9.3 we will discuss the speed of some of the parts of a monitor, such as the probes or the accumulators and comparators.

6.9.2 INPUT SENSITIVITY

Input sensitivity is related only to those hardware monitors using probes. With all their monitors, Tesdata markets probes having a sensitivity of twenty nanoseconds, and 'DYNAPRØBE's' probes have a thirty nanosecond sensitivity. These sensitivities are necessary for the monitoring of today's fastest computers.

With the probes designed to sense the data at such fast rates, the monitors are required to process the data almost as fast; thus input buffering of the probes is used for 'SLUR' (one register), the 'UNIVAC INSTRUMENTATION' (four registers) and 'SNUPER' (specially designed buffer). This permits the monitors to have some time lag before data is lost, but timing is still a major problem with monitors. It is difficult, if not impossible, for a monitor to detect and take action on all events occuring within the object system. Theorem--In order to take action on all events, the monitor must be 'n' times as fast as the object system. What about monitoring the monitor system?

6.9.3 ACCUMULATORS, COMPARATORS AND LOGIC

Again, this category applies to only hardware monitors. (The characteristics involving accumulators and comparators for software monitors are dependent on the object system. The logic for software monitors is restricted to only software.) Accumulators are used for both timing and counting of events. The description of a monitor's accumulators involves the magnitude, the resolution and the number of counters available. The specifics of these three items are dependent on the individual monitor, but to be able to perform a worthwhile experiment on any computer system, a monitor would require at least four 16 bit accumulators with 50ns resolution. Similarly, a few comparators of 36 bit magnitude would be a requirement. Comparators are a vital entity for computers designed like 'SNUPER', which uses comparators for data selection before it accesses its 8K words of accumulators.

Actually most monitors use logic units and patchboards for data selectivity and data reduction. The facility of the set-up of the comparators and logic unit is an important area. Some systems utilize manual plugboards, and thus are not as flexible as those having firmware or hardware logic units or software programmable logic. 'SLUR' accomplishes data transferring, timing and accounting, all on a programmable level. 'ADAM' can be rearranged for a new experiment with only a 0.02% degradation of the object system and is therefore very flexible. Of course, the cost of the gain in flexibility is an increase in price.

6.9.4 CONTINUITY OF DATA COLLECTION

There are two modes of operation which describe the collection of data with respect to the object system's operation: serial and parallel. Serial involves stopping the object system, gathering the data, starting the object system and recycling through the procedure, whereas parallel monitors gather data without interruption of the object system.

The interleaving of data collection and object system operation is a trait common to all software monitors since they are forced to be either sampling or event driven monitors. The advantage is that the software monitor can complete the data gathering before the object system changes state. This is also possible with those hardware monitors which can interrupt the object system and cause a pause in the program execution while the interrupt is serviced--the servicing of the interrupt is assumed to be uninteresting data.

The action of software monitors interleaving with the programs which they measure degrades the system. The sampling rate must be wisely chosen, for the degradation due to varying the sampling rate has been observed to vary from 5% to 45%. This rate is dependent on timer interrupts though and if no clock is available, then the dependency is with other hardware interrupts and this certainly does not yield any continuity to the results. Thus, software monitor data collection can be solely dependent on events occurring within the object system, whereas hardware monitors can collect data continuously or use the object system's events as flags to stimulate a variety of actions such as record data, send a malfunction error to the operator, change some part of the experiment set-up, or to stop or start monitoring. The parallel collection capabilities of hardware monitors also permits the recording of simultaneous events, which is impossible using software techniques. A well-designed hardware-software monitor system would be able to exploit the advantages of both serial and parallel data collection, providing complete, continuous data, when desired.

6.9.5 TYPE OF DATA COLLECTED

The monitor events were discussed previously as a group which should be accessible to the monitor system, but software monitors and hardware monitors when working independently of each other, can only obtain subsets of the events described in Chapter IV. The differences in the two sets of measurement data become obvious when specific items like hardware activity and data structures are monitored.

Hardware activity is easily detected by hardware monitors, but software monitors can only use the object system's instruction set and thus are limited to the data which is stored in either program-addressable device registers or system software. The software monitor's advantages are that it can easily access system data bases, follow programs in core and read queue lengths, whereas hardware monitors cannot normally directly access storage devices. (The object system would have to be designed to permit at least two processors to access storage, and then a processor of the monitor system could access the data.)

Since both basic types of monitors can only obtain subsets of the required data, for complete experiments a hardware-software monitoring system is a necessary entity.

6.10 DATA REDUCTION

Data reduction is an important area of concern. The sooner data is reduced, the easier it is to handle, present and store, yet enough data must be kept available to ensure a certain degree of completeness for any later analysis. In the early stages, monitors collected data and stored it for further analysis. To minimize overhead, software monitors have a trade-off between the amount of data reduction done at collection time and the data storage required (both in terms of main memory required and the frequency of data transfers).

The 'PROGRAM MONITOR' is a good example of the lack of initial data reduction. This monitor detected software loops as its initial selection method and then recorded special loop information for each loop. The data was put on magnetic tape for later reduction. The speed of the object system--an IBM 7090--and the size of the magnetic tape restricted the experiment length to five and one-half minutes, but the ensuing software analysis used from two to seven and one-half hours of the IBM 7090 CPU time for reduction. This example should certainly strengthen the claim of a need for careful data reduction within the monitor!

6.11 DATA OUTPUT

The type of output is dependent on the purpose of the monitor. Various output devices have been used; meters, cards, printers, tapes, disks, drums, CRTs, and core. System operators have the choice of obtaining real-time feedback from monitors like the 'DOLBY' which displays device utility as a percentage on meters or the 'GDM' which has its own "Display Description Language", (DDL) to aid the operator in data selecting and display formatting on a CRT.

The special output needed for data validation during the initial stages of monitoring was discussed above, but another criterion for special output features will become evident as operating systems, which are capable of dynamic tuning, are written. This need is already acknowledged though, with respect to computer networks, for routing algorithms need periodic feedback on the network status in order to maintain uniform flow and minimize transmission delays.

6.12 CONCLUDING REMARKS

After studying the existing monitors it is obvious that the design is dependent on the final purpose of the monitor. The 'DOLBY', IBM 'PROGRAM MONITOR' and 'EXECUTION PLOTTER' (The 'EXECUTION PLOTTER' displayed instruction address versus time on a CRT) are three examples of simple purpose hardware monitors, whereas the discussions in references (BURK732, FULL732, BORD714, ROEC690, GROC690 and SALT69A) cover general utility monitors designed for unique computers. A summary of the above characteristics and limitations is contained in Table #6.1.

The following design guidelines are provided as a basis and are not meant to be applied to the design of all monitor systems. The design guidelines obtainable from the above discussion, presented in no specific order are to:

1. Minimize the cost without compromising the other design goals;
2. Provide an adequate level of intelligence and programmable control to relieve the user of any unnecessary tasks;
3. Be adaptable to as many computer systems and networks as practical;
4. Require minimum modification when not compatible with an object system;
5. Be easy to use;

6. Interfere minimally with the object system, considering both performance and integrity;

7. Require minimal modification to the object system, if any at all;

8. Permit feedback to the object system when required;

9. Educate systemS personnel and users in the benefits provided by and in the security of their work from the presence of the monitor;

10. Be reliable, and

11. Provide

    a. Adequate feedback to systemS personnel and users;

    b. A sufficient means of data validation;

    c. Various degrees of accuracy, such that the unit of measurement fits what is measured;

    d. Adequate data selectivity;

    e. Enough of data continuity;

    f. Flexibility in the data reduction and anlysis, and

    g. An appropriate set of output devices, permitting the display of legible, interpretable results.

CHARACTERISTICS AND LIMITATIONS OF MONITORS TABLE #6.1

HARACTERISTIC	SOFTWARE MONITORS			HARDWARE MONITORS		HARDWARE-SOFTWARE MONITORS
	ACCOUNTING	PERFORMANCE	MALFUNCTION	NOT INTELLIGENT	INTELLIGENT	
COST	$2-5,000	$3-25,000	?	$5-40,000	>$40,000	?
CONTROL TYPE	EVENT DRIVEN OR PART OF THE EVENT.			EVENT DRIVEN OR CONTINUOUSLY OBSERVING		
EXTERNAL	NONE	NONE	NONE	MANUAL	MANUAL	MANUAL
INTERNAL	SOME	SOME	SOME	NONE	PROCESSOR	PROCESSOR
FROM OBJECT SYSTEM	HARDWARE INTERRUPT OR SOFTWARE CALL.			OPTIONAL	OPTIONAL	OPTIONAL
INTELLIGENCE	OBJECT SYSTEM PROCESSOR.			NONE	MINIPROCESSOR	MINIPROCESS
EASE OF MODIFICATION	DEPENDS ON THE PROGRAM LOGIC AND DOCUMENTATION AND THE PROGRAMMER'S UNDERSTANDING OF IT.			VARIES FROM MONITOR TO MONITOR.		<-----
ADAPTABILITY	NORMALLY ARE DESIGNED FOR A SINGLE SYSTEM, BUT MAY BE MODIFIABLE FOR ANOTHER SYSTEM.			ALTHOUGH SOME MONITORS HAVE BEEN DESIGNED FOR SPECIFIC SYSTEMS, MOST CAN BE ATTACHED TO ANY OF A NUMBER OF SYSTEMS, BY SELECTING APPROPRIATE PROBE POINTS.		<-----
DATA SELECTIVITY	FIXED. REQUIRES SOFTWARE MODIFICATION. - - C O M P L I C A T E D - -	FIXED OR MODULAR.	FIXED.	MOVE A PROBE OR CHANGE A LOGIC BOARD.	EASY TO DIFFICULT, DEPENDI ON WHETHER PROBE MOVEMENT PROGRAMMABLE LOGIC ARE ENVOLVED. SOFTWARE COMMENT APPLY.	

ARACTERISTIC	SOFTWARE MONITORS			HARDWARE MONITORS		HARDWARE-SOFTWARE MONITORS
	ACCOUNTING	PERFORMANCE	MALFUNCTION	NOT INTELLIGENT	INTELLIGENT	
ACILITY OF USE OBJECT SYSTEM INTERFERENCE	NORMALLY ONLY LOADED ONCE.			PROBE SET-UP MAYBE ONLY ONCE, OR FOR EVERY EXPERIMENT.		<-----
EXPERIMENT SET-UP AND START.	CONTINUOUSLY OPERATING.	CONTROLLED BY JCL STATEMENTS.	CONTINUOUSLY OPERATING.	SIMPLE TURN-ON SWITCH.	WRITE A PROGRAM TO CONTROL INITIATE AND STOP THE EXPERIMENT.	
LEARNING TIME BEFORE USING MONITOR.	TRIVIAL	TRIVIAL	TRIVIAL TO DIFFICULT	PROBE POINT KNOWLEDGE IS NEEDED. REQUIRES COURSES AND AT LEAST ONE FAMILIAR PERSON ON SITE		
BJECT SYSTEM INTERACTION PERIPHERALS	YES	YES	YES	NOT NECESSARY, BUT MAY HAVE.	NO	EXPERIMENT DEPENDENT. MAYBE
PROCESSORS	YES	YES	YES	NO	NO	MAYBE
TIMER	YES	YES	YES	NO	NO	NO
MAIN MEMORY	YES	YES	YES	NO	NO	MAYBE
INTERFERENCE	YES	YES	YES	NO	OPTIONAL COMMUNICATION WITH AN OBJECT SYSTEM.	
EFFECT ON SYSTEM PERSONNEL PROGRAMMERS OPERATORS	MUST BE ACCUSTOMED TO HAVING THEIR WORK MONITORED AND RECEIVING-FEEDBACK FROM THE MONITOR.					
ENGINEERS	UNAFFECTED	UNAFFECTED	INDICATES NEED FOR MAINTENANCE	HARDWARE INTERFACES AND PROBES WORRY THEM.		

ARACTERISTIC	SOFTWARE MONITORS			HARDWARE MONITORS		HARDWARE-SOFTWARE MONITORS
	ACCOUNTING	PERFORMANCE	MALFUNCTION	NOT INTELLIGENT	INTELLIGENT	
FFECT ON YSTEM PERSONNEL SALESMEN	ANY MEASUREMENT DEVICE WORRIES THEM!					
ATA VALIDATION	DONE ONLY AT DEBUG TIME.			DONE ONLY WHEN THE PROBES ARE REALLOCATED, THUS NONE IS DONE FOR HARDWIRED INTERFACES.		<-----
ESOLUTION	ONLY AS ACCURATE AS THE OBJECT SYSTEM'S CLOCK. EXPERIMENTERS MUST CONSIDER THE TIME LAGS WHICH MAY OCCUR BETWEEN THE EVENT AND THE ACTION TO BE TAKEN.			VARIES WITH RESPECT TO THE MONITOR. RANGE HAS PROGRESSED FROM 1ms. in 1961 TO 30ns.		<-----
ROBE SENSITIVITY	N/A	N/A	N/A	AT PRESENT, AS FAST AS 20 NONOSECONDS		
NPUT SENSITIVITY	N/A	NØA	N/A	THE SPEED OF DATA COLLECTION AND THE MONITOR'S RATE OF HANDLING IS A CRITICAL BALANCE AND CAN RESULT IN INCORRECT OR LOST INFORMATION.		
ACCUMULATORS & COMPARATORS	USE'S OBJECT SYSTEM'S ARITHMETIC UNIT.			SHOULD HAVE AT LEAST FOUR, SIXTEEN BIT ACCUMULATORS WITH 50ns. RESOLUTION AND TWC THIRTY-SIX BIT COMPARATORS.		
OGIC	SOFTWARE	SOFTWARE	SOFTWARE	MANUALLY OPERATED PATCH-BOARDS.	PROGRAMMABLE LOGIC UNITS OR SOFTWARE.	PROGRAMMABLE LOGIC UNITS, OBJECT AND MONITOR SYST SOFTWARE.

CHARACTERISTIC	SOFTWARE MONITORS			HARDWARE MONITORS		HARDWARE- SOFTWARE MONITO
	ACCOUNTING	PERFORMANCE	MALFUNCTION	NOT INTELLIGENT	INTELLIGENT	
DATA COLLECTION						
TECHNIQUE	SERIAL	SERIAL	SERIAL	PARALLEL	PARALLEL	<-----
CONTINUITY	EVENT DRIVEN	SAMPLING OR EVENT DRIVEN	SAMPLING OR EVENT DRIVEN	CONTINUOUS OR EVENT DRIVEN	CONTINUOUS, SAMPLING OR EVENT DRIVEN	CONTINUOUS SAMPLING O EVENT DRI
SIMULTANEITY	NONE	NONE	NONE	YES	YES	YES
TYPE HARDWARE ACTIVITY	RESTRICTED TO READING DEVICE REGISTERS.			YES	YES	YES
DATA BASE	YES	YES	YES	NO	NO	YES
DATA REDUCTION	CRITICAL BALANCE BETWEEN DOING REDUCTION AT COLLECTION TIME AND LATER; SINCE IT EFFECTS STORAGE NEEDED AND DEGRADATION OF THE OBJECT SYSTEM.		NO MATHEMATICS IS NORMALLY PERFORMED.	CAN BE CAPABLE OF SUBSTANTIAL REDUCTION AS DATA IS RECEIVED, WITHOUT LOSING ANY INFORMATION.		<-----
OUTPUT	STORAGE PERIPHERALS	MANY DEVICES.	TO OPERATORS TERMINAL OR A MESSAGE TO A PROCESS WITHIN THE OBJECT SYSTEM.	METERS, STORAGE PERIPHERALS.		
					MESSAGES OR DISPLAYS TO TERMINALS OR PROCESSES WITHIN EITHER MONITOR OR OBJECT SYSTEM.	

network of monitors; whether software, hardware or both.

The characteristics and limits of these "new types" of monitors certainly are the same as those listed in Chapter VI, but new problems arise. Consider a hardware-software monitor at each node of a network. Each one of these monitors must be capable of having its activities controlled and coordinated by a computer external to the network. The advantages of centralization (economy, easier control, and coordination) imply there should exist the capability to control all the monitors from a single computer, but the topology of some networks, with some nodes separated by distances as large as thousands of miles, causes communications problems which do not occur when measuring a single computer system.

Thus, for a network monitor system, we have a few design guidelines to add to the list in Section 6.12:

A. The monitor must span the network.

B. The activities of each monitor at a node must be capable of being coordinated from a central control.

C. Often data for several nodes of a network must be analyzed as a whole in order to gain the required insight. In these cases, the measurement data must be transmitted to a central analysis centre. Thus the monitor system should be capable of transferring data between its own subsystems (logical and physical separation of the monitor's communications system from that of the network is preferred).

D. To achieve dependability, the monitor system could be designed such that there is more than one computer which can provide centralized control and analysis.

It is with the above considerations that a network of hardware-software monitors is being constructed. A description of this network monitor and some proposed experiments for its calibration and preliminary evaluation are presented below in Sections 7.2 and 7.3.

7.2 MONINET: A Network Activity Monitor

7.2.1 INTRODUCTION

A computer system for network activity monitoring--MONINET--has been designed, partially implemented, tested and developed in an attempt to meet our requirements for a hardware-software monitor system. Moninet is designed to be capable of performing both performance measurement and maintenan8e functions for a network of computers (as well as for a single computer system). We have tried to achieve many of the advantages listed in Table #6.1 as possible.

MONINET is being designed modularly and with as little inter-module dependence as is possible. The basic components of the system are:

A. A Remote Computer-controlled Hardware Monitor

(RCHM);

B. A control centre using a minicomputer and a software control system;

C. A set of software modules to perform special functions such as data reduction and analysis, or network traffic generation;

D. A Measurement Control Language, and

E. Measurement software in each computer monitored.

A network of computers typically consists of at least two computers linked together. The typical components of a network are:

A. The Host, a computer which performs as little of the network's communications control functions as possible;

B. The Node, a computer directly involved in the network's data switching control;

C. The Terminal, a communications interface between the network and man, and

D. The Transmission lines, the information carriers.

If we restrict the monitoring to the network, MONINET's configuration will appear as in Figure #7.1, with one RCHM per node and the control centre at the Network Measurement Centre (NMC). The measurement software (MS) within each node will be minimized and be under the control of its respective RCHM. The MS is, of course, object system dependent.

7.2.2 THE REMOTE COMPUTER-CONTROLLED HARDWARE MONITOR

The design of an RCHM is also modular and constructed such that the RCHM at each node can be of a different configuration from that of the RCHMs at other nodes. The only compulsory parts are the communications and control equipment: a MONIBUS, a MONIBUS-to-communications-line interface and controller, a MONIBUS-to-Unibus interface (We are currently using a PDP-11 as a control centre, since its bus structure easily permits the desired modular construction), an interrupt generator and some programmable switch matrices. The remainder of the RCHM will have one or more of each of the following specially-designed components:

A. Event Detectors

1. Masked-word range comparator;
2. Character Detector;
3. Combinational Logic Unit, and
4. Sequential Logic Unit.

B. Time Measuring Components

1. Time-Stamp Unit;

2. Time and Event Counter;

3. Network Clock(including a way of synchronizing to a standard time source, such as CHU or WWV), and

4. Interval Timer.

C. Data Recorders

a. Time and Event Counter;

b. Flip Flop Bank;

c. Event Memory;

d. Histogram Generator, and

e. Moment Generator (Yielding the first four moments of a distribution).

Figure #7.2 illustrates the interconnection and control of these components to form the RCHM and a description of each component following the order given above, is presented below.

7.2.2.1 COMMUNICATIONS AND CONTROL EQUIPMENT

The RCHM architecture is bus-oriented. The monitor bus (MONIBUS) permits the flow of both control information and data between monitor components. The MONIBUS is connected to the control computer's Unibus (since a PDP-11 is used at present) via a special interface or via a telecommunications line and the interface. This architecture permits the control comp-ter to address the RCHM components directly, but at present, the address restri-ts the number of RCHMs per control computer to four. Thus, to monitor an object system needing over four RCHMs, we introduce the Regional Network Measurement Centre (RNMC) and Figure #7.1 is modified and displayed as Figure #7.3.

The Interrupt Generator

has several input lines to permit both fixed interrupts for control such as overflow or timing, and selectable interrupts for the experimenter's use. Selective generation of interrupts is particularly useful for malfunction monitoring.

The Switch Matrix

permits connections (input to output) one to one, and one to many, but not many to one. The switch matrix receives input from the probes connected to the object system and from most of the RCHM's components. The switch matrix is controlled from the controlling computer via commands sent on the monibus.

7.2.2.2 EVENT DETECTORS

The Masked Word Range Comparator

is designed to test if a bit

string, regarded as a binary value, falls within two limits. Each comparator tests strings of length sixteen, but four can be combined to test a 64-bit string. There are five output lines to indicate whether the bit string is above, below or within the range, or if it is on the upper or lower boundary of the range. Four of these comparators can be combined to provide a set of ranges for some purposes, such as the histogram generator. The comparators are also software controlled.

The Character Detector Unit

receives eight bit characters in a serial pattern and tests to see if the input is one of sixteen desired patterns. Each detector will have sixteen subunits which receive a parallel pattern, mask it and compare it. There will be thirty-two outputs, one for the result of each comparison and its complement.

The Combinational Logic Unit

receives eight input lines as selected using the switch matrix. The unit is designed to test if the input represents a specific event as defined in Chapter IV. This event is defined by a Boolean function given to the controlling computer, which reduces the functional representation to a Karnaugh Map and feeds the map to the Combinational Logic Unit for use.

The Sequential Logic Unit

determines if a sequence of events represented on its input lines is following a specific pattern. The pattern is defined by a regular expression which was given to the controlling computer, manipulated and passed to a sequential unit. The current design has eight inputs, eight outputs and thirty-two states, but, of course, the units could be built of nearly any size.

7.2.2.3 TIME MEASURING COMPONENTS

The Time Stamp Unit

receives an event stimulus, which causes it to record an identifier for the stimulus, the time of day and 16 selected indicators of the state of the object system when the event occurred. Currently, up to four different events can be permitted as stimuli and the resolution for the time is in multiples of 200 nanoseconds.

The Time and Event Counter

receives a single input, counts the number of times it goes high and how long it stays high. The magnitude of each register is 32 bits. The timing resolution is 0.1 us. and the maximum count rate is 10 Mhz.

The Network Clock

will be used to synchronize the monitor experiments from RCHM to RCHM. This clock will supply the time of day. The synchronization currently possible is within 100ns.

The Interval Timer

is a 100 nanosecond pulse generator. It generates one pulse every 'n' x 100 nanoseconds, where 'n' is specified by the controlling computer. It is used to indicate to the monitor when to sample the system, should sampling rather than event driven monitoring be desirable for a particular application.

7.2.2.4 DATA RECORDERS

The Flip Flop Bank

consists of a set of individual flip flops, each having two inputs, a set and a reset, and two outputs, the current state and its complement.

The Event Memory

is the storage provided to record data within each RCHM until the controlling computer can receive it.

The Histogram Generator

consists of three parts: the mask, the comparison and the counting. The input data, currently 16-bits wide, is masked, compared with several interval values, and then the appropriate counter (determined by the interval that corresponded with the input) is incremented. The mask and the interval boundaries are supplied by the control computer. A control line has been included to define when the histogram generator should accept data. The resulting histogram can have arbitrary scales on the absicissa and ordinate.

This concludes the discussion on the RCHM. The discussion is not as detailed as it could be, since only a limited number of the components are constructed and even though the designs are complete, the model being constructed is a prototype, restricted to an overall resolution of 100 nanosecond, and some revisions are being made as the implementation occurs. For more detail, see "PERFORMANCE MEASUREMENT IN COMPUTER NETWORKS", D.E.Morgan, University of Waterloo, Computer Communications Network Group Annual Report, October 1973.

7.2.3 SOFTWARE CONTROL

Programs have been writen to set-up, test the set-up, control, and exercise the constructed components of an RCHM. A complete description of the software is expected to be published by W.Colvin as a Master's Thesis from the University of Waterloo, in 1974.

In parallel with the writing of the individual control programs for the RCHM components, the Fortran compiler on the PDP-11 has been given more features. Subroutines and primitives have been added to permit the writing of control programs which permit either the same experiment to be conducted on several RCHMs or a different experiment to be conducted simultaneously on each RCHM, and a prescanner was added to permit the placing of regular and logical expressions and special assignment statements within the user's program.

7.2.4 LOAD GENERATOR

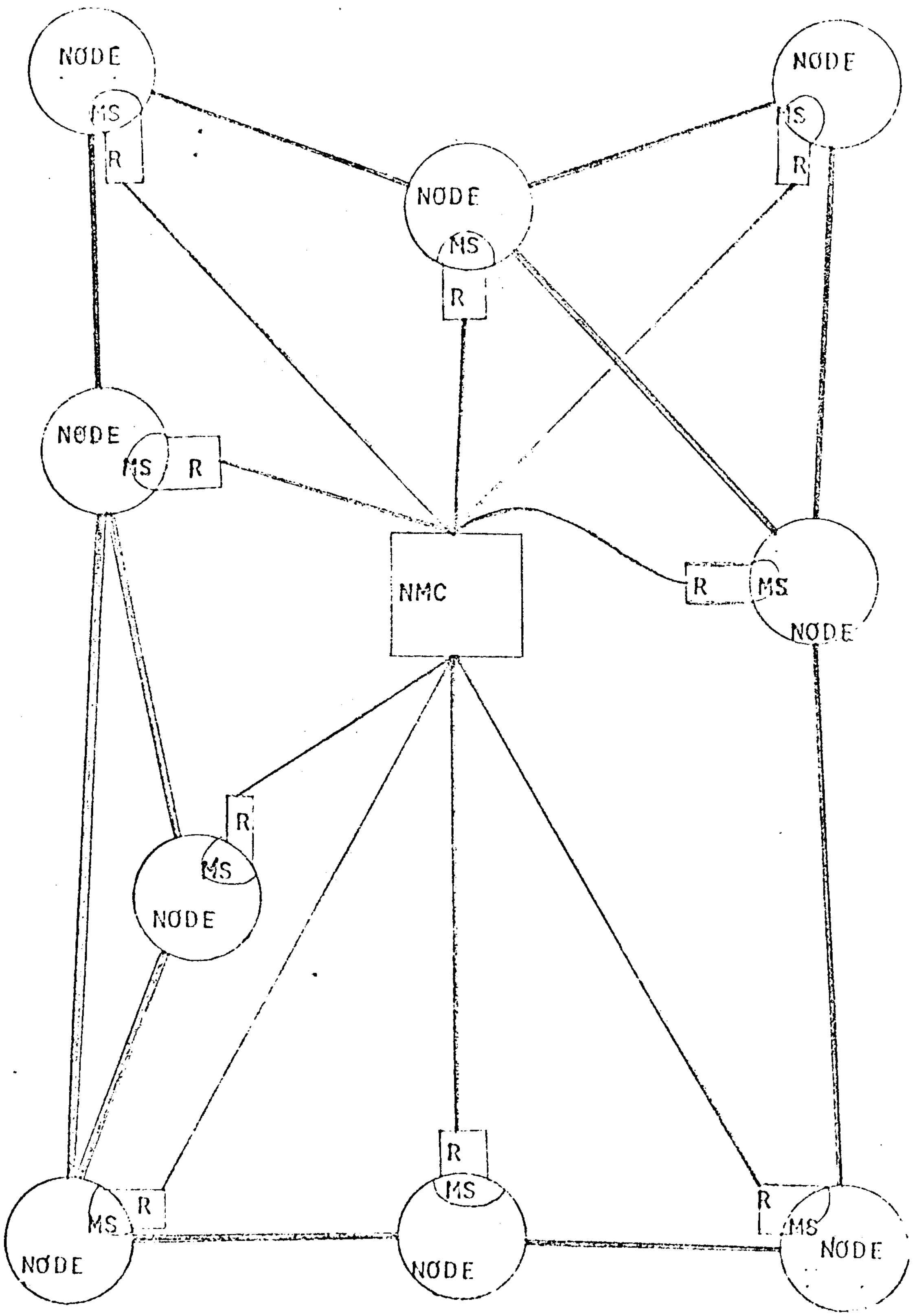
A load generator has been written which accesses a given data base containing typical messages and sends these messages through the network. It simulates the action of sixteen terminals, is implemented and is working, but more features are being added. The intention is to build a library of load generator programs which are as object system independent and as general as possible.

7.2.5 DATA REDUCTION AND MANIPULATION

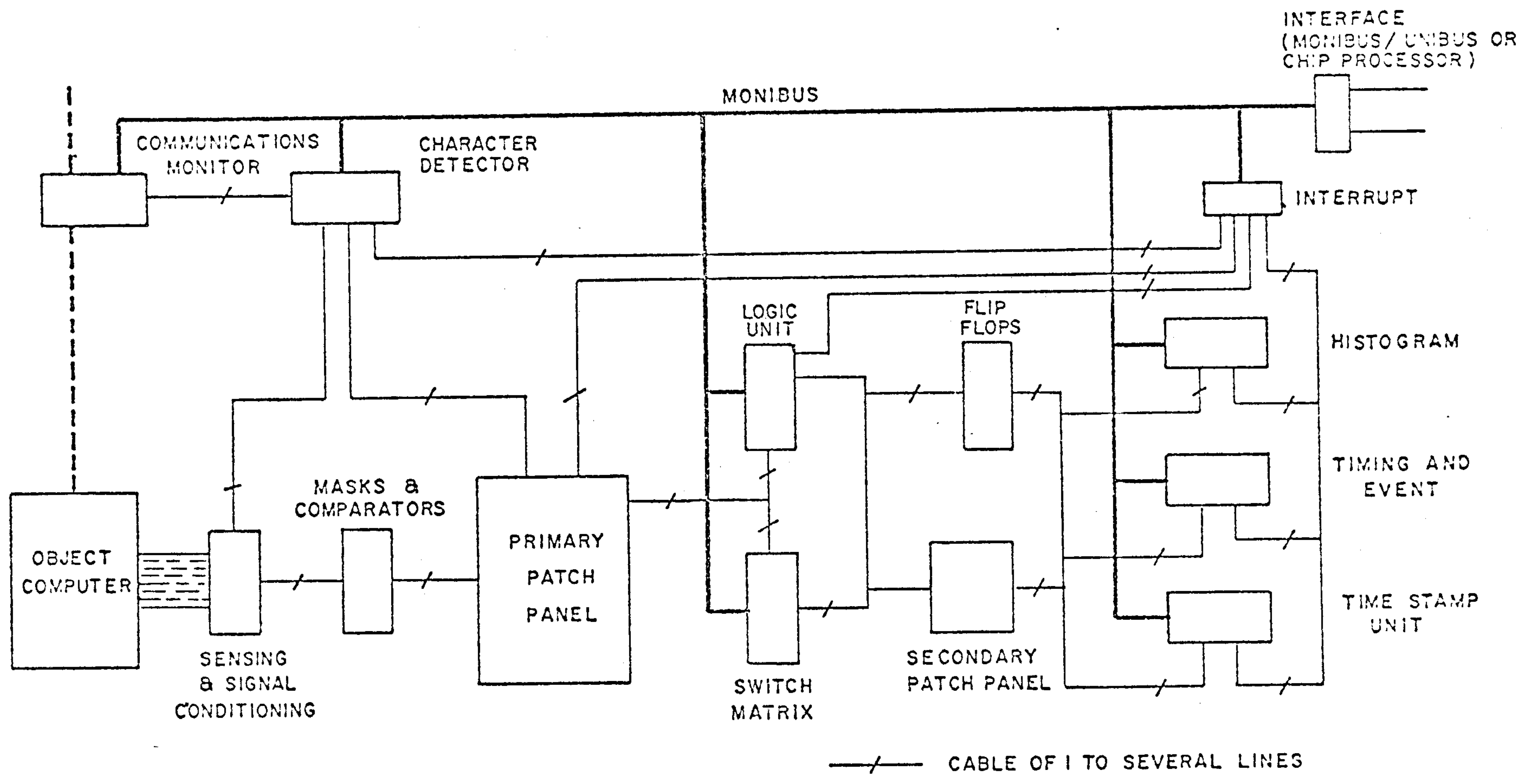
Software is required to provide the data reduction and manipulation. The functions to be performed will be the reduction of any data from the software monitor sections, the time stamp and the time and event counters, and the manipulation of all data into legible formats. At present, the experimenter has the use of a teletype, a printer, and a CRT, but no data manipulation routines have been written. The aims are to provide standard programs which will perform both reduction and manipulation according to the parameters specified by the experimenter(as well as having default routines if the parameters are not specified.)

FIGURE 17.1

MONITORING A COMPUTER NETWORK



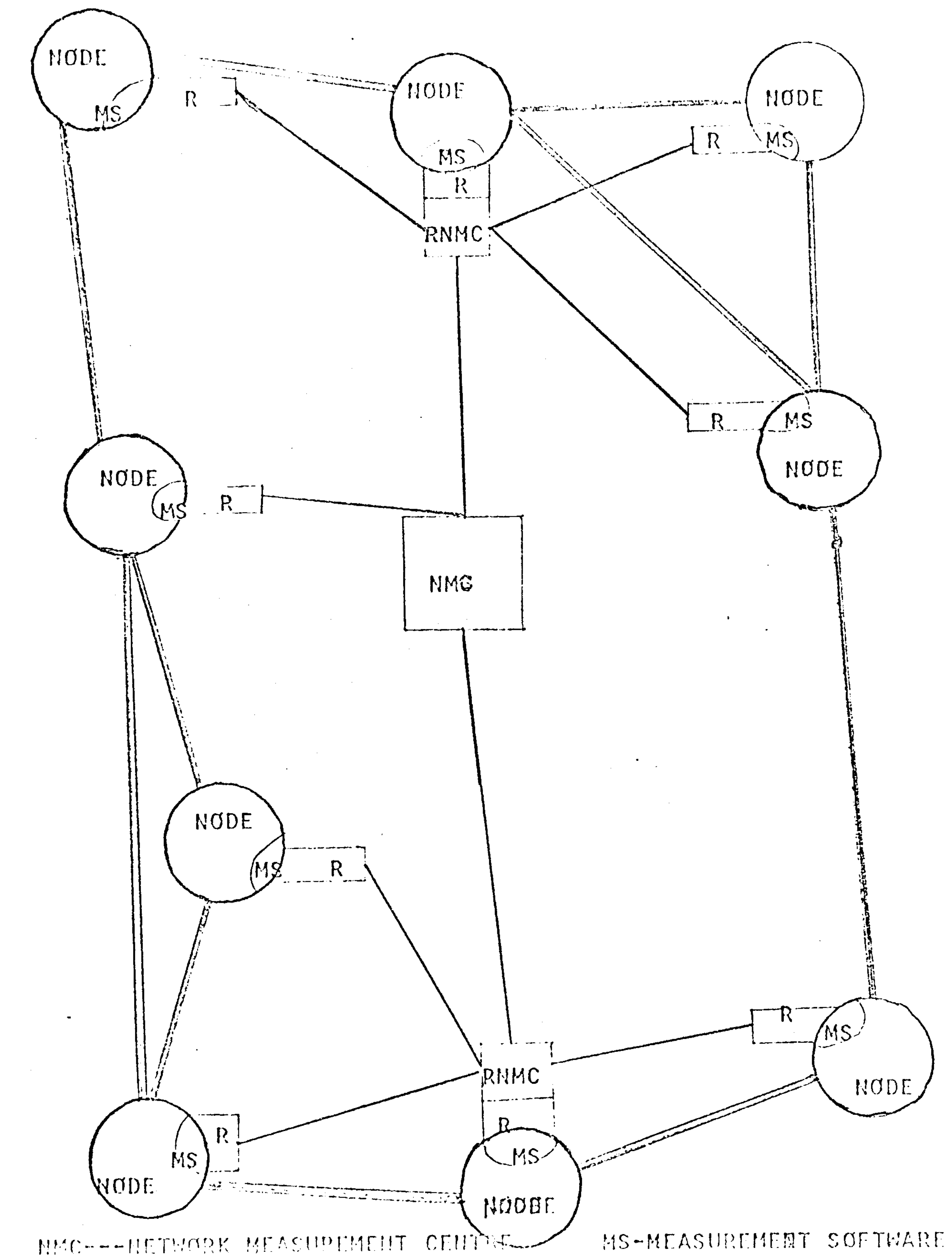
NMC___NETWORK MEASUREMENT CENTRE R___RCHM
MS___MEASUREMENT SOFTWARE



Remote-Computer-controlled Hardware Monitor (RCHM)
Figure 7.2

FIGURE 7.2

ACTUAL CONFIGURATION OF MONINET



NMC---NETWORK MEASUREMENT CENTRE   MS-MEASUREMENT SOFTWARE
RNMC--REGIONAL NMC
R-----RCHM

FIGURE #7.4

EXPERIMENT CONFIGURATION--SECTION 7.3.3.3., PARAGRAPH A.10

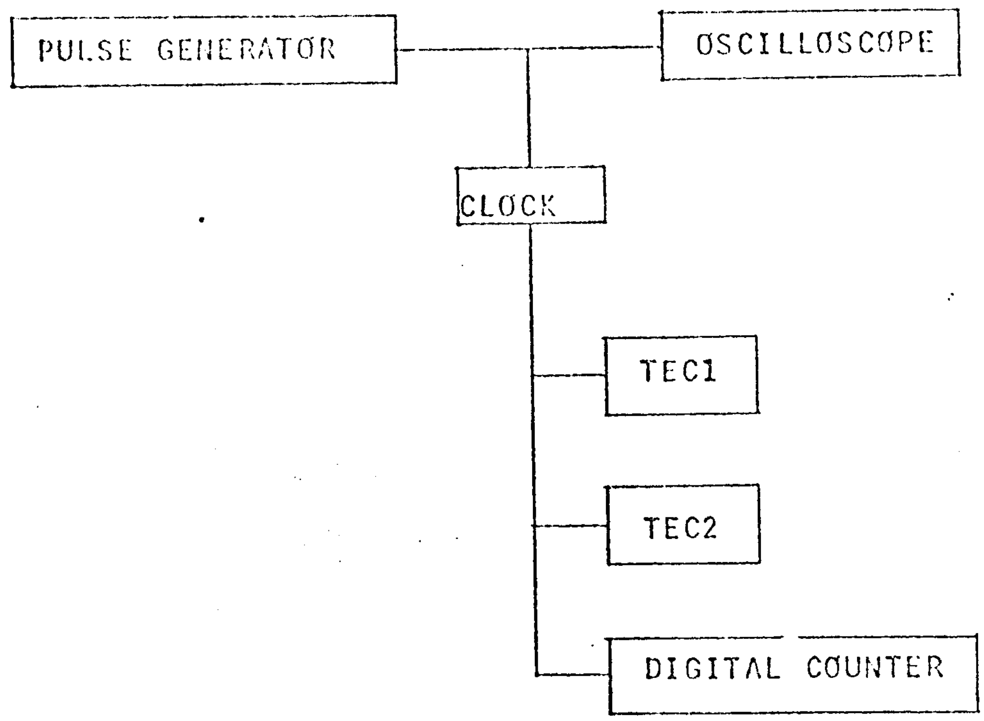
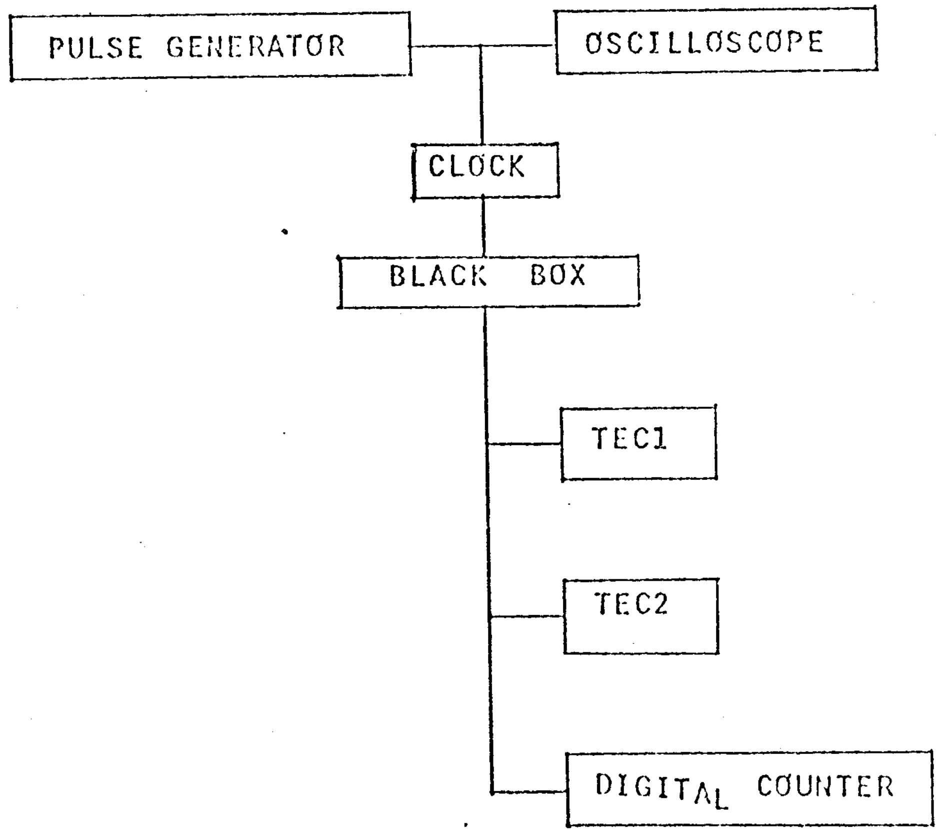


FIGURE #7.5

EXPERIMENT CONFIGURATION--SECTION 7.3.3.3, PARAGRAPH A.1



7.3 EVALUATION AND CALIBRATION OF MONINET

7.3.1 INTRODUCTION

MONINET is to be used as a measurement tool and thus requires evaluation and calibration. To accomplish calibration, the accuracy of MONINET must be tested and thus MONINET must be driven with controlled stimuli, which have dimensions with absolute uncertainties (*) at least as small as those desired for MONINET. These stimuli must traverse each component of MONINET, to discover how each component affects the accuracy of the data recording components. This will involve the repetition of many standard experiments.

For evaluation, MONINET should be applied to functional equipment, a computer system which is executing programs of known characteristics. This standardization is expected to be less reliable, since the times quoted for instruction speeds by many manufacturers have poor accuracy. The object system to be used for these initial experiments is a PDP-11/20 and the quoted times are accurate only to within 20%.

(*) The uncertainty of a measured value is the outer limits of confidence, within which we are "almost certain" (perhaps 99% certain) that the measurement lies. There are two general methods of expressing an uncertainty. The first (absolute uncertainty) is expressed with the uncertainty having the same units as the measured value, whereas the second technique (Relative uncertainty) expresses the uncertainty as a fraction or percentage of the measured value.

Thus, the purpose of this chapter is to present a technique to calibrate and evaluate the current version of the monitor.

7.3.2 EQUIPMENT

Only some of the equipment available at this writing will be discussed, since the concepts can easily be expanded to include other monitor components as they are constructed and attached. Thus the equipment discussed is a patch panel (PP), a switch matrix (SM), two combinational logic units (CLU1,CLU2), two flip flops (FF1,FF2) and two time and event counter units (TEC1,TEC2).

7.3.3 CALIBRATION

7.3.3.1 PURPOSE

A. To discover the accuracy of the two time and event counters.

B. To test if any of the existing components affect the counters.

7.3.3.2 APPARATUS

A. TEC1 and TEC2.

B. Patch Panel

C. Pulse generator with variable pulse duration and a variable frequency.

D. A digital counter accurate to 0.01us. (A power of ten more accurate than the counters of MONINET) and a magnitude of at least ten decimal digits, since MONINET's counters have 32 binary digits.

E. A timer which will permit a signal to pass for preset lengths of time (We require a 2% accuracy).

F. One oscilloscope capable of displaying pulses which range from 1.0ns to one second in duration.

7.3.3.3 METHOD

A. 1. Set up equipment as shown in Figure #7.4.

2. Discover interference from noise.

a. Without generating any pulses from the pulse generator, put the digital counter, TEC1, and TEC2 in their read mode.

b. Set timer for six hours and start timer (assuming that starting the timer permits signals to flow on the input lines to the digital counter and each TEC.

c. When the timer stops, record the values on the digital counter and each TEC. These values will be the noise level.

3. a. Set the pulse generator to a frequency of 100 Khz.

b. Set pulse size to 5.0us. and check with the oscilloscope.

c. Set timer for 100 seconds (*) and start the experiment.

d. When the timer stops, read the values in the three counters.

4. Repeat step #3 with a 1.0us. pulse size and an experiment time of 500 seconds.

5. Repeat step #3 with 0.1us. pulse size and an experiment time of 5,000 seconds (83.0 min.).

6. Repeat step #3 with a 1.0 Mh. pulse frequency, 0.9us pulse duration and 55.0 second experiment time.

7. Repeat step #6 with a 0.2us. pulse and 250 second experiment time.

7.3. Repeat step #6 with a 0.1us. pulse and 500 second experiment time.

9. Repeat step #3 with a 10 Mhz. pulse frequency and experiment length of 50 seconds. The pulse duration must be under 0.1us. Since MONINET

(*) Times are chosen to obtain a maximum count on each TEC, without causing overflow of the counters. See Calculations for overflow times.

samples at this rate, the technique is to discover how small a pulse MONINET will detect(expected to be between 10 and 20 ns.)

10. Set equipment up as in Figure #7.5.

11. Repeat steps 3 to 9 for this experiment set-up, using for the black box the switch matrix, then repeat for each CLU as the black box, and then repeat for the black box being the CLU1 into the switch matrix and then for CLU2 into the switch matrix. (*)

7.3.3.4 CALCULATIONS

1. The register size of each TEC is 32 bits, implying that a maximum count of 530,092,696 is available(we use 530*10**6, assuming that three decimal digit accuracy is available throughout the experiment).

2. MONINET's sample rate is 10 Mhz with a relative uncertainty of 0.05%.

3. If counting continuously, the time to overflow the counters (TO) is (5.30*10**8) / (10**7) which is 53.0 seconds.

4. For the experiments, the experiment duration is calculated as the Pulse Size times the Pulse Frequency

(*) For testing each CLUn various boolean functions should be tested and all eight input lines should be individually tested at the critical level of 10 Mhz. For example, use th e function A+B+C+D+E+F+G+H and repeat #9 for the clock pulse on each of the eight input lines, or use the function A.B.C.D.E.F.G.H with seven high lines and one input line from the experiment clock.

(PS*PF).

5. Therefore, the maximum experiment time is TØ/(PS*PF). See Table #7.1.

For insurance and simplicity the experiment times were chosen slightly below the maximum experiment time. These times, and the expected readings are also given in Table #7.1.

TABLE #7.1

EXPERIMENTAL TIMINGS AND APPROXIMATIONS

PULSE FREQUENCY Mhz	PULSE SIZE us.	PS*F	MAXIMUM TIME seconds	EXPERIMENT DURATION seconds	EXPECTED COUNT (10**6)
0.100	5.00	0.500	106	100	500
	1.00	1.00	106	100	500
	0.100	0.0100	5,300	5,000	500
1.00	0.900	0.900	58	55.0	495
	0.20	0.20	265	250	500
	0.10	0.10	530	500	500
10.0	<0.100	<1.00	53.0	50.0	500

7.3.3.5 ERROR ANALYSIS

The relative uncertainty for the expected count is a sum of the relative uncertainties for the setting of the experiment time(reading the clock), the pulse frequency, the pulse duration, and the sample time. This is because the expected time is calculated from the following equation:

EC = (ET * PF * PS) / SR

The relative uncertainty for SR is 0.05%, so that the critical uncertainties are only the other three.

7.3.4 EVALUATION

7.3.4.1 PURPOSE

To use MONINET in a few basic experiments in order to judge its effectiveness on a PDP-11/20.

7.3.4.2 APPARATUS

A. A patch panel, switch matrix, two CLUs, two TECs, and two flip flops.

B. A board containing signal conditioning components for eight probes which will be attached to a PDP-11/20, will contain the following probes: ANY_INSTR, EXEC, EMT, BINTR, RTI, BR, ANY_BR, VERIFIED_BR.

The meaning of each of these is:

ANY_INSTR -The signal is high when the '11/20 does a fetch cycle from ISR 0, which represents the fetching of an instruction for execution. The rising edge of the pulse is used to define the instruction start time.

EXEC -The signal is high for the execution of the operation specified for the instruction. The falling edge of the pulse determines the instruction stop time.

EMT -Indicates the occurrence of a trap to an emulating routine.

BINTR -indicates the occurrence of areturn from an interrupt or trap routine.

RTI -indicates the occurrence of a return from an interrupt or trap routine.

BR -indicates the occurrence of an unconditional branch instruction.

ANY_BR -indicates the occurrence of a branch instruction, whether branch occurs or not.

VERIFIED_BR -indicates the occurrence of a branch.

C. Digital Counter

7.3.4.3 METHOD

A. The equipment should be assembled as in Figure #7.6. The wiring on the patch panel should be of a distinct colour code, such that the patch panel may be used for supplementary connections for each sub-experiment without interfering with the original configuration.

B. We will write a general Fortran program which permits interactive control of the experiments and can contain subroutines to do the set-up for each of the experiment sections to follow. Software will have to be used to control the time of the experiment.

C. Validate the probes on the BR, ANY_BR, and VERIFIED_BR.

1. Connect BR to TEC1 and ANY_BR to TEC2.

2. Load a program of N "BR .+1" instructions into the object system.

3. Initialize and start MØNINET.

4. Let the object system run through the program.

5. Check that TEC1 and TEC2 are both equal to N and stop MØNINET.

6. Load a program consisting of all branch instructions (exluding the BR, there will be sixteen branch instruction.) with operands to branch to next instruction (so that whether or not the system branches or simply falls through, the next branch instruction will be executed. Let the last instruction be a HALT.

7. Initialize and start MØNINET.

8. Let the program on the object system execute.

9. Check that TEC1=0 and TEC2=16.

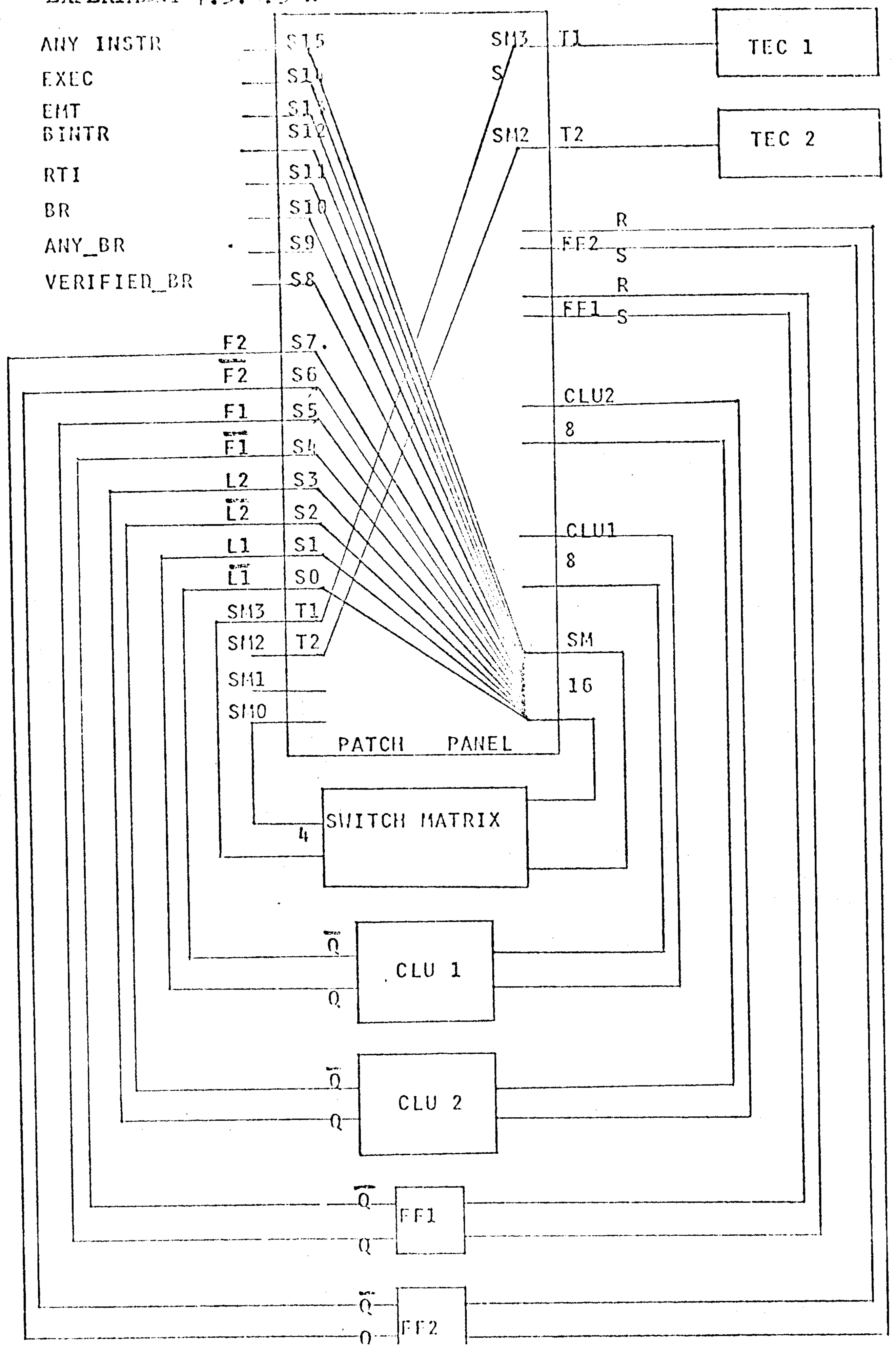
10. Connect the VERIFIED_BR to TEC1 and ANY_BR to TEC2.

11. Use the same program as is in the object system, but add an instruction to the start(namely a TST (Switch Register)).

12. Set the Switch Register on the object system

EXPERIMENT 7.3.4.3-A

FIGURE #7.6



to zero.

13. Initialize and start MONINET.

14. Execute the loop on the object system.

15. Check the TEC1=8 and TEC2=16.

D. Measure the average instruction time of a PDP-11/20 for three different instruction types.

1. Arrange equipment as in Figure #7.7.

2. Load program #1 of Table #7.2 into the object system.

3. Initialize and start MONINET.

4. Use the single instruction mode on the object system to execute one instruction at a time for one pass through the loop, and obtain the time for one loop's duration.

5. Clear and prepare the digital counter for reading.

6. Initialize and start MONINET.

7. Start loop in the object system and let it execute for approximately six minutes. (*)

8. Read the digital counter and TEC1&2.

9. Repeat steps #2 through to #8 for programs #2 and #3 of Table #7.2.

(*) Due to the software control of the duration of the experiment, it will be much harder to estimate values for each TEC.

TABLE #7.2

EXPERIMENTAL PROGRAMS

```
                #1                     #2               #3

          EMT   (TØ X)          JSR   R5,A        NØP
          BR    .-2             BR    .-4         NØP
        X TRAP  (TØ Y)          A JSR  R5,B       NØP
          RTI                   RTS               NØP
        Y IØT   (TØ Z)        B JSR   R5,C        NØP
          RTI                   RTS               NØP
   (**) Z ATRAP (TØ A)        C JSR   R5,D        NØP
          RTI                   RTS               NØP
        A RTI                 D RTS               NØP
                                                 NØP
                                                 BR   .-10
```

(**) This is a trap with no mnemonic and it will designated as a 'ATRAP'.

E. Measure the time within a routine which handles EMT traps.

1. Set up the equipment as in Figure #7.8.

2. Load a program of known duration into the object system and set the object system ready to execute an EMT to the program, but waiting in the Halt mode. Program returns after execution to a HALT instruction.

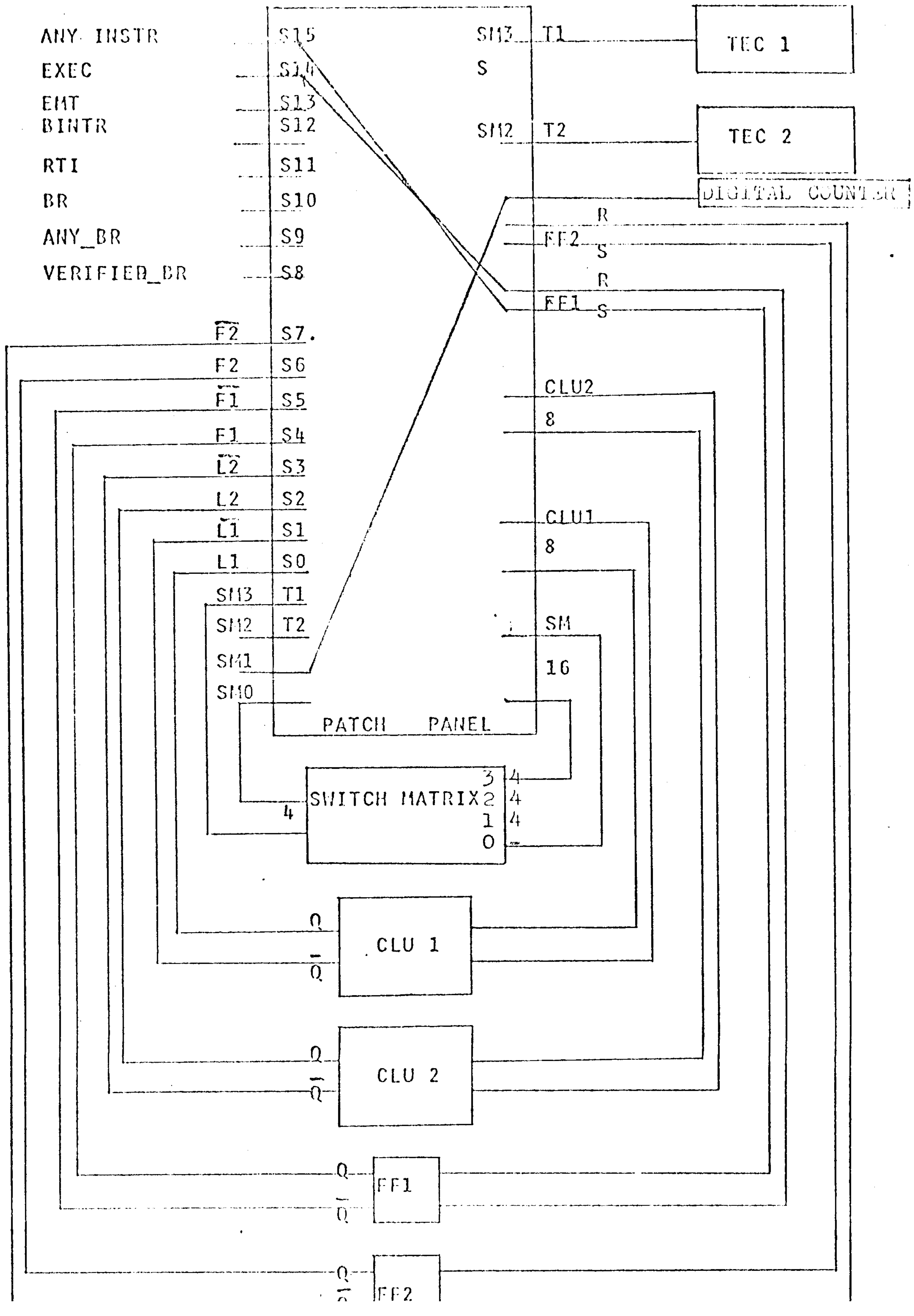
3. Activate MONINET to read.

4. Start the object system.

5. When the object system halts read each TEC and halt MONINET. Repeat once for verification.

EXPERIMENT 7.3.4.3-D

FIGURE #7.7



F. Measure the time within a routine which handles bus interrupts.

1. Arrange the equipment as in Figure #7.9.
2. Load a program of known duration into the object system and set the object system into a simple loop.
3. Initialize and start MØNINET.
4. Cause a bus interrupt to occur.
5. When the object system goes back into the loop, halt MØNINET and read each TEC.
6. Repeat once for verification.

7.3.4.4 CALCULATIONS

1. For Method C, see Table #7.3 for the instruction times and Table #7.4 for the estimated results based instruction times quoted with a 20% relative uncertainty. The experiment time is assumed to have 5% error for 360 seconds. The average instruction time is not estimated since it would have a relative uncertainty of 50%.

TABLE #7.3

SOME PDP-11/20 INSTRUCTION TIMES

INSTRUCTION	TIME	INSTRUCTION	TIME
EMT	9.3	BR	2.6
TRAP	9.3	JSR	4.4
IOT	9.3	RTS	3.5
ATRAP	9.3	NOP	1.5
RTI	4.8		

EXPERIMENT 7.3.4.3-F.1

FIGURE #7.9

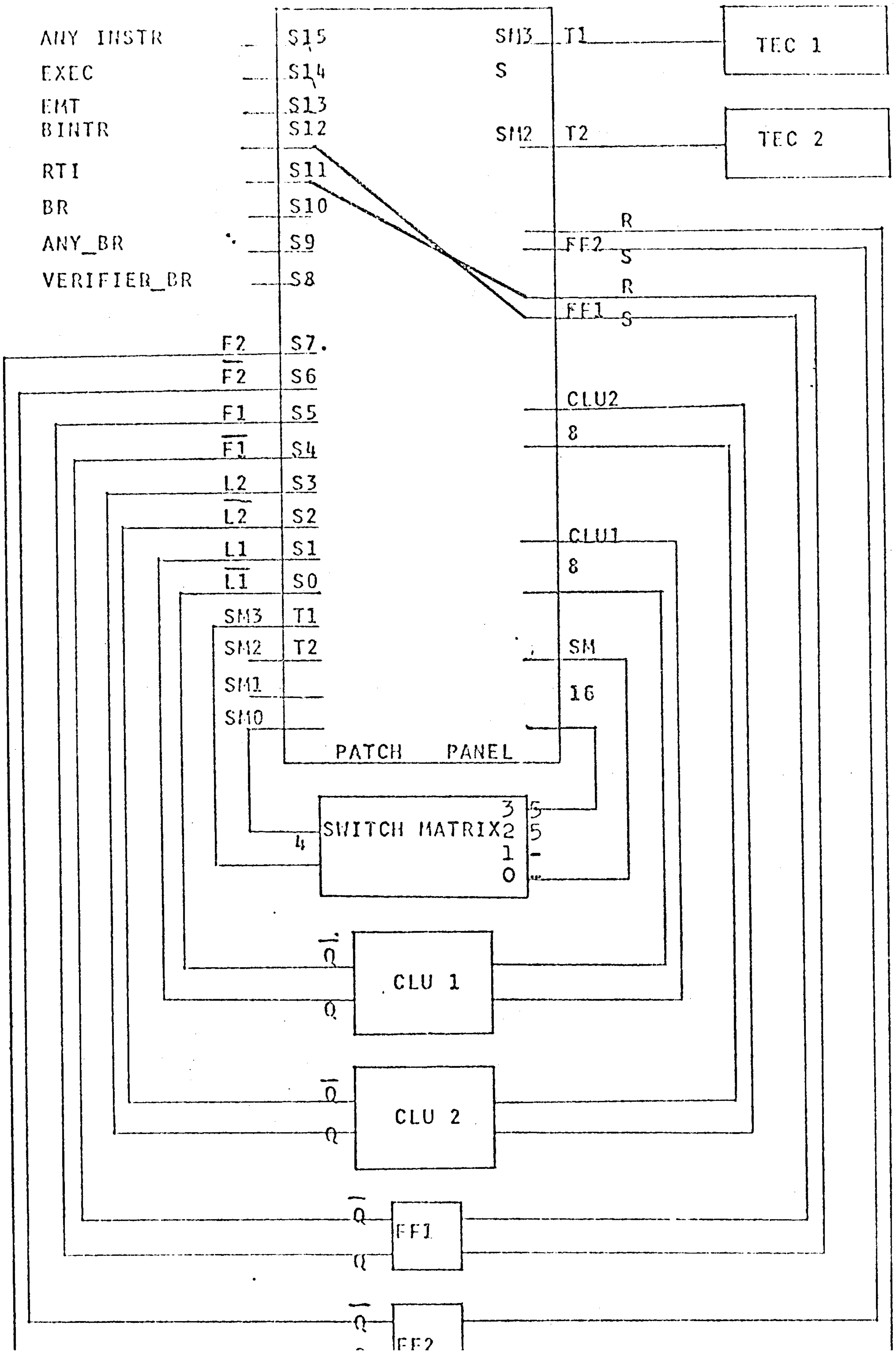


TABLE #7.4

ESTIMATED RESULTS

PROGRAM NUMBER	LOOP TIME us.	EXPECTED COUNT (10**6)	EXPECTED INSTRUCTION COUNT (10**6)
1	59.0±11.8	6.10±1.52	54.9±13.7
2	34.2±6.84	10.5±2.6	94.5±23.6
3	17.6±3.52	20.4±5.1	224±56

7.3.4.5 COMMENTS

A. The initial test in Method "C", to determine the loop time measured by MONINET will provide much more accurate results for Table #7.4. It is expected that this uncertainty and the time uncertainty together would provide a relative uncertainty under 6%.

B. Note that the figures given are calculated on instruction time and neglect the time between each instruction. This assumption will tend to cause the actual values for the loop count and instruction count to be less than predicted and the instruction time to be less than the experiment time.

C. The loop count is expected to be beneficial, due to the uncertainties involved.

7.3.5 SUMMARY

It is hoped these experiments (*) will prove that MONINET in its present status is an accurate measurement tool and that it is useful for monitoring processor states on a PDP-11/20.

It should be noted that as more components (eg. a hardware clock with a time-out feature, a larger SM, more counters, an ability to access and reduce object system data) are constructed MONINET will be able to perform the previous task easier and with better accuracy.

7.4 CONCLUSIONS

MONINET is a device which can be used to monitor a network of computers, a single computer system, or a telephone switching office. With some minor modifications, MONINET could also be used as the heart of a computer network diagnostic system.

(*) Some of the experiments proposed above were completed between the writing of this thesis and its publication. The results are in Appendix B.

SUMMARY, CONCLUSIONS and FURTHER WORK

8.1 SUMMARY

Consideration was given to the people interested in monitoring and to why they would want to monitor a system of computers. We conclude that people, who are responsible for a computer system, should do some research and discover if they can find a flexible monitoring system which can easily be used on their system by their own personnel and be cost-effective when compared to the costs incurred if their own system was inefficient or unreliable.

The next topic discussed was what items should be detectable by a monitor, when it is applied to the measurement of the workload and computer system paratmeters. A formal defintion of these items, which can be expanded into a measrement control language, was presented and explained. The interesting conclusion here, is that with this definition of an event, the procedure is to only detect the event, then perform at least one of the seven basic monitor actions. (See Section 4.1)

To perform the detection of these events, a tool is necessary. Three basic types of such monitoring tools have been created over the years (software, hardware and a combination of both). The thesis includes a comparison of types of monitoring tools. Besides the ten basic characteristics and limitations presented, there is still a need for the simple, single purpose, inexpensive monitors, but there is also an increasing requirement for a monitor system which is multi-purpose, in that it can accomplish many tasks for a variety of computer systems or networks.

The multi-purpose, flexible network monitor, MØNINET, which has been designed with these considerations in mind, was then presented. MØNINET appears capable of monitoring many different configurations of computer systems, and this is only a prototype.

Thus, we have covered the people, the tools, the techniques and some of the problems of monitoring computer systems and networks.

8.2 FURTHER WØRK

In the area of reasearch, besides the maintainance of an updated bibliography on performance measurements, we plan to contact at least one person who has either designed, built or used each of the monitors which we know exist. This would be beneficial, since it could complete Table #5.1; could permit another column on this table which presented the costs of the monitors; could provide enough facts about existing software monitors to permit us to build a chart for the physical characteristics of the present software monitors; and could finally lead to further research in the area.

With respect to MØNINET, the work planned entails the completion of general analysis routines, a measurement control language and the operating system; as well as further implementation of some of the modules of the RCHM. Also needed for each object system which we wish to monitor, will be load generators and a library of probe points.

BELL71S
Bell T.E., Boehm B.W., Watson R.A.
Computer Performance Analysis : Framework and Initial
Phases for a Performance Improvement Effort
Rand Report, R-549-PR, Aug. 1971.

BELL72C
Bell T.E., Boehm B.W., Watson R.A.
Framework and initial phases for Computer Performance
Improvement.
AFIPS, v41 (FJCC)Dec 1972, pp. 1141-1154.

BENN697
Benson E.
An Analysis of Terminal Systems
U.of Waterloo, Dep.of Applied Analysis and Computer
Science, Masters Thesis, July 1969.

BONN690
Bonner A.J.
Using System Monitor Output to Improve Performance
IBM Systems Journal, v8, No.4, 1969, pp. 290-298.

BORD714
Bordsen D.T.
Univac 1108 Hardware Instrumentation System
ACM SIGOPS Workshop on System Performance Evaluation,
Harvard U, April 1971, pp. 1-29.

BOYL735
Boyle L.
HARDMON: The U.of Waterloo Hardware Monitor.
U.of Waterloo,AA & CS Dept.,Masters Essay, May 1973.

BRYA69A
Bryan G.C., Shemer J.E.
The UTS Time Sharing System:Performance Analysis &
Instrumentation
Second Symposium on Operating System Principles, Oct
1969, pp. 147-158.

BUCH693
Bucholz W.
A Selected Bibliography on Computer System Performance
Measurement
IEEE Comp. Group News, March 1969, pp. 21-22.

BUNT728
Bunt R., Tsichritzis D.
An Annotated Bibliography for Operating Systems.
Computing Reviews, Aug 1972, pp. 377-398.

BURK732
Burke F.L.
A Computer Architecture for System Performance
Monitoring.
1st Annual SIGME Symposium on Measurement and
Evaluation, Feb 1973, pp. 161-169.

CAMP68C
Campbell D.J., Heffner W.J.
Measurement and Analysis of Large Computer Systems
During System Development
AFIPS, v33, pt. 1 (FJCC)Dec 1968, pp. 903-914.

CAMP732
Campbell J.A., Morgan D.E.
An Answer to a User's Plea.
1st Annual SIGME Symposium on Measurement and
Evaluation, Feb 1973, pp. 112-120.

CANN713
Canning E.G.
Get More Computer Efficiency
EDP Analyzer, March 1971.

CANN729
Canning E.G.
Savings from Performance Monitoring
EDP Analyzer, Sept. 1972.

CANN731
Canning E.G.
The Emerging Computer Network
EDP Analyzer, Jan. 1973.

CANN732
Canning E.G.
Distributed Intelligence in Data Communications
EDP Analyzer, Feb. 1973.

CANN733
Canning E.G.
Developments in Data Transmission
EDP Analyzer, March 1973.

CANT683
Cantrell H.N., Ellison A.L.
Multiprogramming Systems Performance Measurement and
Analysis
AFIPS, v32 (SJCC)Apr 1968, pp. 213-221

CARB653
Carbaugh D.H., Drew G.G., Ghiron H., Hoover L.S.
No. 1 ESS Call Processing.
Bell System Tech. J., Vol.43, Pt.3, 1965, pp.
2483-2531.

CARL718
Carlsen G.
A User's View of Hardware Performance Monitors or How
to Get More Computer for Your Dollar
IFIP Congress, Ljubljana, Yugoslavia, Aug 1971, TA-5,
pp. 128-132.

CARL728
Carlsen G.
Practical Economics of Computer Monitoring

IFIP Congress, Ljubljana, Yugoslavia, Aug 1971, pp.
193-198.

CHAN722
Chang J.H.
Some Design and Evaluation Techniques for Data
Communications Systems
IBM Research Report RC3736, Feb 1972, 31pp.

CHU*722
Chu W.W., Konheim A.G.
On the Analysis and Modelling of a Class of Computer
Communication Systems
IBM Research Report RC3727, Feb 1972, 58pp.

COLE71A
Cole G.D.
Computer Network Measurements : Techniques and
Experiments
UCLA PhD Thesis, Oct. 1971, UCLA-ENG-7165 (ARPA).

CONT642
Conti C.
System Aspects : S360/92
AFIPS, v26, pt. II (FJCC)1964, p. 81.

COOP729
Cooperman J.A., Lynch W.W., Tetzloff W.H.
SPG: An Effective use of Performance and Usage Data
Second Symposium on Operating System Principles,Oct
1969, pp. 20-23 OR Computer, Sep/Oct 1972, pp. 20-23.

DEUT719
Deutsch P., Grant C.A.
A Flexible Measurement Tool for Software Systems
IFIP Congress, Ljubljana, Yugoslavia, Aug 1971, pp.
7-12.

DOWN640
Downing R.W.,Nowak J.S.,Tuomenoksa L.S.
No. 1 ESS Maintenance Plain
Bell System Tech. J., Vol.43, Pt.3, Sep 1964, pp.
1961-2019.

DRUM690
Drummond M.E., Jr.
A Perspective on System Performance Evaluation
IBM Systems Journal, v8, No.4, 1969 pp. 252-263.

DYNA72?
Comress , Inc.
Dynaprobe: System Measurement Guide.
Comress Report No. CR2-0223.

DYNA7?
Comress , Inc.
Dynaprobe 7700: System Specifications.
Comress Report No. CP1-0023-2.

DYNA732
Compress , Inc.
Dynaprobe 7710: SystemuSpecifications.
Compress Report No. CR4-0023-2.

DYNA733
Compress , Inc.
Dynaprobe--Mini-Probe: System|Specifications.
Compress Report No. CR4-0031.

DYNA734
Compress , Inc.
Dynaprobe 7900: System Specifications.
Compress Report No. CR4-0023-1.

ESTR67C
Estrin G., Hopkins D., Coggan B., Crocker S.D.
SNUPER Computer
AFIPS, v30 (SJCC)1967, pp. 645-656.

FERR727
Ferrari D.
Workload Characterization and Selection in Computer
Performance Measurement.
IEEE Comp., v5, No.4, Jul/Aug 1972, pp. 18-24.

FREE710
Freeman, D.N.
Performance Measurement and Tuning for System 360
IEEE Intl. Comp. Society Conf., 1971, pp. 21-22.

FRE1725
Freiberger W.
Statistical Computer Performance Evaluation
Academic Press, New York, 1972.

FULL732
Fuller S.H.,Swan R.J.,Wulf W.A.
The Instrumentation of C.mmp, A Multi-(Mini)Processor.
IEEE Intl. Computer Society Conf., Calif.,Feb 1973,
pp. 173-176.

GOLD699
Goldhirsh I.
Data Communication Systems Performance and Evalutaion
Standards.
???

GOOD728
Good J., Moon P.A.M.
Evaluating Computers for the New Zealand Universities
Datamation, v18, No.11, Nov. 1972, pp. 96-99.

GROC69B
Grochow J.M.
Real Time Graphic Display of Time-Sharing System
Operating Characteristics
AFIPS, v35 (FJCC)Nov1969, pp. 379-386.

GROC72S
Grochow J.M.
Utility Function for Time Sharing System Performance
Evaluation
Second Symposium on Operating System Principles, Oct
1969, pp. 15-19, CI Computer, v5n5, Sep/Oct 1972, pp.
16-19.

HART70C
Hart L.E.
User's Guide to Evaluation Products
Datamation, v16, No.17, Dec. 15, 1970, pp. 32-35.
81101

HART71B
Hart L.E., Lipovitch G.J.
Choosing a System Stethoscope
Comp. Decisions, Nov. 1971, pp. 20-23.
DYNAPROBE,CMS.

HERM640
Herman D.J., Ihrer F.C.
The Use of a Computer to Evaluate Computers
AFIPS, v25 (FJCC)1964, pp. 383-395.

HERM672
Herman, D.J.
SCERT : A Computer Evaluation Tool
Datamation, v13, No.2, Feb. 1967, pp. 26-28.

HOLT711
Holt R.C.
On Deadlock in Computer Systems
University of Toronto, Tech. Report CSRG-6,Jan 1971.

HOLT714
Holtwick G.M.
Designing a Commercial Performance Measurement System
ACM SIGOPS Workshop on System Performance Evaluation,
Harvard U, April 1971, pp. 29-58.

HOLT726
Holt R.C.
Some Deadlock Properties of Computer Systems.
ACM SIGOPS Operating Systems Review, Third Symposium
of Operating Systems Principles, Jun 1972, pp. 64-71.

HUGH731
Hughes J., Cronshaw D.
On Using a Hardware Monitor as an Intelligent
Peripheral
Xerox Corporation, Jan. 1973.

HUGH733
Hughes J.
Performance Evaluation Techniques and System
Reliability - A Practical Approach
Xerox Corporation, March 1973.

IHRE72B
Ihrer, F.C.
Benchmarking vs Simulation
Computers and Automation, Vol.21, No.11, Nov. 1972, pp. 8-10, 13.

JOHN70C
Johnson, R.R.
Needed: A Measure for Measure
Datamation, Vol.16, No.17, Dec.15, 1970, pp. 22-30.

JOHN703
Johnston, I.Y.
Hardware versus Software Monitors
Proc of SHARE, XXXIV, Vol.1, March 1970, pp. 523-547.

KARU70[illegible]
Karush, A.F.
Two Approaches for Measuring the Performance of
Time-Sharing Systems
Software Age, March 1970, pp. 10-13. OR Second
Symposium of Operating Systems Principles, Oct 1969,
pp. 159-166

KARU717
Karush A.F.
Performance Measurement
Data Management, Vol 8, No. 7, July 1971, pp. 36-40.

KEEF680
Keefe D.D.
Hierarchical Control Programs for System Evaluation
IBM Systems Journal, v7, No.2, 1968, pp. 123-133.

KLAR700
Klar R.
A Counting Monitor for Measurements of Dynamic Program
Behaviour in Memory
Personal Communication, 1970.

KLEI691
Kleinrock L.
Scheduling, Queueing, and Delays in Time-sharing
Systems and Computer Networks.
???, Sep . Part of KLEI725.

KLEI640
Kleinrock L.
Communications Nets: Stockastic Message Flow and
Delay.
McGraw Hill Book Co., Toronto, 1964.

KLEI720
Kleinrock L.
Queueing Systems: Theory and Applications
John Wiley and Sons, 1972.

KLEI725
Kleinrock L.
References for Talks on--Analytical Models for
Time-sharing Systems and Models, Measurement and
Simulation of Computer Networks.
Unpublished, May 1972.

KLEI729
Kleinrock L.
Performance Model and Measurement of the ARPA Computer
Network.
Online 72, Conf. Proc., v2, Sep 1972, pp. 61-88.

KNIG66[illegible]
Knight K.E.
Changes in Computer Performance
Datamation, v12, No.9 Sept. 1966, pp. 40-49, 54.

KNIG681
Knight K.E.
Evolving Computer Performance 63-68
Datamation, v14, No.6, Jan. 1968, pp. 31-35.

KOBA728
Kobayashi H.
Some Recent Progress in Analytic Studies of System
Performance
IBM Research Report RC3960, Aug. 1972, 31 pp.

KOHN717
Kohn C.F.
Organization for System Evaluation
INFOR v9, No.2, July 1971.

KOLE699
Kolence K.W
System Improvement By System Measurement
Data Base, Winter, 1969, pp. 6-11
10036

KOLE711
Kolence K.W.
A Software View of Measurement Tools
Datamation, v17, No.1, Jan 1, 1971, pp. 32-38

LUCA719
Lucas H.
Performance Evaluation and Monitoring
Computing Surveys, v3, No.3, Sept 1971, pp. 79-91
SOFTWARE-MONITOR.MODEL.

LUCA720
Lucas H.C
Synthetic Program Specifications for Performance
Evaluation
Proc. of the ACM Natl. Conf., 1972, pp. 1041-1058

LYNC727
Lynch W.C.
Operating Systems Performance
CACM, v15, No.7, July 1972, pp. 579-585

MADL722
Madden R.L.
Software Accounting and the Hardware Monitor: Their
Marriage in Performance Analysis
Proc. of the ACM Natl. Conf., 1972, pp. ???

MEND732
Mendicino S.F., Sutherland G.G.
Perotormance Measurements in the LLL Octopus Network.
COMPCON, Feb 1973, pp. 109-125.

MILL717
Miller E.F., Jr., Hughes D.E., Bardens J.A.
An Experiment in Hardware Monitoring
General Research Corporation, RM-1517, July 1971,

77,pp.

MURP69B
Murphy R.W.
The System Logic and Usage Recorder
AFIPS, v35 (FJCC)Nov1969, pp. 219-229

OPLE647
Opler A.
Measurement of Software Characteristics
Datamation, v10, No.7, July 1964, pp. 27-30

PATR647
Patrick R.L.
Measuring Performance
Datamation, v10, No.7, July 1964, pp. 24-27

ROEC69C
Roeck D.J., Emerson W.C.
A Hardware Instrumentation Approach to Evaluation of
Large Scale Systems
Proc. of the 24th ACM Natl. Conf., 1969, pp. 351-367

ROSE72C
Rose C.W.
LOGOS and the Software Engineer.
AFIPS, (FJCC)Dec 1972, pp. 311-324.

ROTA65S
Rota F.A., DePuyt J.B.Jr
Trafficism - A Design Evalutaion Tool
Proc. of the 20th ACM Natl. Conf., Aug 1965, pp. 38-53

ROZW732
Rowzadowski R.T.
A Measure for the Quantity of Computation.
1st Annual SIGME Symposium on Measurement and
Evaluation, Feb 1973, pp. 100-111.

SALT69A
Saltzer J.H., Gintell J.W.
The Instrumentation of Multics
Second Symposium on Operating System Principles, Oct
1969, pp. 167, 174 CA Communications, v13#8, Aug 1970,
pp. 495-504.

SCHE664
Scherr A.L.
Time Sharing Measurement
Datamation, v12, No.4, Apr 1966, pp. 22-26.

SCHW72C
Schwetman H.D., Brown J.C.
An Experimental Study of Computer System Perfomance
Proc. of the ACM Annual Conf., 1972, pp. 693-703

SHAW72C
Shaw A.C.
The Logical Design of Operating Systems.

U.of Washington, Comp. Science Group, Seattle,
Washington, Dec1972.

SHEM689
Shemer J.E., Heying D.W.
Performance Modeling: Empirical Measurement in a System
Designed for Batch and Time Sharing Users
AFIPS, v35 (FJCC)Nov 1969, pp. 17-26

STAT642
Statland N.
Methods of Evaluating Computer System Performance
Comp. and Automation, v13, No.2, Feb 1964, pp. 18-23

STIM69
Stimler S.
Real Time Data-Processing Systems
McGraw-Hill Book Co.,Toronto, 1969

TESD720
Tesdata Sys.Corp.
System 1000-Computer Performance Management System,
1972
Tesdata Sys. Corp., Chevy Chase, Maryland 20015, 5530
Wisconsin Ave.,

TESD730
Tesdata Sys.Corp.
Load Generator System, 1973
Tesdata Sys. Corp.
Virginia 22101

TESD731
Tesdata Sys.Corp.
System 1000-General Information 1100 Series, 1973
Tesdata Sys Corp.
Virginia 22101

TSIC728
Tsichritzis, D.
Lecture Notes of Operating Systems.
U.of Toronto, Comp. Science Dept., Report
No.TR-44,Aug. 1972.

WARN711
Warner C.D.
Monitoring: A Key to Cost Efficiency
Datamation, v17, No.1, Jan. 1, 1971, pp. 40-42, 49

WATS70
Watson,R.W.
Time-sharing System Design Concepts.
McGraw-Hill Book Co.,New York,1970.

WATS71
Watson R.
Computer Performance Analysis: Applications of
Accounting Data
Rand Report, R-573-NASA/PR, May 1971, 70pp.

WIEN701
Wiener J., DeMarco T.
Tuning for Performance
Modern Data, Jan. 1970, p. 54

WILE700
Wiley Systems Inc.,
System Performance Measurement and Analysis: Notes.
ACM Professional Developement Seminar, pp 105

APPENDIX "A"

MONITOR EVENTS--CLASSIFIED BY ATTRIBUTES

A. TIME ONLY

1. To set up a logical or physical path through a node or network.

2. To disconnect a logical or physical path through a node or network.

3. To detect, correct, or recover from trouble within the network, for the following troubles:

   a. Data transmission error;

   b. Line out of service;

   c. Node out of service;

   d. Storage device out of service;

   e. Software error in a node;

   f. Host out of service, and

   g. Link problems.

4. To detect or take appropriate action for network overload due to saturation of a line, node, storage, or a Host.

5. To disassemble or reassemble a message into/from a set packets.

6. For a program to execute.

7. Percentage of CPU-I/O overlap.

B. FREQUENCY

1. For transmiting a message or packet through a net, node or specific component.

2. For service calls.

3. Of time intervals between arrivals of service calls.

4. Of time intervals between departures of service calls.

5. Utilization of software and hardware resources for nodes and hosts.

   a. Software activity by module or groups of modules.

   b. CPU activity.

   c. Line or link activity.

   d. Channels, controllers or auxilliary storage devices.

   e. Main storage devices.

   f. Data set activity.

   g. File structure activity.

6. In each processor state.

7. Of user response time intervals.

8. Of task switching.

C. COUNT ONLY

1. For setting up logical or physical paths via nodes or networks.

2. For disconnecting logical or physical paths via nodes or networks.

3. The number of packets, messages, or jobs handled by a net, node or host.

4. The number of bits transmitted or received by a line, link, node, host or network.

5. The number of requests for service.

6. The number of changes of state of a processor.

7. The number of times an instruction is executed.

8. Raw speed of a component--hardware or software.

9. Number of active jobs.

10. Number of messages, packers, jobs in a system.

D. SPACE AND TIME

1. Reference patterns of software.

2. Auxillary storage space used in a system, net or node.

3. Main Storage space used or requested in a system, network or node.

4. I/O organization and selection by channel, controller, device and device components.

E. MAGNITUDE

1. Of a selected queue or queues.

2. Of messages, packets, or jobs in various units of size.

3. Of a data set for storage in either main or auxilliary.

F. LOGICAL AND SEQUENTIAL COMBINATIONS OF THE ABOVE.

APPENDIX "B"

DEM 1

18-19 January 1974

EXPERIMENT 1

I. Purposes:

A. To obtain average instruction time on PDP-11/20 for various types of instructions.

B. To determine whether the monitor is working properly.

II. Procedure

A. Outline of Procedure

1. Locate signals which define start and end of each instruction.

2. Define and construct probes to detect these signals and condition them to match TTL logic levels of the monitor.

3. Decide which monitor components are necessary.

4. Decide what instruction mixes should be measured.

5. Write the monitor control program and the data analysis program.

6. Perform the experiment with each of the instruction mixes.

B. Signal Location

Figure B.1 depicts the sequences of states involved in the execution of an instruction on the PDP-11/20. On many instructions, the start of the Fetch signal (see PDP-11 Processor Manual) indicates the start of an instruction, while the end of the "execute" signal (see PDP-11 Processor Manual) indicates the end of execution (see Figure B.2). Using oscilloscope and Processor Manual we set out to determine when this was not the case, i.e., under what circumstances do we not get an execute pulse to terminate execution of an instruction? (see Figure B.3)

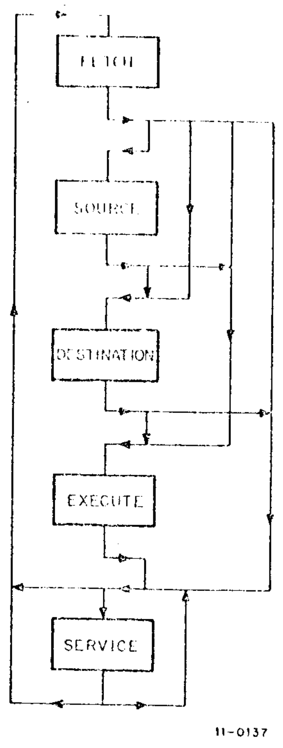


Figure B.1 KA11 Major State Flow

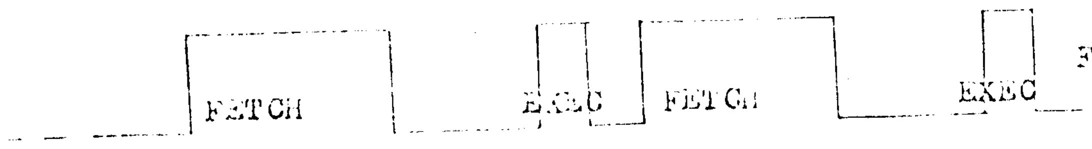


FIGURE B.2

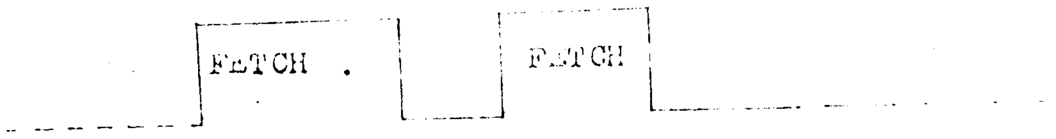


FIGURE B.3

The following test program confirmed our suspicion that a conditional branch with an unsatisfied condition would not have an execute pulse.

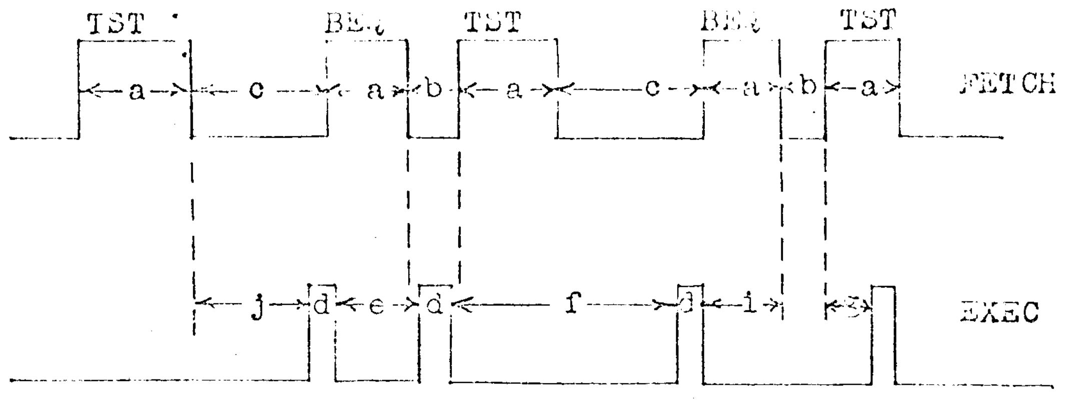
```
LOOP      TST      SWR

          BEQ      LOOP

          BR       LOOP
```

Placing oscilloscope probe no. 1 on the "FETCH" signal and no. 2 on the "EXECUTE" signal yielded the pattern shown in Figure B.4 when SWR=0, and the pattern shown in Figure B.5 when SWR≠0.

In order to get around this problem, the circuit shown in Figure B.6 was set up and the output was placed on the oscilloscope. The result is shown in Figure B.7. The spikes are too fast for us to monitor reliably.

FIGURE B.4

SWR=0



a= 1.2 ms

b= 1.4 ms

c= 3.0 ms

d= 0.4 ms

e= 1.7 ms

f= 4.4 ms

g= 0.24 ms

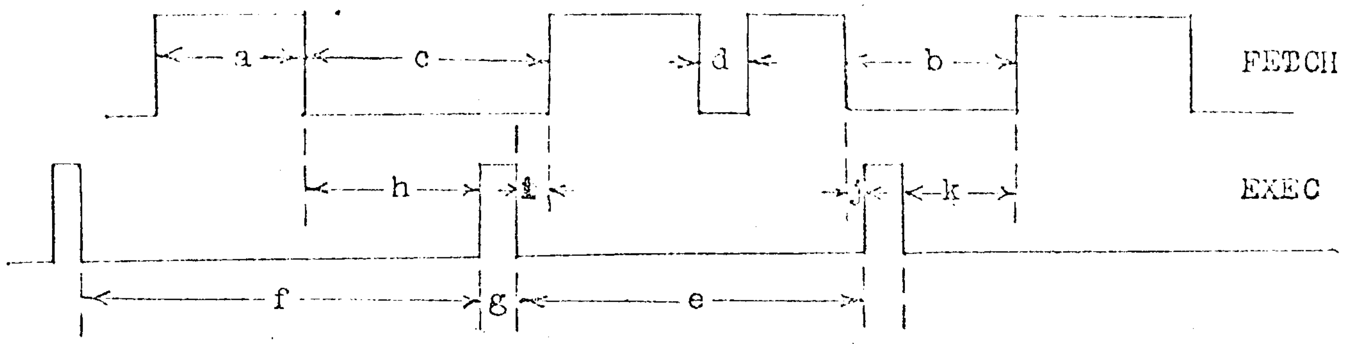
h= 0.8 ms

i= 0.24 ms

j= 2.4 ms

FIGURE B.5

SWR ≠ 0



a	=	1.2 ms
b	=	1.4 ms
c	=	3.0
d	=	0.25
e	=	3.2
f	=	4.4
g	=	0.4
h	=	2.4
i	=	0.25
j	=	0.24
k	=	0.8

The next approach was to set a flip-flop when the instruction fetch started and reset it when the execute pulse ended or when the "service" state was entered, meanwhile counting the number of instruction fetches and measuring the time the flip-flop was set. These signals were found, and signal conditioning probes were constructed appropiately (see Figure B.8 for the signals currently available from the processor for this and other experiments).

FIGURE B.8

1. Fetch and ISR0
2. Execute
3. Service and ISR8
4. Branch (conditional)
5. Branch (non-conditional)
6. Bus Interrupt
7. Branch Instruction
8. EMT.

C. Monitor components needed

2 Timer and Event Counters

Patch Panel

1 Swtich Matrix

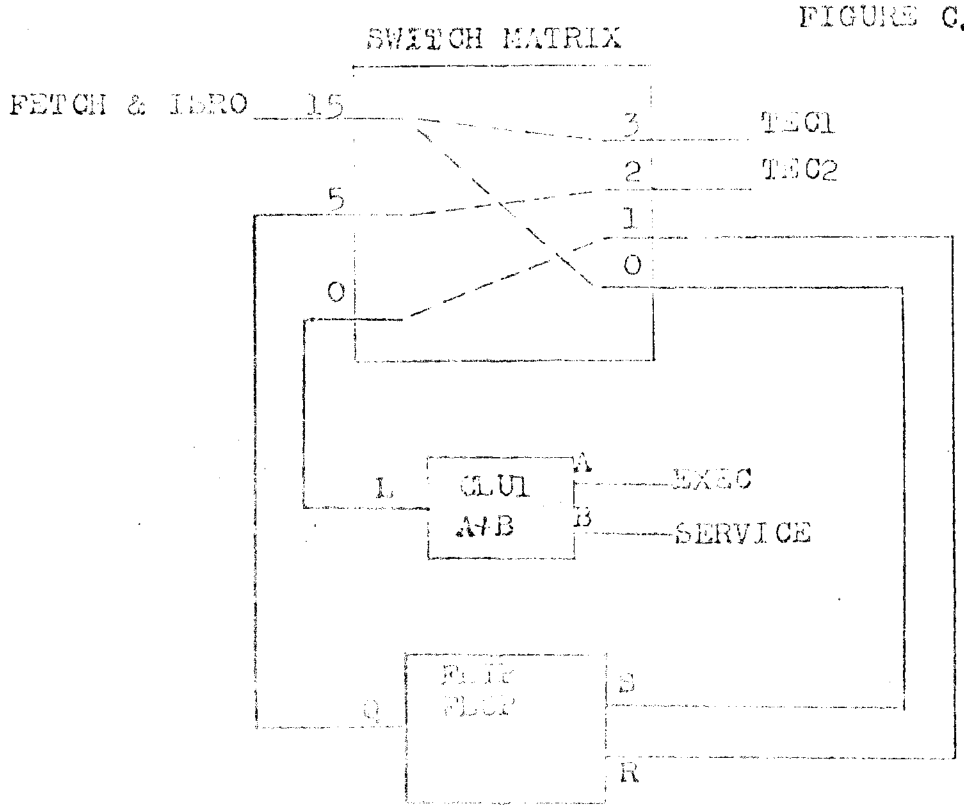
1 Flip-flop

1 Combinational Logic Unit

Probes to get "fetch and ISR0", "execute" and "service and ISR8".

The configuration used was shown in Figure C.1.

FIGURE C.1



D Monitor Control Program

Figure D.1 is a listing of the simple program used to control the monitor for this experiment.

E Instruction Mixes Measured

1.	NOP HALT	Performed 18 January 1974 by Goodspeed and Jederman.
2.	NOP NOP HALT	
3.	5 NOP's HALT	
4.	10 NOP's HALT	

We discovered with this instruction mix on 19 January 1974 that the "halt" instruction terminates only when the console says to start executing somewhere; thus, our function for shutting off the timer measuring instruction duration remained on until the end of the experiment.

We then tried the programs in Figures E.1, E.2, and E.3.

FIGURE D.1

```
      EXTEST.MON
      COMMON/CONFIG/MONITR,UNIT
      INTEGER START(2),TIME(2),EVENT(2),N(2)
      INTEGER UNIT,COMAND
C
      MONITR=1
      UNIT=1
      CALL DISCON
      CALL DELLOG(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
      CALL MASTER(4)
      CALL MASTER(1)
      UNIT=1
      L1:=A+B
      CALL CONECT(15,0,15,3,4,2,0,1)
C
5     WRITE(6,100)
100   FORMAT(' ','*',/)
      READ(6,110) COMAND,UNIT
110   FORMAT(A2,I2)
      IF(COMAND.NE.'ST')GO TO 20
      CALL TV HALT(1,2)
      CALL GETIME(N)
      WRITE(6,140)N(2),N(1)
140   FORMAT(' END=',O6,1X,O6)
      GO TO 5
20    IF(COMAND.NE.'GO') GO TO 30
      CALL GETIME(START)
      CALL TVGO(1,2)
      WRITE (6,120) START(2),START(1)
120   FORMAT(' ','START=',O6,1X,O6)
      GO TO 5
30    IF(COMAND.NE.'RE') GO TO 40
      CALL READTV(TIME,EVENT)
      WRITE(6,130) TIME(1),TIME(2),EVENT(1),EVENT(2)
130   FORMAT(' ','TIME=',O6,1X,O6,/,' EVENT=',O6,1X,O6)
      GO TO 5
40    IF(COMAND.NE.'CL')GO TO 50
      CALL SETTV(3,0,1,1)
      GO TO 5
50    IF(COMAND.NE.'EX') GO TO 5
      STOP
      END
```

FIGURE E.1

```
            R0 = %0
START :     MOV #10.,R0
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DEC R0
            BGT LOOP
            HALT
            .END START
```

FIGURE E.2

```
                R5 = %5
;
START:          JSR  R5,X
                BR   START
X:              JSR  R5,Y
                RTS  R5
Y:              JSR  R5,Z
                RTS  R5
Z:              RTS  R5
                .END START
```

FIGURE E.3

```
            EMT
            BR   .-2
x           TRAP
            RTI
Y           IOT
            RTI
Z           RTI
```

III RESULTS

NOP HALT	required 2.8 microseconds, as compared with 3.1 microseconds with the ±20% quoted by DEC.
NOP NOP HALT	required 4.1 microseconds compared with 4.4 us. quoted by DEC.
NOP NOP NOP NOP NOP HALT	Required 8.4 us. compared with 8.4 us. estimated by DEC.
10 NOPs HALT	required 15.1 us. compared with 15.0 us. qu quoted by DEC.
START NOP 9 NOPs BR START	run for 30.1 seconds yielded 24.14 sec for for fetches of 14,654,700 instructions and and 30.05 seconds to execute these instruction. Only 1,604,987 instruction terminations were noted.

The average instruction time was 1.7 us., and the ratio of fetches to executes was exactly 11, as it should be.

The same average instruction time and ratio were obtained when the experiment was run again.

Figure E.2 The average instruction time for this experiment was measured as 3.75 us. vs 3.67 calculated from DEC's data.

Figure E.3 Average measured was 5.93 us. vs. 6.5 calculated.

All of these resulsts are well within DEC's error tolerance of 20%.