A STUDY OF INFORMATION IN MULTIPLE-COMPUTER AND MULTIPLE-CONSOLE DATA PROCESSING SYSTEMS

K. B. Irani
I. S. Uppal
G. A. McClain
D. L. Hinshaw

University of Michigan

FOREWORD

This document, designated Annual Report No. 5, was submitted by the Department of Electrical Engineering, Systems Engineering Laboratory, the University of Michigan, Ann Arbor, Michigan. The research was sponsored by Rome Air Development Center, Griffiss Air Force, Base, New York, under contract F30602-69-C-0214, Job Order Number 55810006. Mr. Rocco F. Iuorno, ISIM, was the RADC Project Engineer.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS).

This technical report has been reviewed and is approved.

Approved:

ROCCO F. IUORNO
Project Engineer
Rocco F. Iuorno

Approved: William P. Bethke
WILLIAM P. BETHKE
Chief, Information Sciences Division

FOR THE COMMANDER: Fred I Diamond
FRED I. DIAMOND
Acting Chief, Plans Office

ABSTRACT

This report summarizes the achievements from April 1969 to June 1972 of continuing research into the development and application of mathematical techniques for the analysis and optimization of multiple-computer, multiple-user systems.

EVALUATION:

The purpose of this research was to apply mathematical techniques to aid in the analyses and systemization of hardware and software configurations for the solution of general purpose data processing problems. The work covers three years of effort and is documented in ten technical reports. Topics in this investigation included: computer memory system design, communication system design, microprogramming control, central processor design, optimum program pagination, data base structures, optimization of computer scheduling, and performance monitoring.

The studies pursued are directly related to the field of information processing. They provide a theoretical treatment of the various components that make up or support an information processing system. The rigorous treatment in paging algorithms, scheduling theories in a time sharing environment, modeling software for test and evaluation, and effects of reentrant coding on program/computer efficiency can be effectively used by those developing or implementing advanced computer operating systems. Designers faced with providing systems to handle large data bases with remote access capabilities will find the work in communication systems very helpful The modeling work in data base structures can be used by data management researchers to assist them in selecting the proper structure for their particular application. The research in central processor design, memory design, and data path optimization, can be used by those working in advanced computer designs.

The quality of reports generated from this research are excellent with most of them approaching the beyond the-state-of-the-art in information processing. The reports are getting a wide dissemination especially to those active in the field.

This program has had an impact on the information processing research program at RADC (Information Processing Branch). The theoretical investigations have provided credibility and technical depth to the Branch's exploratory research in: advanced operating system development, associative processor design, data management development, distributive data base systems, and test methodology for complex software systems. The fruitful results from this study have warranted the initiation of a follow on three year effort which will address topics in security and protection, computer languages, test methodology , data management, program documentation, advanced computer designs, and software reliability.

ROCCO F. IUORNO

Rocco F. Iuorno

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Contract Objective

This effort is for applied research in the area of mathematical techniques for analyzing multiple computer, multiple console, real-time on-line data processing systems, and for analytical techniques and hypotheses to assist system designers and users in determining the optimum configuration, most complete utilization, and most efficient scheduling of this type of system.

The main objective of this work is to make it possible, through development and application of new mathematical techniques, to more optimally design and control computer systems. A computer system consists of a collection of electronic data processing machines, data transmission channels, and multiple user terminals, organized to efficiently service the computational needs of a geographically or functionally diverse population of users. Such systems permit: remote communication and manipulation of shared data bases; cooperative operations between user and computer (symbiosis); an immediate access to a high-capability facility for problem solving and data manipulation.

1.2 Contract Requirements

1) Exploration of Queueing Theory to enable the analysis of more general models of computer utilities and their subsystems. Emphasis shall be concentrated on numerical techniques, and

shall include extension of earlier work on quasi-birth-and-death (QBD) models and the Recursive Queue Analyzer.

2) Collection and analysis of statistical data from existing systems to determine the validity of the mathematical models developed, and to isolate problem areas in need of attention. The techniques of computer data collection shall be studied.

3) Application of new mathematical techniques in conjunction with those previously available, to the analysis and optimization of hardware and software configurations of general purpose computers. These techniques shall be applied to typical systems in order to test the analytical methods and provide specific analyses/recommendations concerning the effectiveness of these systems.

4) Continued development of general design guidelines for time-shared computer systems with distributed processing capabilities. Distributed processing has become economically feasible because of rapidly decreasing small computer costs. This task shall extend previous investigations of remote display terminal structures.

5) Continued development of mathematical models for the optimal structuring of communication networks associated with computing systems.

6) Continued development of optimal design of storage systems and data base structures.

7) Continued exploration of Descrete Optimization Theory and Graph Theory in relation to application concerning the scheduling of programs in multi-processor systems. Also to investigate the use of these theories in relation to problems of program organization.

8) Development of new conclusions and rules which can be used by persons performing initial designs of real-time computer systems having a large number of user consoles. Such rules shall allow system designers to more rapidly choose the type of hardware/software system needed to fit a particular organization or problem.

9) Application of statistical analysis to data collected from various computing systems in order to gain an understanding of user demand structures and their effects on systems performance. A search for other theoretical approaches to the analysis of multiple computer systems shall be pursued.

1.3 Progress Toward Contract Objectives

During the course of this contract, work progressed over the entire spectrum of contract requirements, in the realm of both theoretical and empirical investigations. As a result of this effort, ten RADC technical reports [1,2,3,4,5,6,7,8,9,10] were published. These reports document the theoretical and empirical investigations conducted, the conclusions reached, and some typical applications. A number of papers based on the results of these reports will soon be presented [11,12].

The research conducted under this contract can be categorized into several main efforts. The first of these, discussed in detail in RADC-TR-70-272 [1] and previous annual reports [9,10], is concerned with the area of the design of message processing and communications systems.

The second effort concentrates on the application of optimization theory to the problems of program organization and program scheduling in complex multi-programmed computer systems [3,4] . Work was also conducted in the analysis and optimization of such systems using storage hierarchies [2] . The third area of endeavor attacks the problem of optimal representation of data structures within the computer memory. Under this category two approaches were formulated; one focuses on a particular application, namely interactive computer graphics [5] , while the second handles this problem in a very general manner by developing a generalized computer representation which is capable of representing any given collection of data [6] . The fourth area investigates the design and optimization of digital computer central processors [7,8] . For the sake of completeness, abstracts of all the RADC Tech. Reports published under this contract are included in section 2. For details the reader is referred to the references at the end of this section. Appendix A contains a reprint of a paper which will be presented at the Ninth Annual Design Automation Workshop at Dallas, Texas.

References for Section 1

1) V.K.M. Whitney, "A Study of Optimal File Assignments and Communication Network Configuration in Remote - Access Computer Message Processing and Communication System," RADC-TR-70-272, Jan 1971, AD 881 929L.

2) A. M. Woolf, "Analysis and Optimization of Multiprogrammed Computer Systems Using Storage Hierarchies," RADC-TR-71-165, Aug 1971, AD 730 325.

3) D. Coleman, "On Binding Groups - A Quadratic Programming Approach in Zero/One Variables with Applications," RADC-TR-71-162, Aug 1971, AD 729 437.

4) J. W. Boyse, "Solutions of Markov Renewal Decision Processes with Application to Computer System Scheduling," RADC-TR-71-180, Sep 1971, AD 731 220.

5) J. H. Jackson, "Optimum Implementation of Topological Structure for Interactive Computer Displays," RADC-TR-71-91, May 1971, AD 885 492.

6) L. S. Randall, "A Relational Model and its Optimization for the Representation of Structured Data Within a Random-Access Computer Memory," RADC-TR-72-25, Feb 1972, AD 740 581.

7) D. Hinshaw, "Optimal Selection of Functional Hardware For Microprogram Controlled Central Processors," RADC-TR-72-182, Apr 1972, AD 748 221.

8) G. A. McClain, "The Optimal Design of Central Processor Data Paths," RADC-TR-72-183, Apr 1972, AD 748 224.

9) K. B. Irani, J. W. Boyse, et. al., "A Study of Information Flow in Multiple-Computer and Multiple-Console Data Processing Systems," RADC-TR-70-87, May 1970, (no AD number).

10) K. B. Irani, I. S. Uppal, et. al., "A Study of Information Flow in Multiple-Computer and Multiple-Console Data Processing Systems," RADC-TR-71-160, Aug 1971, AD 729 194.

11) V.K.M. Whitney, "Minimal Spanning Tree," Algorithm 422, CACM, Vol. 15, No. 4, April, 1972, pp. 273-274.

12) K. B. Irani, G. A. McClain, "Optimal Design of Central Processor Data Paths," To be presented at the 9th Annual Design Automation Workshop, Dallas, Texas, 1972.

2. ABSTRACTS OF TECHNICAL REPORTS

In this section we present abstracts of various technical reports prepared as a result of this research effort.

2.1 Message Processing and Communication System

The goal of this research is the solution of several problems involved in the optimal design of computerized message processing and communication systems. The class of systems studied includes airline and hotel reservation systems, time-shared computer systems, and corporate message communication systems.

A precise mathematical model of the systems being studied is formulated to draw attention to the fundamental subsystems and to clarify the basic independent variables of the design processes. Systems considered must have a large data base organized into record files, widely distributed users, an on-line communication network joining users to data base sites, a quantitative performance measure and specified performance constraints. The most general system model studied uses a queueing model for individual communication channels, with the channel interconnections specified by a weighted linear graph. The data base is assumed to be organized into files of homogeneous records, and the processing costs are specified by a single function of the number of messages processed.

One major problem area studied is the determination of the optimal number and locations of sites for the system files. Several basic properties of file site assignment on linear graphs are demonstrated and used in efficient procedures for file assignment.

When the file locations are specified, the communication requirements of the system are known. A second major problem area considered is the optimal design of communication channels and networks of these channels. A thorough study of stochastic message transmission channels has been undertaken. The effects of channel capacity with one or more channels, message length distributions, message retransmission order, and departures from the model assumptions on message delay are studied, both analytically and by simulation. Several channel and network performance measures are defined and compared. The problem of the optimal allocation of channel capacity among the channels of a network is solved for several important system models. A complete set of guidelines is given to aid the designer of communication systems and channels.

The third major problem area studied is the optimal design of communication network topologies. Several previously available techniques for the design of centralized networks are critically evaluated. New techniques for the design of centralized and noncentralized networks are presented. These new procedures more faithfully model communication networks than the other procedures by including more realistic cost considerations, and yield more general solutions.

Finally the solution procedures of these problems are integrated into a systematic design procedure for a general class of computer message processing and communication systems. An example of a complete system design is given.

2.2 Multiprogrammed Computer Systems

2.2.1 Multiprogrammed Systems Using Storage Hierarchies

This research is concerned with the problem of predicting and controling the behavior of complex computing systems using storage hierarchies and multiprogramming.

A computing system using a storage hierarchy generally involves 2 or more different type of storage devices. Different device types have widely varying performance characteristics. User programs can be big or small. They can access data in a serial or random manner. They can do large quantities of I O or almost none at all. The data paths and routing of information influence the load seen by storage devices. The logical record sizes throughout the system and the number of programs running in a multiprogrammed system also affect the system performance. All of these factors and many others combine to create a complex and remarkably difficult analysis problem.

The specific objective of this research is to develop and demonstrate a mathematical model of computing systems which accounts for the above factors. Such a model has been developed and implemented as a fast, highly interactive computer program. The model developed exhibits the following characteristics:

1. The model includes the effects of user program behavior, operating systems characteristics and hardware performance. Models for core or random access devices, drums, disks and data cells have been developed.
2. The model is versatile and easily applied to a wide range of system configurations. The model is useful as a tool for both analysis and optimization. Investigation of computing systems in general as well as detailed investigations of a particular system can be carried out.
3. The results obtained from the model approach the accuracy and realism of those obtained from simulation models. A number of examples showing the usefulness and the generality of the model are given.

2.2.2 Optimal Task Scheduling

In multiprogrammed computer systems, the way in which system resources are allocated among tasks can strongly influence system efficiency. In such systems, as opposed to monoprogrammed computers, separate but interdependent consideration must be given to scheduling of two critical system resources: main memory and the central processors. A rather wide class of scheduling procedures is available to the designer of such a system, and so it is useful to be able to determine task priority rules which allow the maximum number of task completions per unit time or throughput. This research pursues the theoretical development and application of Markov renewal decision processes for this purpose.

A major effort is devoted to theoretical development of a new technique for finding the optimal stationary policy in a class of infinite time Markov renewal decision processes in which the criterion is either maximization of expected return per transition or maximization of expected return per unit time. The modeling of stochastic service systems with Markov renewal decision processes commonly leads to transition matrices which are sparse, and the derived optimization method offers definite computational advantages for this wide class of practical problems.

A Markov process is used to model a problem of current import in the software design of multiprogrammed computer systems; namely, the problem of assignment of central processors and main memory to those jobs requesting them. Application of the optimization method to this model allows the determination of scheduling rules which maximize throughput. An additional complexity in scheduling arises when tasks are allowed to share information with other tasks in main memory, and this problem is treated also. Parameters which may be varied in the model include the size of the main memory relative to the sizes of individual tasks, the execution characteristics of tasks themselves, the number of central processors, and the fraction of its total memory requirement each task may share with other tasks. Paging is important in many modern computer systems and is treated indirectly in the model by treating it as a special kind of input/output operation. Results for a range of values of parameters are presented which compare the throughput possible under optimal scheduling rules and under a good heuristic scheduling rule.

In order to obtain results applicable to a typical general purpose multiprogrammed computer system, data from such a system were collected and analyzed in order to set appropriate values for parameters of the model and also to test the validity of certain assumptions made in the model. These data are general enough to be of interest in their own right and are therefore presented in the form of graphs and tables as part of this research effort.

2.2.3 On Finding Optimal Groups - A Quadratic Programming Approach in Zero/One Variables with Applications

In this research we consider a mathematical programming problem of the following form:

$$\text{Maximize} \qquad f(\underline{x}) = \sum_{i,j} c_{ij} x_i x_j - \sum_{j} b_j x_j$$

$$\text{subject to} \qquad \sum_{j} a_j^k x_j \leq \mu_k \qquad k = 1, 2, \ldots, m,$$

where b_j is an arbitrary real, $c_{ij} \geq 0$ for all i, j, and a_j^k, μ_k are positive reals for all j and k, $x_j \in \{0, 1\}$ for all j.

For values of $m \leq 2$ we present an algorithm for solution of this problem based upon generalized Lagrange multipliers. This algorithm is suitable for application to problems with a rather large number of binary variables.

This formulation is shown to be applicable to a number of optimal grouping problems; a particular application to computer systems paging is examined in detail.

2.3 Data Structures and Their Representation

2.3.1 Computer Memory Data Representation

In solving any problem on a digital computer one must store within the computer memory certain pieces of information, or data, upon which the program implementing the solution process makes its decisions or performs its calculations. For virtually all problems the manner in which the relationships of accessibility among the various items are portrayed by the organization of these items in the computer memory can have a marked effect upon the efficiency of the solution process. The objective of the research reported here is the development of a rigorous quantitative method for the automatic design of optimal computer memory representations for data.

To this end, we define a relational model of data structure within which we can specify the logical ordering or structure of the data involved in the solution of a given problem. We then develop a decision model for the specification of the storage structures (i.e., the computer memory representations) which can represent an arbitrary data structure.

We define two basic measures of performance - a time cost function and a storage cost function - for use in comparing the relative merits of a collection of storage structures. The time cost function reflects the number of time units required to perform certain of a set of primitive operations using a particular storage structure, and the storage cost function reflects the total number of storage units occupied by the storage structure.

Finally, we present a procedure which determines of the storage structures which can feasibly represent a given data structure the storage structure for which the time and storage costs satisfy certain optimality conditions. A storage structure is considered to be optimal if it minimizes the time cost function, subject to an upper limit on its storage cost.

To demonstrate the feasibility and the effectiveness of the techniques we develop, we apply them to a problem for which a solution program already exists and for which fairly extensive data about system performance are available. Our results demonstrate conclusively that significant improvement in program efficiency can be obtained by applying these techniques, as opposed to the historically intuitive and qualitative methods used for storage structure design.

2.3.2 Optimal Implementation of Interactive Display Data Structures

One of the many uses of data structures in connection with computer graphics has been to represent hierarchies of picture parts so that a picture may be easily modified. Essentially one data structure has been used for this purpose, although the means by which it has been implemented has assumed various forms. Previous research has described various implementations of this data structure, rather than the information which it contains. Because the data structure has not been described independently of a particular implementation, past comparison of implementations has been largely heuristic.

In this research, a mathematical model of the common display data structure is given so that it may be considered independently of any particular implementation. Then, a method is presented for applying this model to determine the implementation of the data structure which minimizes response time to user inputs for a given application on a given hardware configuration.

2.4 Central Processor Design

2.4.1 Optimal Design of Central Processor Data Paths

This research deals with the design automation of the central processor of a digital computing system. The principal aim has been to lay the groundwork for a new and unexplored area in the field of computer design. While conventional design automation systems provide the user with a convenient method of describing his design, we explore the possibility of giving the user the ability of describing the desired computer architecture along with the available building blocks with which it is to be implemented.

This leads to an investigation of the relationships between the architecture of a computing system and various hardware designs which implement that architecture. By developing a model of a central processor and a language for describing the architecture, we can study the manner in which the architecture influences the data path design and develop algorithms that will yield an optimal data path for a specified performance or cost.

The three inputs of this model are:

1) the architecture which describes the computer as the programmer sees it,
2) the algorithm library which gives a set of possible translations of the operators used in the architecture description language,
3) the hardware library which gives a set of hardware units to be used as the building blocks in constructing a data path.

These two libraries allow a large number of data paths to be constructed which implement the given architecture. The problem is to select the one data path that gives the lowest weighted average instruction execution time but which does not exceed a specified cost limit.

In order to evaluate a particular data path and architecture implementation, we need to know: the cost of the data path, the number of cycles required for each instruction, and the duration of a data path cycle.

The cost of the data path is taken to be the total cost of the individual units. The computation of the minimum number of cycles per instruction and the duration of each cycle are rather complex computations, but the techniques to compute these values have been developed and are presented in this dissertation.

Obviously a problem of this scope implies serious computational difficulties if a working implementation is to be developed. In constructing a system of computer programs to illustrate the potential of this research, we have chosen to assume independence among a few key parameters in order to compute estimates of the performance and cost of various subportions of data paths. With these assumptions we can no longer guarantee an absolute optimal solution, but we find instead an optimal in a reduced solution space. The assumptions enable us to use our programs to find solutions for some substantial architecture descriptions in reasonable execution times. A number of examples are presented which illustrate both the model of a computer and the programming implementation we have developed.

2.4.2 Optimal Selection of Functional Components for Microprogrammable Central Processing Units

The introduction and the wide acceptance of both microprogram control and medium to large scale integrated circuitry have drastically altered the manner in which central processing units are implemented. The use of microprogram control and the tendency to acquire hardware in functional units allows a microlevel hardware software tradeoff to occur during the design of the central processing unit. The combined effect of microprogram control and the availability of powerful functional units has been to alter the CPU design environment. The overall goal of this research is to develop a design philosophy which is applicable to the new CPU design environment. The work proceeds in four steps: 1) the development of the general design method; 2) the initial development of the Program Type Language; 3) the development of the CPU model; and 4) the development of an efficient optimization procedure.

The Program Type Language is an initial version of a machine independent intermediate language. The language is designed to meet two principle objectives. First, the language allows valid statistics to be gathered on the functional requirements of the computer user. Each program type is selected to provide a basic function which is a major building block of a highly used computer area. Second, the language is machine independent and it meets the requirements of an UNCOL language.

The CPU is modeled as a base hardware set plus a set of hardware options. The base hardware set defines a general architecture and the hardware options allow major design variations about the architecture. The hardware options consist mainly of functional units and the use of the hardware option concept in the model is desirable in view of the rapid acceptance of large and medium scale integration. The cost of a hardware option is dependent upon the choice of the other hardware options as is the number of microbits required to control each hardware option. The specification of microinstruction word size is very flexible, allowing both field encoding and word encoding to be modeled.

The design method is user directed. The user writes or translates his programs into PTL. Since PTL is constructed to allow valid statistics to be gathered on user requirements, PTL provides a mechanism for determining user requirements. The design method makes optimum use of the human design. The connection between the program types and the CPU model is the implementation methods. There is one or more implementation method for each program type. Each implementation method specifies a CPU configuration. The human designer specifies the implementation method and is concerned with performance - cost (hardware) tradeoffs at the program type level. The optimization program performs the performance - cost tradeoffs for the entire set of program types using the individual implementation methods supplied by the designer. The designer does not have to attempt to evaluate the combined effort of adding an optional piece of hardware on 100 or more program types. He can focus his attention on each program type one by one.

The optimization technique involves three distinct parts. First, the use of bounding techniques allows the search procedure to be applied to the hardware option space (H space $< 10^6$) instead of the implementation space (I space $> 10^{12}$). Second, a modification of the Lawler-Bell technique is combined with the bounding procedure to eliminate large sections of the hardware option space. Third, a slope selection technique was developed to select the optimal implementation set for any fixed hardware option set.

APPENDIX A

Reprint from Proceedings of Ninth Annual Design Automation Workshop, June 26-28, 1972.

OPTIMAL DESIGN OF CENTRAL PROCESSOR DATA PATHS

Keki B. Irani
The University of Michigan

George A. McClain
International Business Machines Company
Poughkeepsie, New York

I. Introduction

The research described in this paper deals with the design automation of the central processor of a digital computing system. The principal aim has been to lay the groundwork for a new and unexplored area in the field of computer design. While conventional design automation systems provide the user with a convenient method of describing his design, we explore the possibility of giving the user the ability of describing the desired computer architecture along with the available building blocks with which it is to be implemented.

We feel that in even the most advanced design automation systems that are described in the literature, the assumption is made that the user of the system unambiguously specifies the organization of the central processor data path. The specification may be in the form of register transfer statements, for instance, but this implies all the required interconnections and the user is well aware of what these will be. In contrast to this we are constructing a tool to aid the user in developing this initial data path organization.

The result of the process is to be a data path flow chart and the instruction set microprogram.

In this paper we will describe the facilities of the model and then discuss a solution technique we have developed. In the statement of the model we will not attempt to include any of the mathematical formalism that underlies it.

II. Formulation of the Problem

Many different hardware designs will appear identical to the programmer if they execute the identical architecture. These designs, however, may differ in their performance and cost. A model of a central processor and a language for describing the architecture allows a study of the manner in which the architecture influences the data path design and development of algorithms that will yield the most appropriate data path for a specified cost. The basic design automation problem that we are considering is

Given:

(1) A system architecture

(2) Performance requirements and a maximum cost limit

(3) An algorithm library

(4) A hardware library

Find:

(1) A unique, detailed data path

(2) The microprogram to control the data path for the instruction set.

A. System Architecture

The system architecture specifies precisely how the computer is to appear to the programmer. It is composed of a definition of facilities and the list of instructions the computer is to perform. The facilities consist of input and output busses, with which the computer interacts with its environment, and a set of registers. The architecture specifies a width in bits for each of these.

The most important part of the architecture is the instruction set which is a set of language statement groups that define the data transformations accomplished by each instruction. There is one group for each instruction. These definitions refer to the facilities possessed by the computer and specify the desired contents of each register and output port after execution of the instruction as a function of the input ports and the initial contents of the registers. It is important that the set of statements that define an instruction be complete and unambiguous, but it should not specify a particular algorithm. This can be accomplished by using a rich set of operators for defining architecture instructions and by providing a means to distinguish those portions of an instruction definition which must have a particular execution sequence from those which can be more freely reordered. That is, we recognize that the user will want to specify a particular ordering for some statements in the instruction definitions, but we want to be sure we do not require him to order statements which he knows need not have a particular ordering.

Examples of this would be the use of one symbol to disignate multiply rather than defining the operation as a sequence of adds, shifts, and subtracts since the latter definition would greatly limit the choice of algorithms and data path configurations. Similarly, for the loading of two registers, the user should have freedom to say that it is immaterial which of these is done first.

B. Performance Requirements and Maximum Cost

There are two parts to the specification of performance requirements:

(1) A list of weighting factors for the instructions in the architecture.

(2) A maximum permitted execution time for each instruction.

The fundamental means of evaluating the performance of a data path will be to compute a weighted average instruction execution time for the instruction set. A single value of performance will be found by:

(1) Multiplying the minimum number of cycles for each instruction (on the particular data path) by the weighting factor for that instruction.

(2) Summing these values over the instruction set.

(3) Multiplying this sum by the cycle time of the data path.

The weighting factors used in this calculation are approximations to the frequency of occurrence of the instructions in the expected program environment of the computer. However, an instruction mix is only an estimate of processor usage and is generally derived from a statistical analysis of representative programs. For instructions of very low usage these figures are of questionable validity due to the limited data available and uncertainty over the representative nature of the programs chosen for analysis. Therefore we include a limit on the execution time of the instructions, as part of the performance requirements, which can be used to insure that a reasonable implementation is chosen for instructions of low usage.

The maximum system cost is simply a limit to the cost of the total data path as obtained by summing the costs of the individual data path units.

C. Algorithm Library

A unique feature of this approach to design automation is the incorporation of an algorithm library and a "compilation" or "translation" phase of the solution. The language used to define the architecture instructions uses one set of operators while the definition of the hardware functions uses a different set of operators. The architecture is then compiled by selecting entries from the algorithm library which allow the architecture instructions to be expanded in terms of hardware operations.

The principal advantage of this approach is that by including many algorithms in the library to interpret each architecture operator, a wide range of specific implementations can be investigated. Thus, an architecture operation such as multiply can be implemented in hardware by a number of specific adding, subtracting, and shifting algorithms. But in addition we find there is no requirement to associate some understandable meaning with each operator in order to accomplish the compilation. The meaning is thoroughly and completely defined by the algorithm entries for the operators. Thus, to say some architecture operator o_1 is implemented by doing an o'_1 or two sequential o'_2 hardware operations is to define completely the o_1 architecture operator. There is no need for the optimization strategy to be concerned with what the o_1 architecture operator actually does to data. The development of the optimizing strategies can therefore be carried out in a very general manner and kept independent of any fixed set of operators.

The algorithm library itself is treated as being open-ended in two senses. First, new operators may be added to the set of architecture operators simply by adding new entires to the library to translate them. Second, some particular architecture operator can have a new algorithm added by making a simple addition of the algorithm in the appropriate location of the library. Furthermore, the addition of a new hardware operator implies only the modification or addition of algorithms to take advantage of the new hardware operator. In all of these instances the optimization strategy is not affected by these changes to the library. The effect is only to allow more flexibility in the definition of new architectures or the discovery of better data paths by exploring the new algorithms.

D. Hardware Library

As the algorithm library is a catalogue of different translations of the operators of architecture definitions, so the hardware library is a catalogue of different hardware units which implement the various operators of the translated or compiled architecture. In principle then, when we have a translated architecture, we can go to the hardware library and select a set of units which covers the set of operators required by the instruction set of the compiled architecture.

The hardware library is defined as a set of unit specifications where the entry for each unit includes:

(1) A description of the unit facilities.

(2) The unit cost.

(3) The function set of the unit.

The facilities of the unit can be input ports, output ports, and internal registers, and each of these has a specified width measured in bits.

The cost simply is the total cost for the unit.

Ѓhe most important part of a unit entry in the library is the set of language statement groups, called functions, which specify precisely the data transformation that can be performed by the unit. Each of these functions gives a definition of the contents of the output ports and internal registers as a function of the initial register contents and input ports using only operators from the set of hardware definition operators. Since we place no limit on the size of the function set or the complexity of a function, a unit may perform many different and intricate transformations using any of the hardware description operators. There is virtually no limit to the complexity of a unit described in the hardware library. Each function also has time delay information associated with it which gives the total time for that function to be performed.

Again, as in the algorithm library, the hardware library is doubly open-ended. First, a new unit can be added to the library which has some functions to implement operators not previously covered by library entries. Second, a new unit may be added which performs operators already covered by existing units but perhaps has different cost and delay figures. These changes to the library will perhaps lead to an improved solution to some particular architecture by expanding the set of possible solutions. However, they cause no change in the optimizing strategies and would require no change in a computer implementation of this model.

E. Data Paths and Microprograms

The result of the design and optimizing routines consists of a description of a data path and a set of sequencing charts or microprograms which specify the manner of implementing each architecture instruction on the data path.

The description of the data path is given as:

(1) A list of the hardware units selected from the hardware library.

(2) A description of the connections between the input and output ports of the hardware units.

The implementation of the instruction set is a cycle by cycle description of the operations of the data path for each instruction. This is essentially a microprogram for the architecture on the data path.

For each cycle this specifies:

(1) The connecting busses between unit ports which are being used and the bits of these busses which are being gated.

(2) The function that each unit is performing.

(3) The names of operands present in the data path at the end of each cycle and the registers in which they are stored.

In order to evaluate a particular data path and architecture implementation, we need to know:

(1) The cost of the data path.

(2) The number of cycles required for each instruction.

(3) The duration of a data path cycle.

The cost of the data path is taken to be the total cost of the individual units. Each unit cost is part of the hardware library entry for the unit. This computation is straightforward and the requirement for a valid data path is that this figure be less than the value included in the input data as the maximum permitted cost. We should observe, however, that in this model no allowance is made for the cost of the interconnecting busses nor for the cost of whatever hardware is required to control the data path. This latter cost would normally cover status and sequencing triggers, gate control triggers, clocking circuitry, and a control memory if the data path employed microprogram control.

F. Variables in the Problem Solution

The parameters to be considered in deriving a data path for a given architecture are:

(1) Selection of algorithms.

(2) Selection of hardware units.

(3) Selection of a data path bus configuration.

(4) Determination of the data path cycle time.

(5) Determination of the microprogram.

We will discuss here the difficulty of making these decisions in a general solution. Later we will consider a solution technique which reduces the complexity of these computations.

The selection of algorithms is basically a simple covering problem. We have to explore the different possible covers, but the number of these for a reasonable architecture and algorithm library tends to be very large and it is difficult to establish any convenient objective function which can be used to choose among them or search through them. Any one algorithm defines an architecture operator in terms of other architecture or hardware operators. Therefore, in order to translate some architecture operator into terms of hardware operators only, it may be necessary to apply a sequence of algorithms.

The selection of the hardware unit is also a simple covering problem, but this is complicated by a lack of structure among the various library entries. For instance, we have chosen not to require any monotonic relationships between the costs or delays of units which execute the same operator. Hence, we have little basis for predicting the characteristics of different covering sets based on an analysis of the operators or the evaluation of some similar covering sets. Each set requires an involved performance evaluation. Another complicating factor is the ability for each unit to perform many functions in serial or parallel order. We would nevertheless like to examine all sets of units with particular attention to the comparison between simple but quick units requiring a number of passes to achieve the desired transformation and complex but slow units capable of performing involved functions in a single pass.

For a given set of data path units an enormous number of different interconnecting bus patterns is usually possible. The search for an optimal bussing configuration to maximize the performance of a unit set can be organized for an efficient search of the possibilities, but unfortunately we are generally handicapped by the exceptional number to be examined. As with the unit set selection, an important characteristic of different bussing configurations that we want to explore is a comparison of short, simple data path cycles to long, complex cycles.

The evaluation of the data path cycle time from the individual unit delays can be formulated as a "longest path" problem, but a reasonable model for a computer does not lead to a longest path problem having an efficient solution.

Finally the computation of a microprogram to implement the compiled architecture is itself a major undertaking. The two most important facets of this are the assignment of operands to registers and the sequential ordering of the statements in each instruction. The ordering of these statements is deliberately left vague in the architecture definition to provide flexibility, and it is in the computation of the microprogram that use is made of this freedom. We must look at many different assignments or groupings of statements into instruction cycles to find the one which yields the smallest number of cycles.

G. Data Path Evaluation

The figure of merit of a particular data path design is the weighted average instruction execution time, WAIET. This can be defined as:

$$\text{WAIET} = T \cdot \sum_{i=1}^{N} \phi_i \cdot n_i$$

where

N = number of instructions

T = data path cycle time

n_i = number of cycles for instruction i

ϕ_i = weighting factor for instruction i

Of all the possible compilations of an architecture and all the possible data paths assembled from hardware library units we would like to find the one whose total cost is less than the specified cost limit and which minimizes WAIET.

H. A Simple Example

For a simple example of the general processes just described, consider an architecture with one instruction which is defined by the single architecture operator, o_1. We might write this instruction as:

$$R2 \leftarrow o_1 (R1)$$

where R1 and R2 are the two architecture registers. Figure 1 is a graphic representation of this instruction. Let us assume the algorithm library has an entry that defines o_1 in terms of hardware operators o'_2 and o'_3 as shown in Figure 2. Then we compile the instruction by applying this algorithm to obtain the result depicted in Figure 3.

Going to the hardware library we find a unit, u_1, shown in Figure 4 which has functions for both o'_2 and o'_3. That is, u_1 performs two functions, one of which is o'_2 and the other o'_3. In Figure 5 we have a possible data path constructed with two register units and u_1. An optimal microprogramming of the instruction is quite obvious. There are two possibilities, but both require two data path cycles. We have arbitrarily chosen one of these and listed the data path cycles in Figure 6. This represents an implementation of the instruction on the data path.

This example illustrates the basics of compilation and microprogramming. With just one algorithm and one hardware unit we do not see the process of algorithm and hardware unit selection, and more importantly we have not mentioned the techniques that enable the various units to be assembled and connected into a data path.

III. Development of One Solution

A completely general solution to this problem seems at present to be too complex to be implemented in a computer solution. Since we were interested in developing a working, prototype system, we investigated a number of possible approaches which would simplify the computation necessary to generate data paths from an architecture description.

We eventually selected one particular method of implementing the model and developed techniques that would enable us to solve the optimization problem defined in the last section. This required making compromises such as simplifying certain features of the model and accepting a suboptimal solution for certain problem variables, but as part of the research we then developed a complete programming system to implement this method of solution. One consideration in selecting this particular solution approach was to achieve a balance between the execution time of the programming system on nontrivial examples and fidelity to the complete mathematical model in order to demonstrate its capabilities.

The approach we chose to pursue is based on the assumption that we can compute estimated performances for algorithms using the available hardware units and then apply these estimates to the solution of a particular architecture. Using this approach we have developed a system that can process relatively elaborate architectures and generate good designs without excessive computer execution time.

Basically the procedure followed is to manipulate the algorithm and hardware libraries to construct a table which associates a delay value and a large set of compilations and hardware realizations with each architecture operator. The table, called a GH Table, is constructed giving the complete set of algorithms and possible data path unit sets for implementing each architecture operator and the execution time of the operator on that unit set. For a particular operator the table can have entries for every expansion of the algorithms and every possible hardware implementation of the expanded algorithm.

When we are given a particular architecture, we can use the delay values for each operator in this table to select both a set of algorithms to compile the architecture and a set of hardware library units for the data path. That is, for each required architecture operator we pick one entry from among the set of entries for that operator based on the frequency of usage of the operators in the particular architecture. We then need only maximize the performance for this compilation and unit set without iteratively changing either of these.

There are two basic assumptions implied in this approach which limit the solution space that is explored. First, it assumes that the algorithms can be evaluated independently. During the selection process for a particular architecture we do not take into account the possibility of parallel execution of various operators in an instruction. In a truly optimal data path the execution of some operators might be overlapped. The implementation of each operator will be dependent on the other operators required by the instruction and the execution sequence imposed by the partial ordering of the statements in the instruction statement group. But we do not consider this at the time. We make a selection from the GH Table (although we do consider parallel execution later during performance evaluation). The second assumption on which this method is based is that we need choose only one entry for each architecture operator. We will compile every occurrence of an operator in an architecture in the same manner. We make a selection based on the overall importance of each operator (a weighted average computation is used) in the architecture as a whole and hence arrive at a single choice for compiling every occurrence of an operator.

The programming system that implements this approach falls into two separate parts: (1) the generation of the GH Table for a given algorithm and hardware library, and (2) the generation and evaluation of a data path for a given architecture.

A. GH Table Generation

For each architecture operator there is a group of GH Table entries. Each entry consists of:

(1) A delay value

(2) A sequence of algorithms

(3) A set of units from the hardware library.

The delay values and the unit set are used later to select entries from the GH Table for a particular architecture. The algorithm sequence describes the compilation process used in generating the GH Table entry from the initial architecture operator. After entries have been selected for a particular architecture, the algorithm sequences for each selected entry allow the architecture instructions to be compiled in a manner compatible with the unit set that has been selected.

The procedure for constructing one entry for the table is simply to enumerate one of the possible translations of an algorithm for an architecture operator that leads to a statement of the algorithm in terms of hardware operators only. Then select one of the possible coverings of the required hardware operators by hardware units.

The result of the compilation of an algorithm can be represented as a noncyclic, directed graph. Each node corresponds to an operator and the edges correspond to transfers of operands. When the delays to perform each operation are associated with each node according to the covering of the operations by the selected unit set, then the computation of execution time is simply the problem of finding the longest path through the graph. This value of longest path delay is the figure of merit associated with the particular GH Table entry.

The GH Table must be essentially a complete enumeration of the possibilities for each operator because it is not possible to choose here between two different implementations of an operator on the basis of their individual cost and performance. This can be done only by considering the whole set of operators required by some architecture. To illustrate this point consider an architecture operator that can be implemented by both units b and c, where c is the more costly. Now, if they have identical execution times, we might be tempted to omit the unit c implementation from the GH Table. But some specific architecture may require both this operator and a second operator which can only be performed by unit c. Had we omitted the entry for unit c for the first operator, we would now be forced to add unit b to the data path to implement the first operator rather than take advantage of unit c which must be included to cover the second operator.

B. Data Path Generation

There are three principal steps in the process of generating a data path for a specified system architecture after the GH Table has been computed. First, the GH Table is used to select entries for the architecture operators used in the given architecture. This process yields both a unit set for the data path and an algorithm sequence for compiling each architecture operator. Next, the architecture is compiled using these algorithm sequences. Finally the compiled architecture is implemented and evaluated on the unit set.

This process can be considered one complete pass through the data path generation program. However, there is a complicating factor. We have not discovered any procedure to predict the optimal number of registers a data path should have. No registers were considered in the generation of GH Table entries and the unit sets associated with the entries in the table do not include registers. How do we determine the number of registers to be included in a data path?

We obviously need at least one hardware register for each architecture register, but there will usually be an advantage to including some temporary storage registers for intermediate results during an instruction execution. Without a cost limit we would simply put in a very large number of registers. Additional registers cannot cause a loss in performance because our model does not consider fan-in and fan-out limitations, the cost of busses, or increased bus delays that would, in reality, result from increasing the number of registers. But since there is a total system cost limit, unnecessary registers may reduce the number of transformational units the data path can contain and hence lead to a suboptimal data path. Starting from the other extreme of including too few registers, in order to be able to afford more transformational units within the cost limit, may result in some units being unusable because there is no available temporary storage for their results. This again leads to a suboptimal data path. The proper number will usually represent a balancing of these two, opposing considerations.

Our research has not produced a satisfactory solution to this problem. So we will try different numbers of registers and compare the data paths and performance for each. The process just described then becomes one independent computation for a fixed number of registers, and the entire process is simply repeated with an increasing number of registers. We start with the number of registers in the architecture and add one for each iteration until either we find that an additional register has not improved the performance or we have reached some arbitrary upper bound.

The objective of the first portion of the data path generation program (selection of GH Table entries) can be outlined as follows:

Given:

(1) The system cost limit for the architecture

(2) The number of registers in the data path

(3) The frequency of occurrence of the architecture operators in each instruction

(4) The weighting factor of each instruction

Find:

(1) The number of copies of each unit that the data path should have

(2) An entry in the GH Table for each operator that occurs in the architecture

Subject to the constraints that:

(1) The cost of the selected unit set, including registers, does not exceed the systems cost limit

(2) The estimated delay (from a weighted average computation) for the selected unit set and GH Table entries based on the entry delay values, is minimized over all possible unit sets and table entry selections.

The second portion of the program uses the algorithms associated with the entries selected from the GH Table to compile the architecture. Recall that there is an algorithm sequence associated with each GH Table entry that describes how that operator has been compiled for that specific entry.

Now this algorithm sequence is used to compile every occurrence of that operator in the architecture in exactly that same manner. This is an important step in the programming system; no other compilation procedure can be used. If the architecture were compiled by algorithm sequences other than those used in forming the selected entries, then the compiled architecture could require operators not coverable by the selected unit set.

The compilation phase of this programming system is very straightforward. Each operator in the architecture is treated independently, and the program simply proceeds by compiling the first statement of the first instruction, then the second statement of the first statement and so forth until the last statement of the last instruction has been done.

At this point in the generation of a data path we have determined both the compilation of the architecture and the unit set for the data path. For the third step we need to determine the optimal interconnection of the units, the cycle time, and the number of cycles per instruction. Because we chose to make rather an accurate measure of performance, this component of the programming system is rather complex. We will not attempt to describe this procedure in detail.

The only solution variable remaining unassigned at this point is the set of interconnecting busses. We have mentioned previously that since we have no cost associated with the busses, the only detrimental effect of having an extra bus is that it may contribute to an unnecessarily long cycle time. So the basic approach in optimizing the bus configuration is to start with essentially all possible busses and compute a performance; then see if an improvement can be obtained by temporarily deleting any bus that is part of the longest path. (Busses unused by the microprogram are not included.) If that is possible, we make the deletion of that bus permanent. For the new data path we can again compute a performance and a longest path and repeat the above procedure. In this program we choose to assume we have an optimal bus configuration when no improvement results from deleting any bus in the longest path. We recognize that will be only an approximation to the optimal data path since it is possible that in some cases deleting two or more busses will yield improved performance even though deleting any one bus may not.

IV. An Example

After the programming system was completed, it was used in an undergraduate course in computer organization at The University of Michigan. Students have been able to use these programs to "design" a DEC PDP-8 computer. This means, of course, they described a computer architecture based roughly on the PDP-8 architecture but with a number of simplifications required by limitations of the programming system. Then they defined a set of appropriate algorithm and hardware libraries. The particular data path which is generated by this program, even for fixed algorithm and hardware libraries, is dependent on the cost and performance specifications and may range from a mini-computer to a super-computer in terms of the number of hardware units included in the data path.

The actual input and output to the program for this example is too large to be included here, but we can summarize the characteristics of one design that was done in this course:

Architecture:

```
 4 registers
 3 input ports
 3 output ports
10 instructions (averaging 7 statements per instruction)
```

Algorithm Library:

```
10 architecture operators
 1 algorithm per architecture operator
```

Hardware Library:

```
10 hardware operators
10 unit types
```

The ten instructions (using DEC mneumonics) are: TAD;AND;DCA; JMP;JMS;ISZ; and four instructions to define some of the Group 1 micro-instructions, Group 2 microinstructions, teletype reader instructions, and teletype printer instructions.

A typical instruction in this architecture (AND) is written:

```
MB(1,12):FTCH(R3(1,12))               00000
R3(1,12):ADDD(R3(1,12),R4(1,12))      10000
XX(1,12):ADRR(MB(1,12))               10000
W(1,12):READ(XX(1,12))                10100
R1(1,12):ANDD(W(1,12),R1(1,12))       10110
```

R_1 through R3 are names of architecture registers, and MB, XX, and W are temporary names of operands. R4 is a register whose contents are always one (one method of handling literals in this program). In these statements the value computed to the right of the colon replaces the operand whose name appears to the left of the colon.

The first statement specifies instruction fetching and decoding; the second is instruction counter incrementing; the third computes the operand address; the fourth reads the operand from memory; and the last AND's it with the accumulator.

The binary number to the right of each statement defines the execution sequence of the statement group. This is done by associating the i^{th} bit of each number with the i^{th} statement in the list. If the binary number of the j^{th} statement has a one in the i^{th} position, this means that statement i must precede statement j. If it has a zero there, then statement i does not have to be executed before statement j. In this example this flexibility is used to specify that instruction counter updating can be done at any time during the instruction execution. (All the binary numbers have the second bit equal to zero.)

A typical algorithm to define one of the arcnitecture operators (FTCH) is:

```
    MB(1,12):FTCH(PC(1,12)
    X(1,12):ZREA(PC(1,12)) 00
    MB(1,12):OPDC(X(1,12)) 10
```

The structure here is analgous to a programming MACRO where the first statement is a model for the usage of FTCH in an architecture instruction, and the second two statements give a translation in terms of hardware operators (ZREA and OPDC). This algorithm would be used to compile the first statement of the AND instruction where the names MB and R3 from the instruction statement would be substituted into the algorithm statements to replace the dummy variable names MB and PC there.

After all statements have been compiled, this instruction becomes:

```
T1(1,12):ZREA(R3(1,12))                0000000000
MB(1,12):OPDC(T1(1,12))                1000000000
R3(1,12):ZADD(R3(1,12),R4(1,12))       1100000000
T2(1,12):ZREA(MB(1,12))                1100000000
T3(1,12):ZSAN(MB(1,12),T2(1,12))       1101000000
T4(1,12):ZTZE(T3(1,12))                1101100000
T5(1,12):ZSAN(T4(12,1),MB(1,12))       1101110000
XX(1,12):ZADD(T5(1,12),T3(1,12))       1101111000
W(1,12):ZREA(XX(1,12))                 1101111100
R1(1,12):ZAND(W(1,12),R1(1,12))        1101111110
```

Finally a typical hardware unit (MEMORY) is described as:

```
Cost = 300
Delay = 15
Number of Functions = 2
  P1:ZREA(Q1)
  : ZWRT(Q1,Q2)
```

The two statements define the two separate functions which the unit can perform (only one function can be selected for a single data path cycle). Q1 is the address input port; Q2 is the data input port; and P1 is the data output port.

One data path generated by the programming system is illustrated in Figure 7. The program has included three extra registers for temporary results. Because there is no cost associated with the busses, we find a larger than usual number have been included by the program. Also, we do not distinguish which port is used on units having more than one input port.

We can assume all busses to a unit go to all input ports since we have no fan-in or fan-out limitations. The execution time for this example was about three minutes (CPU time) on an IBM System 360 Model 67.

V. Conclusion

The most important objective of this research has been to develop a model of a central processor and of a computer architecture in order to formalize and investigate the way in which the architecture influences the processor organization. The particular implementation and solution technique we developed is only one of a great many possible interpretations of the model. We were forced to simplify or omit certain features of the model and to accept approximate solutions instead of a true optimal result for some computations in the program. These trade offs were necessary in order to allow a reasonable solution time for nontrivial problems.

This paper is based on research that was supported by the Rome Air Development Center, Research and Technology Division, Griffiss Air Force Base, New York under Contract No. F30602-69-C-0214. The complete report is available from The Systems Engineering Laboratory, The University of Michigan, Ann Arbor, Michigan as Optimal Design of Central Processor Data Paths, SEL Technical Report No. 58, May 1972.

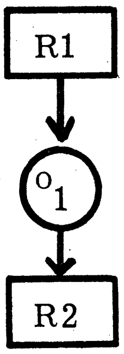

Figure 1. The Instruction Graph.

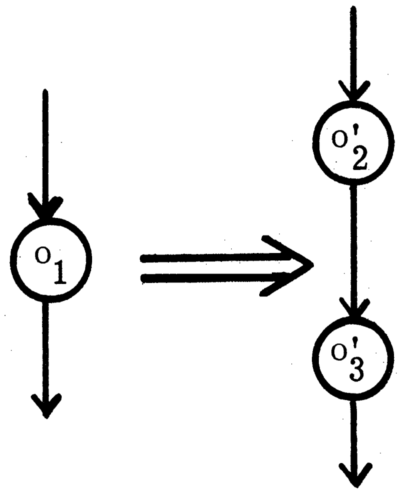

Figure 2. An Algorithm for o_1.

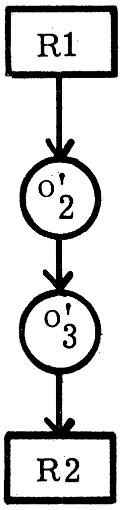

Figure 3. The Compiled Instruction.

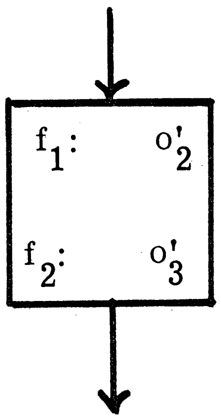

Figure 4. A Hardware Unit, u_1.

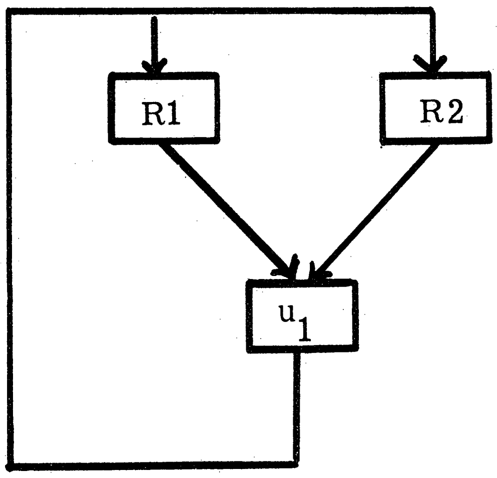

Figure 5. The Data Path.

cycle 1	$R1 \rightarrow u_1 \rightarrow R2$	u_1 does f_1
cycle 2	$R2 \rightarrow u_1 \rightarrow R2$	u_1 does f_2

Figure 6. The Microprogram.

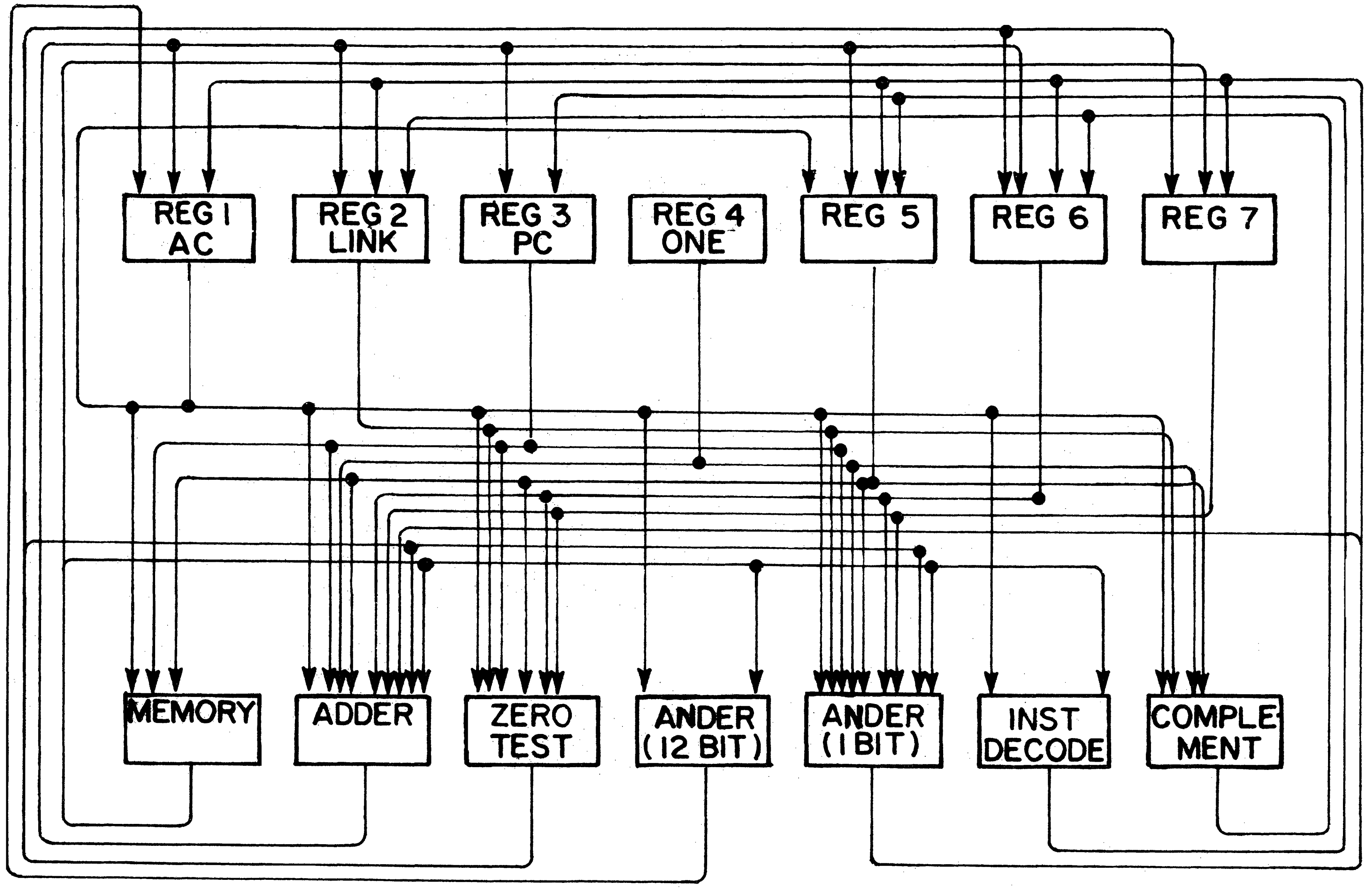

Figure 7. An Example Data Path

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* University of Michigan Dept. of Electrical Engineering, Systems Eng. Lab., Ann Arbor, Michigan 48104	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP N/A

3. REPORT TITLE

A STUDY OF INFORMATION IN MULTIPLE-COMPUTER AND MULTIPLE-CONSOLE DATA PROCESSING SYSTEMS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report Apr 69 - Jun 72

5. AUTHOR(S) *(First name, middle initial, last name)*

K.B. Irani, I.S. Uppal, G.A. McClain, D.L. Hinshaw

6. REPORT DATE October 1972	7a. TOTAL NO. OF PAGES 50	7b. NO. OF REFS 12

8a. CONTRACT OR GRANT NO. F30602-69-C-0214 Job Order No. 55810006	9a. ORIGINATOR'S REPORT NUMBER(S) Annual Report No. 5 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* RADC-TR-72-250

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES RADC Project Officer: Rocco F. Iuorno (ISIM) AC 315 330-7011	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center ISIM) Griffiss Air Force Base, New York 13440

13. ABSTRACT

This final report summarizes the achievements from April 1969 to June 1972 of continuing research into the development and application of mathematical techniques for the analysis and optimization of multiple-computer, multiple-user systems.

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Mathematical Models						
Multiprogramming						
Paging Algorithms						
Data Base Structures'						
Microprogramming						
Central Processor Design						
Design Automation						

SAC--Griffiss AFB NY

UNCLASSIFIED
Security Classification