

To the Graduate Council:

I am submitting herewith a dissertation written by Monte Lee Meals entitled "Microprocessor Applications in a Distributed Energy Management System." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Chemical Engineering.

Charles F. Moore
Charles F. Moore, Major Professor

We have read this dissertation
and recommend its acceptance:

Walter L. Dineen

John Douglas Birdwell

E. E. Stansbury

Duane D. Burns

Accepted for the Council:

C. W. Mink
The Graduate School

MICROPROCESSOR APPLICATIONS IN A DISTRIBUTED
ENERGY MANAGEMENT SYSTEM

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Monte Lee Meals
December 1985

ACKNOWLEDGMENTS

Deepest appreciation is extended to my parents and family for their moral encouragement and spiritual assistance during my academic career. The author wishes to extend appreciation to the following:

Dr. Charles F. Moore, my advisor, who initiated this project and gave invaluable assistance and recommendations without which the completion of this study would have been impossible.

Ray Gaddis for implementing the computer, the computer interface, and teaching me the real world aspects of computing.

Elizabeth Burnett for her unending aid in keeping the world in focus and proving that for every successful man there is a successful woman, by his side.

ABSTRACT

The purpose of this study is to develop a microprocessor based controller for applications in a distributed energy management system. The controller is applied to two energy devices, an industrial boiler and a chiller.

The hardware/software system that developed is quite general in nature and is discussed in the text with emphasis on essential functions. The appendices contain a listing of a macro type control language that also developed with the two applications as examples.

The two applications are an industrial boiler and a chiller. The control strategies discussed in the text are aimed at reducing energy consumption on a local basis while allowing overall actions from a higher level control system for plantwide energy reduction.

The results of the work lie in two areas. The first is application oriented. Control systems were developed which reduced the energy consumption by a significant amount relative to environmental conditions. The hardware/software system that was developed is quite general in nature and has been applied to several other applications.

The second area deals with design philosophy. Throughout this work a common design theme was followed: The structure of the control system is defined by the structure of the process. This theme provided insight into not only the structure of the control strategy but also the implementation of process control elements.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
2. HARDWARE	15
3. SOFTWARE ORGANIZATION.	26
4. STEAM DRIVEN CHILLER CONTROLLER.	69
5. OIL FIRED BOILER CONTROLLER.	94
6. SUMMARY AND CONCLUSION	117
LIST OF REFERENCES	119
APPENDICES	126
A. CHILLER SOFTWARE	127
B. BOILER SOFTWARE	145
C. MACRO MAP\$	182
D. MACRO RTMO\$\$	189
E. MACRO DATA\$\$	201
F. MACRO VALV\$\$	249
G. MACRO PICO\$\$	254
H. MACRO SCAN\$\$	258
I. MACRO PARI\$\$	265
J. MACRO PARO\$\$	271
K. MACRO STASK\$	274
L. MACRO TASK\$	276
M. MACRO LELA\$	278
VITA	281

LIST OF FIGURES

FIGURE	PAGE
2-1. Digital control hardware organization	18
2-2. Digital controller design	19
2-3. Communication network linkage	23
2-4. Communication network configuration	24
3-1. The cascade hierarchy structure	27
3-2. The cascade hierarchy structure applied to software	29
3-3. Control table format.	32
3-4. Software organization	35
3-5. Steam chiller temperature control and optimization strategy	39
3-6. Software implementation of chiller temperature control and optimization strategy	40
3-7. Software implementation execution cycles.	42
3-8. Real time task execution	46
3-9. Proportional plus integral (PI) controller.	52
3-10. Valve controller flow diagram.. . . .	60
4-1. Steam driven chiller.	70
4-2. Initial chiller control	72
4-3. Chiller variables	75
4-4. Chilled water temperature	79
4-5. Refrigerant temperature	80
4-6. Chilled water temperature control	82
4-7. Refrigerant temperature control	83
4-8. Chiller load control.	85

FIGURE	PAGE
4-9. Compressor surge line	87
4-10. Control strategy for compressor surge	88
4-11. Valve position control.	90
4-12. Flow and speed control.	91
4-13. Complete chiller control.	92
5-1. Boiler.	95
5-2. Initial boiler control.	97
5-3. Combustion control.	101
5-4. Air lead.	105
5-5. Master-slave pressure control	107
5-6. Multiple boiler pressure control.	108
5-7. Steam drum level control.	110
5-8. Air temperature control	112
5-9. Firebox pressure control.	114

CHAPTER 1

INTRODUCTION

The work described in this dissertation covers the development and implementation of the hardware, software, and strategy for two controllers in an existing energy management system. The control systems were needed for an industrial boiler and chiller. The controllers were intended to reduce energy consumption by providing more efficient operation locally and allowing overall control from a higher level through a communication highway.

These controllers were implemented with a Digital Equipment Corporation LSI 11-based system. Hardware development included interface design to the process, the local operator, and a data highway.

This dissertation describes some of the design features which may help to develop a microprocessor-based control system. Some of the specific areas include:

1. A complete control strategy for a medium-sized boiler with a wide range of operation loading.
2. A complete control strategy for an industrial chiller.
3. A method of classifying process variables with implications for developing a control strategy for those variables.
4. A division by function of the software required for a digital control system. A macro-based control language is included as an example.

5. A detailed description of the control algorithms is also included. Emphasis is placed on features not reported in the literature but necessary or beneficial for implementation.
6. The structure of a minimal real-time monitor is described.
7. Programming conventions which simplify the software structure are also described. These conventions include the importance of table driven, re-entrant code and the asynchronous nature of control functions.

Much work has been done with computers in digital process control from both strategic and implementation aspects (1,2,3,4,5). Prior to the introduction of microprocessors, economic constraints limited this work to specialized applications. Traditionally, control applications have been an economic trade-off. As a result, the cost of owning and operating computers limited the high quality of digital control to critical applications.

Advances in microelectronics have radically improved the economic picture (6,7). Today it is not unreasonable to consider computer (microprocessor) control for even the most simple control problems such as a wall thermostat. A search of some of the literature in the field reveals the following:

Kahne et al. (8) discuss the impact of low cost computers and some of their applications in the steel industry. The desirability of distributed control with overall supervision is presented in their paper. Several different data highway structures are presented and various communication protocols discussed.

Anderson (9) discusses issues similar to Kahne et al. (8) in his paper with emphasis on a one task per microprocessor structure. In addition, he discusses a master communication controller in several network configurations. Cost-performance characteristics are discussed along with an inquiry-response protocol. Possible applications in the machine tool industry are discussed.

In his paper, Mears (10) discusses a distributed system with "over 50 micro and mini computers." The application is in an offshore oil platform. The control modules are discussed in terms of their function. The communication network is arranged in a tiered structure with redundant critical points and a radio link to shore at the highest level for mass storage.

Thor et al. (11) discuss microprocessor applications in a chemical plant. The approach taken is distributed control on a "unit operations basis . . . to reduce energy usage." They analyze where energy savings can be achieved through local control and briefly mention the control functions of the various microprocessors.

Thomson (12) in his paper discusses the application of a microprocessor as a data acquisition device. The particular measurement problems of a two-phase-flow system are discussed and the hardware implementation is presented. A brief discussion of the software structure is included along with a description of the operator control panel.

In his paper, Thomas (13) also discusses the architecture of a 12-bit microprocessor. The instruction set and input-output structure emulate the Digital Equipment Corporation PDP 8 minicomputer.

Applications are discussed as a low-cost replacement for this widely used minicomputer.

Parker (14) discusses a range of microprocessor designs from calculator chips to microprogrammable devices. The capabilities and software requirements for various applications are presented for comparison.

Mallicote (15) in his thesis discusses an application involving power distribution substations. A single master with multiple-remote units is implemented for data acquisition and remote control of the substation equipment.

Hamilton (16) describes the replacement of an industrial regulator by a microprocessor. The requirements of a single-loop industrial regulator are presented along with the final microprocessor design. A performance comparison is made with the analog regulator.

Bryan's (17) thesis describes the use of a microprocessor as an optimal discrete-time regulator. A review of the mathematical basis of the regulator is followed by the development of the dynamic equations for the heater. The implementation of the controller on a dedicated microprocessor is presented.

Monks (18) in his thesis describes an application using microprocessors for the data acquisition and control of a heating, ventilating, and air conditioning system. A master controller-remote data acquisition slave system is described. The remote slaves take temperature readings throughout the building and these readings are transferred to the master controller through a data highway. The master device provides direct digital control of the air handler.

The Energy System

This work is part of an existing program between The University of Tennessee, Knoxville and the IBM Corporation. The program involves the development of a plantwide energy management system (19). The energy system under study is at a multi-acre plant located in Kingston, New York. The plant is a combined office, laboratory, and manufacturing facility having a constant annual cooling and heating demand.

The primary heating media at this plant is medium pressure (150 PSI) steam provided by four 40 klb/hr oil fired boilers. Part of the steam is used to drive four of the seven water chillers. The remaining three chillers are electric. Chilled water (44° F) is the primary cooling media for both space conditioning and laboratory usage. The total cooling load is split between the steam and electric chillers depending on relative operating costs. Two forced-draft, down-flow cooling towers provide final waste heat rejection to the atmosphere for the chillers.

The chilled water and steam generated by the boiler house are used in the facility by the various HVAC units to supply cooling and heating as needed in the plant for comfort and equipment cooling in the various office and research areas. The various types of space conditioners include hot water systems for heating perimeter spaces, bounded by exterior walls, and air handlers. Several varieties of both single and dual duct air handlers provide air circulation, fresh air ventilation and enough hot or cool air for humidity and temperature requirements.

Purpose of the Energy Management System

The initial energy control system was supplied with the original equipment (20,21). The control systems were designed when energy was inexpensive. They were aimed at stable, consistent operation and not necessarily at efficient operation. The purpose of the first level of the energy management system is to develop and implement operation control strategies for each energy device (boilers, chillers, and other heating and ventilating equipment) incorporating, where possible, local optimization. In this dissertation, optimization will refer to minimizing energy consumption for a given load condition. The approach taken to the control problem was simple and direct. An initial study was performed to determine the process variables, their relation to each other, and their effect on energy consumption. An initial strategy was formulated using those relationships. The microprocessor was attached to the process and the strategy was implemented. The results were observed and the strategy was modified until the final goals were met. Thus, stable control systems were generated with inherent energy savings. The purpose of the first level can then be stated as developing individual control systems which perform the same function as the original systems while attempting to minimize energy consumption. Another object of the first level systems is to provide communication handles to a second level control. The second level will achieve additional improvements by coordinating the operation of the local controllers. This improvement will occur in the same manner as conventional supervisory digital control (1) for overall energy reduction. This cascade control organization has been successfully

used widely in the past with analog controllers (22). This work attempts the same while exploiting the advantages of microprocessors for direct digital control.

Energy Management System Implementation

Along with the development of the control and optimization strategies, a primary concern in an energy management system is the microprocessor implementation of the controllers. Many papers (23, 24,25,26,27) have examined and reported the advantages of microprocessors in a distributed control system. The main findings of these papers are summarized below:

1. While the inherent computational power of a small micro-processor does not yet allow implementation of some of the advanced control theory, it does allow implementation of very sophisticated strategies.
2. Low cost per microprocessor allows computational distribution with many microprocessors, each dedicated to a single device.
3. Communication between digital controllers can be as simple as a two-wire data link servicing all microprocessors for higher level control.
4. Each digital controller can be designed with identical hardware. That is, the same hardware can be used on different processes. Individual characterization can be completely contained in software.

Goals of Development

With the stated purposes and means in mind, the goals of this energy management system can be presented.

1. Develop and implement with microprocessors energy conscious control strategies for each energy device (boilers, chillers, HVAC, etc.) in the energy system.
2. Develop and implement a communication network connecting all local controllers to a central computer for overall coordination, load distribution, and supervisory control.
3. Develop and implement higher level control strategies for the central computer.

The work in this dissertation covers the first two of these goals: local controllers for a boiler and a chiller as well as a communication network are developed.

Historical Development of Digital Control Hardware

Digital process control is a rapidly developing field. The first systems date back only to the mid-fifties, and major advances have been made in the last 25 years (28,29,30). Initial computer control systems were in numerical machine control and later high-speed aircraft. The computers were custom built and required teams of programmers.

By the mid-sixties, major advances had been achieved in computer hardware. Costs had been reduced four-fold, reliability had increased ten-fold, and speeds increased 5000-fold. The mid-sixties saw the introduction of the minicomputer, a moderately low-cost general-purpose

machine, well suited for control. The chemical industry, realizing their advantages, began employing minicomputers widely for supervisory control. The overall control of a complex operation was performed by a combination of analog and digital controllers. The primary control elements (valves, motors, etc.) were modulated by analog devices to maintain flows, temperatures, etc. The setpoints for these analog controllers were adjusted by the supervisory digital computer to provide overall coordination. This type of control structure had many advantages and is still employed extensively today.

Direct digital control was also used at this time for critical processes. Minicomputers were connected directly to the process for both high level and control element operation.

Microprocessors were introduced in the mid-seventies. Their cost ranged from hundreds to a few thousands of dollars. Their capabilities equaled that obtained from the largest computers of the mid-fifties, and their reliability increased proportionally to the reduction in total number of components.

Today, microprocessors and the emerging microcomputers are a very active field for hardware development prompted mainly by market demand and growth. This cost-performance ratio of these units has been the basis for rethinking the conventional analog-digital control practices of the previous decade.

Historical Development of Digital Control Software

Computer software for process control has followed a development parallel to microprocessors in this maturing field (31,32,33,34,35).

Of necessity, however, software advances have lagged behind hardware by several years to allow experience in each to accumulate.

In the fifties, because of the custom nature of computers, there were virtually no developed software or software generation tools. Programming was done in machine language. Software for a specific application was conducted by teams and required a large number of man days to complete. By the early sixties, assemblers were available for most machines, allowing a significant reduction in the man days per project. A growing trend developed toward higher level procedural languages, such as Fortran, to allow software transportability.

By the late sixties, as minicomputers were becoming well established, general-purpose real-time monitors were announced by several major computer vendors such as IBM, GE, CDC, and DEC. These monitors did not provide control functions; however, the monitors did offer a framework to build a digital control system.

The sixties also saw the introduction of several languages written specifically for process control. There are two main varieties of these "higher structured" languages, problem oriented languages and fill-in-the-blank systems.

Problem oriented languages such as AUTRAN developed by CDC or BATCH developed by Foxboro had much in common structurally with procedural languages such as Fortran. Their difference involved re-writing the instructions to specify the action of a control system, element by element. A program would be constructed by combining these instructional control elements in a procedural type manner until the strategy was complete. In addition, the languages contained input-

output functions and an operating system to provide a complete package.

The most highly structured programming systems for process control were called (for lack of a better name) "fill-in-the-blank programming systems" such as PROSPRO by IBM and BICEPS by GE. These systems generally consisted of separate preprogrammed subsystems for data acquisition, data management, general control actions, and operator logs. Within these subsystems, the user provided data according to some predefined format. A predefined system used this data to link up the data acquisition section with the process, configure the data base, specify control action, and link up the actual outputs to the process.

Today, several excellent real time monitors for microprocessors have been proposed and are available (36,37,38,39,40). Some of these, such as RT11 by DEC, are scaled-down versions of monitors designed for minicomputers. Others such as RMX/80 were designed specifically for microprocessors in a variety of configurations, some requiring mass storage (disks) and some not.

As in the previous era, the state-of-the-art control software invariably includes a real time monitor as an integral part. A new variety of control languages has appeared, the "interactive" languages. In such a system like Foxnet by Foxboro, Beckman's 8080 system, or SYSG11 by the author, the programming is done interactively with a software development system via a keyboard or graphics terminal. The language elements are, as in the problem oriented languages, procedural in nature but may take the form of an actual control

strategy flow chart on a graphics terminal or a description of the control elements and how they are connected. The interactive nature insures syntactically complete systems and can usually be operated by individuals with little or no programming experience.

Process Control Strategies

A computer and a programming language are not, however, a digital control system until a control strategy is developed and implemented in the system. To complete the system, process type control techniques can be used to develop a strategy for a given energy management control problem. From a historical view, the first well-defined use of feedback control seems to have been Watt's application of the flyball governor to the steam engine in about 1775 (41). The general application of process control began to appear in the late 1920's and the first general theory of automatic control was published in 1932 (42). Much has been published since, and the field has grown into a mature science. The basic theory for discrete (computer) as well as continuous (analog) control is well developed with many complete texts on process control (1,22,41).

In process control applications, the relative speed of response of typical processes compared to the high speed of microprocessors makes it possible to treat the digital controller as an analog device. This treatment allows drawing on the large body general control strategies for digital implementation. The emphasis in this work has been on cascade, multivariable decoupling, and feedforward

control (1,22,43) with some improvements from the computational capabilities of microprocessors.

Report Organization

The succeeding chapters will discuss in more detail the various parts (hardware, software, and strategies) developed for two controllers in the energy management system. It should now be noted that the hardware and software, though developed for a specific application, are very general in nature and have been applied to a number of different processes.

The material presented will be on three levels. The top level is the design philosophy of basing the control structure around the process to be controlled. The next level of detail discusses the resulting functional blocks (both hardware and software) and their purpose in the digital control system. The last level of detail is the collection of appendices. They are the assembler and macro level listings of the software for this work.

The material is presented in this tiered structure for a definite purpose--to make available the information on whatever level desired. The appendices can be used directly for an LSI 11 based system or translated to a different assembly language for a different microprocessor. The functional block description and flow diagrams contained in the text can be used by someone interested in developing a control system in a higher level language such as Fortran. The highest level, the design philosophy, is presented as an approach to designing control systems.

Chapter 2 will discuss the general design concepts and functional requirements for the hardware in the multiloop controller and the communication network as it applies to a multiunit distributed control system. The chapter will also contain a detailed description of the microprocessor used and the hardware "valve station" front end.

Chapter 3 takes a similar look at the programming. It discusses the purpose of the software and some simplifying concepts used for dealing with large, real time, software systems. The chapter will then discuss the actual software organization used in the work. The text covers data base and program memory organization, breaks down the total software system into functional groups, and discusses the function and usage of each.

Chapter 4 discusses the chiller controller. The chapter presents a physical description of a conventional mechanical chiller and a control strategy designed to reduce the energy consumption for a given load.

Chapter 5 discusses the boiler controller and presents a discussion similar to Chapter 4 of a medium-sized, oil fired boiler.

CHAPTER 2

HARDWARE

This chapter deals primarily with the hardware used for a particular problem. However, it should be emphasized that the design considerations are applicable to a wide range of process control problems. It should further be stated that the hardware (and software) described in this work are in fact quite general in nature and have been successfully applied to several different processes.

The Design Concepts

A computer system can become impossibly complex unless a simplifying design concept is followed. One such concept used throughout this work is structured modularity. A function, hardware or software, is implemented as a module. In hardware, this module is a printed circuit board. In software, the module is a program segment. If a particular function is needed, it is plugged in; if not, it is omitted. This approach has many advantages. Initial system generation is greatly simplified; the needed modules are made, tested, and plugged into the system. Modularity provides a framework for expansion and maintenance, a crucial factor in any system. Failures can be isolated and eliminated by simply plugging in fresh PC board modules until corrected.

To achieve efficient operation, a control system must not only control each unit, such as a boiler or chiller, but also be able to coordinate the action of these widely separated devices.

The control system must be cost effective considering the size of a multiacre energy system. The system must be flexible to allow various operational configurations considering the number and variety of energy devices and must be maintainable.

Overall Structure

The overall structure of an energy system such as this presents some very common control problems. The energy system is composed of many individual components such as boilers, chillers, and air handlers. Each are independent units but physically connected as energy consumers. For example, the boilers consume fuel oil to produce steam. The chillers consume steam and reject waste heat to the cooling tower to produce chilled water, and the various air handlers consume steam and chilled water to produce a conditioned space.

Control Structure

The control structure used throughout this work, a distributed cascade, deals effectively with these problems. A distributed cascade is a collection of separate cascade control loops connected by a higher level control. It is a mirror image of the physical system and therefore divides the control problem in a similar manner. Each distinct unit such as a boiler or chiller has its own multiloop controller. Each controller is connected to a communication network to extend the cascade and allow a high level controller to perform overall coordination.

With this structure the entire control system is not dependent on any single device or computer, but rather each computer is

responsible only for a small part of the whole. Failure of any piece of control equipment affects only the energy unit to which it is attached and is immediately apparent for maintenance. The distributed nature allows a minimum of expensive, high quality signal wiring; long range communication is handled by phone-grade cable. Startup of each unit can be performed independently. Maintenance is enhanced by the modular nature of each multiloop controller.

Hardware Organization

The hardware for each multiloop controller connected to an individual unit consists of three major subsystems (see Figure 2-1): the digital computer (microprocessor), the valve station (or computer to process interface), and the communication port for network communication. This particular organization was chosen to provide a flexible, easily maintained final design.

Each controller for a chiller, boiler, or air handler has the same basic parts: the microprocessor, the valve station, and communication port. The controller is customized for a particular application by plugging the appropriate interface modules into the valve station.

The Digital Computer

The microprocessor or digital controller (see Figure 2-2) used in this work is an LSI 11 (a 16 bit processor manufactured by Digital Equipment Corporation [44]) with 4K program memory and 4k RAM memory. Communication and process input and output (I/O) are handled through standard serial and parallel cards. The analog to digital converter is a single PC (printed circuit) board unit made by ADAC Corporation

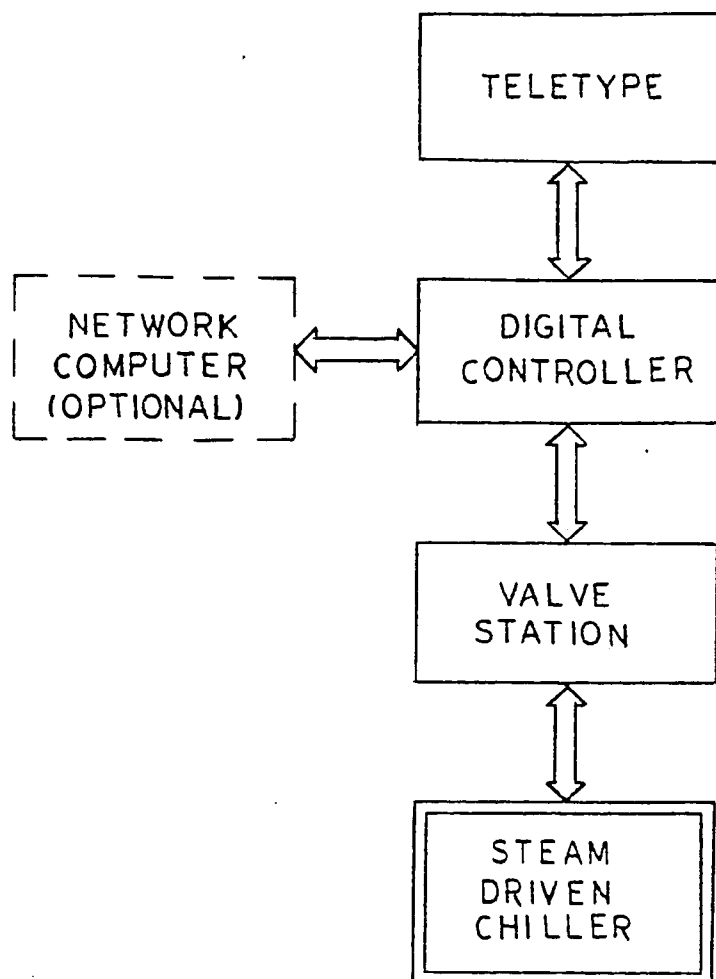


Figure 2-1. Digital control hardware organization.

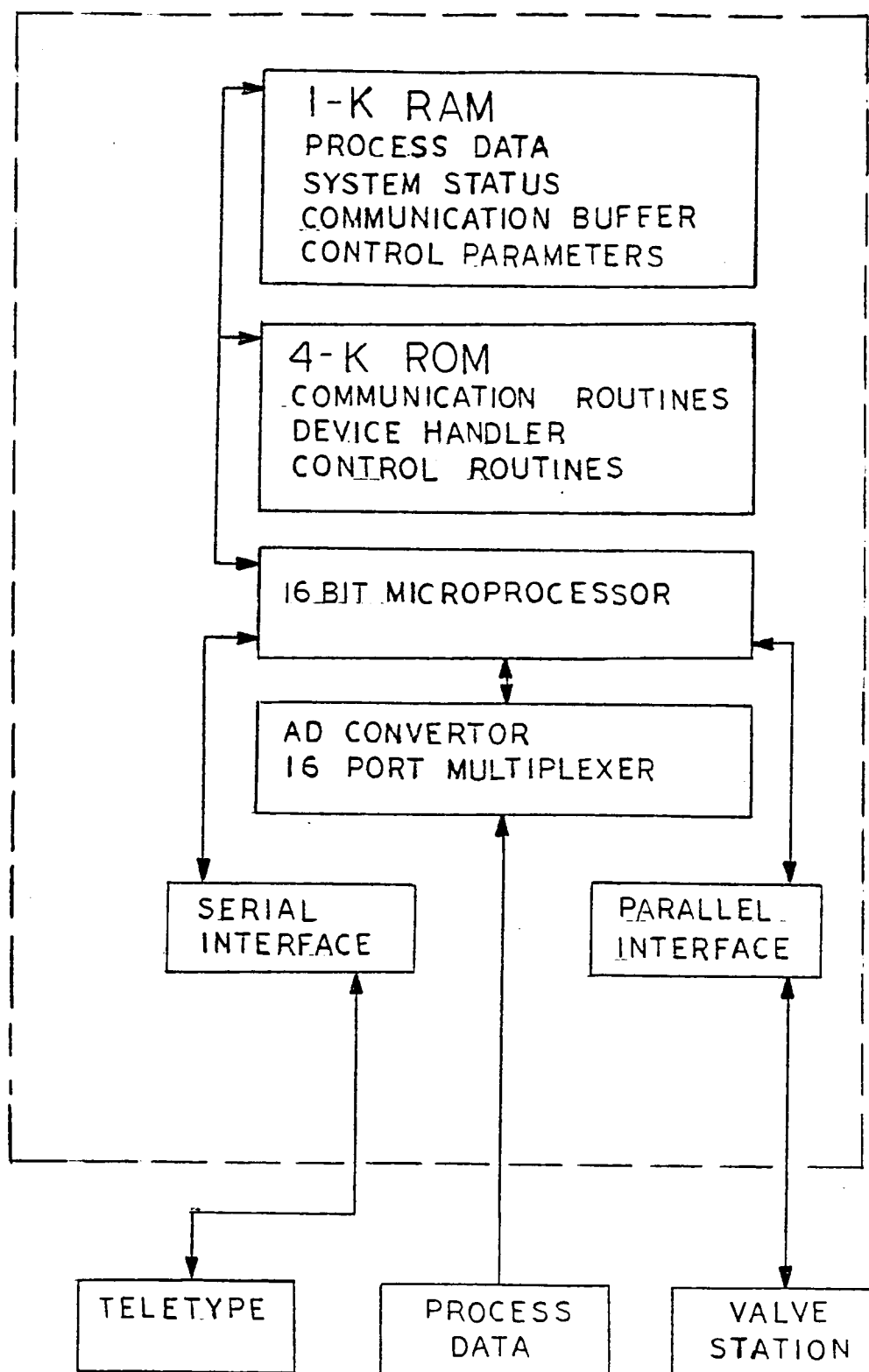


Figure 2-2. Digital controller design.

(45), and program memory is stored in nonvolatile EPROM (erasable, programmable, read only memory) chips on a MDB Corporation card (46). The entire computer for each multiloop controller is contained on six plug in cards. Maintenance can be performed by plugging in fresh cards until any failure is corrected.

The Valve Station

The second major component of the digital controller, the valve station, provides a basic operator and process interface to the computer. It is the modular, plug in structure of the valve station which allows the multiloop controller to be easily adapted to a wide range of processes.

The valve station provides five interface functions:

1. Precomputer analog signal conditioning for all process signals.
2. Metered readout of selected process signals.
3. Amplification and buffering of computer drive signals to process valves.
4. Auto/manual pushbutton lights to initiate automatic control and indicate operating status.
5. Manual control buttons to operate valves when not in automatic.

Analog Signal Conditioning and Metering

By definition, feedback control systems require information from the process. For the digital controller described here, this information is in the form of analog signals from measurement transducers

connected to the process. Electrically, the signals range from milli-amp current signals to multivolt signals. After connecting the wires from the transducer to the valve station, an appropriate signal conditioner card is plugged into the valve station and calibrated to produce a 0-10 volt range for the microprocessor. Finally, these 0-10 volt signals can be connected to meters on the front panel of the valve station for operator observation.

Valve Control Module

To further tailor the valve station for a particular application, there are six plug in slots in the valve station for the control of up to six valves per multiloop controller. There are two varieties of valve drive cards available: the first for electric or air motor driven valves, and the second for valve motors requiring a 4-20 milli-amp signal. Both of the valve drive cards have similar front panel appearance and function.

In addition to buffering and amplification of the drive signals from the computer, the valve drive cards have several important functions. Each valve drive card has several pushbuttons which fit through knockouts in the front panel. For daily startup, each valve card has manual increase and decrease buttons. With the aid of the analog signal meters, an operator can manually operate any valve to bring the process on line. After the operator has the unit close to operational settings, he can press a third button on the valve drive card to control that valve to the computer.

The last important function of the valve drive card is to monitor the state of the microprocessor. In the event of a computer

failure, the valve drive card drops the computer connection and reverts to manual. The valve positions freeze and the front panel lights blink to attract attention.

Communication Port

In addition to the pushbutton lights and meters, there is a communication port for more in-depth information concerning the process and control strategy operation. It is a conventional, high speed (240 characters per second) teletype line and can be used in two ways: during normal operation it is connected to a Network Node Processor (NNP), an intelligent Intel 8748 based network link (see Figure 2-3). During startup or troubleshooting, the NNP is unplugged and a high speed terminal is plugged in.

During startup, the multiloop controller is completely separated from network interaction. The operator can quickly look at any process value or parameter to make corrections or additions as needed with the local terminal.

After startup, the network node processor (NNP) is plugged in, connecting the controller to the network, the communication controller, its terminal, and ultimately to the high level overall controller (see Figure 2-4). The NNP contains a thumbwheel address switch ranging from 00 to 77. Each information package from the network communication controller is tagged with an address. The NNP checks each information package for its address and routes it to its controller if they match.

The second function of the NNP is to check for transmission errors with checksum and parity tests. In the event of a transmission error, the NNP informs its controller and the package is dumped. When

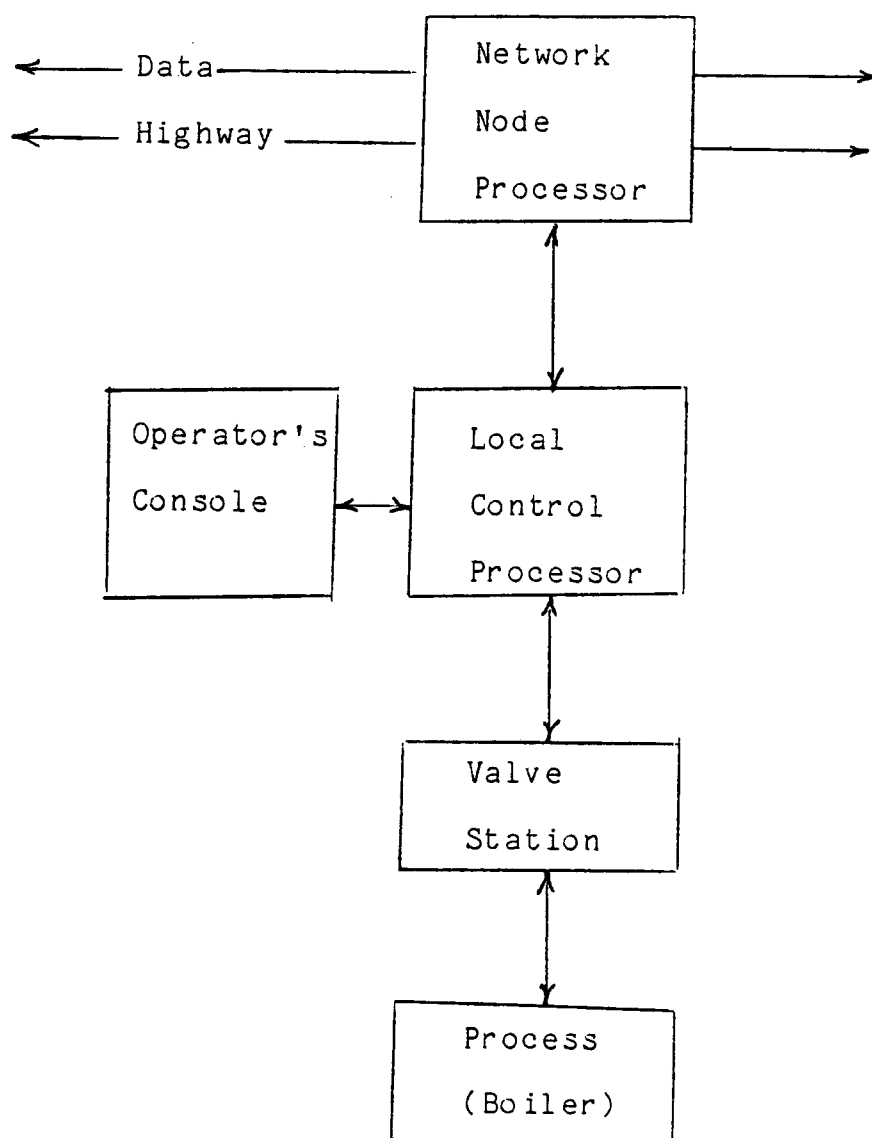


Figure 2-3. Communication network linkage.

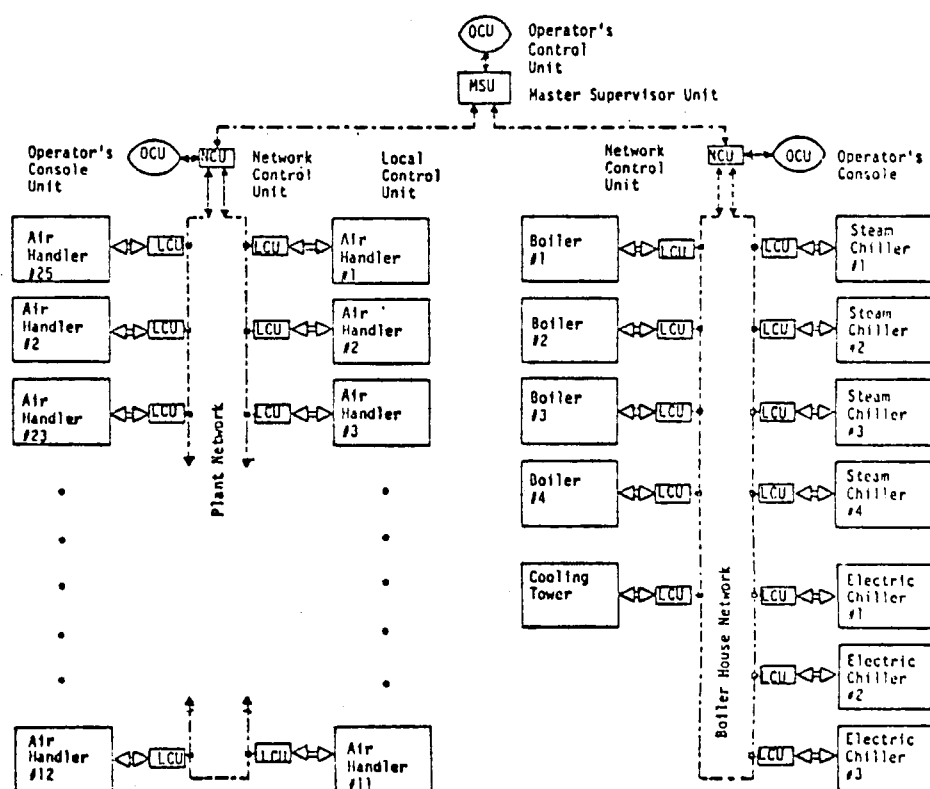


Figure 2-4. Communication network configuration.

the communication controller fails to receive an answer, it tries again during a later time slot.

In addition, the NNP checks the operational status of the network. The NNP will alert the communication controller if there is a break in the network leading to the next NNP as well as indicate if its controller has gone off line.

Conclusion

In conclusion, the hardware in this work was designed to complement the capabilities to today's low cost microprocessor in a dedicated environment. The hardware provides the interfaces along with an operator interface for control. The communication port provides a powerful startup tool as well as a network link for extending the cascade beyond the local controller. Throughout this work, the designs incorporate a unified concept of structured modularity. The next chapter on software organization examines the various system modules in the program software.

CHAPTER 3

SOFTWARE ORGANIZATION

As stated in the beginning of the previous chapter, it is misleading to present individual segments of a system as stand alone portions. All parts are interrelated and rely on each other to complete the desired goal: direct digital control. All parts are equally important; however, the emphasis of this chapter is on software. This chapter will deal primarily with the unified software architecture developed in this and related works (47,48,49).

Software Purpose

The intent of this chapter is to describe the software requirements for a multiloop digital controller. The software, as a direct embodiment of the control strategy must effectively organize and utilize the feedback information from the process in a real time, on line environment. That is, it must be able to dependably handle process control in an industrial environment. To be general, a multiloop control system, including software, must be capable of handling a wide range of processes from very simple items (valve position control) through very complex items (surge control of a large compressor) in a multitasking environment. The control strategy in this work is organized primarily as multiple cascaded loops.

Figure 3-1 is an example of cascade control. This figure shows multiple distinct control loops, each controlling a separate variable, with some of the controllers cascading into lower loops. The loops

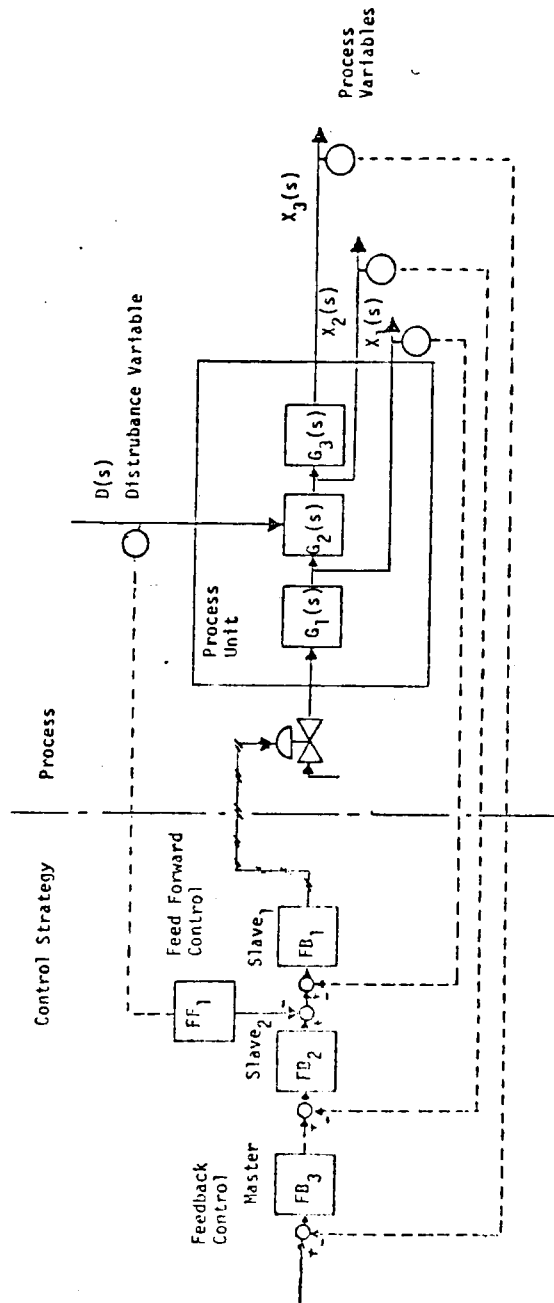


Figure 3-1. The cascade hierarchy structure.

must be independent and operate with a fair amount of time autonomy (50). The main purpose of the software in a digital controller is to implement that strategy; therefore, the principle of a cascaded hierarchy must also apply directly to the structure of a software system. Consider Figure 3-2 in which a software system is suggested to implement the general control strategy shown in Figure 3-1.

In a distributed but interacting environment, such as a plant-wide energy system, the control strategy must extend beyond an individual controller to an overall supervisor for coordination (49). The software must not only provide good local control, but must also transfer and receive information for a higher control element as shown by the network interface in Figure 3-2.

Process Control Software

There are many excellent digital control systems today (31,32, 51,52); however, they are almost all proprietary and tailored for specific systems. The requirements for a cost effective, microprocessor based digital controller necessitates a "bare bones" hardware system with some very specific control features which none of the available, nonproprietary software packages have.

There are, however, several software utilities available, most notably, real time monitors (37,38). While not control languages in themselves, the monitors provide a structure around which control routines can be built. However, the general nature of these utilities translates into a comparatively large routine from both a memory and a computer time basis. For example, RT 11, a very popular real time monitor for the PDP 11 computer, requires in its most stripped form

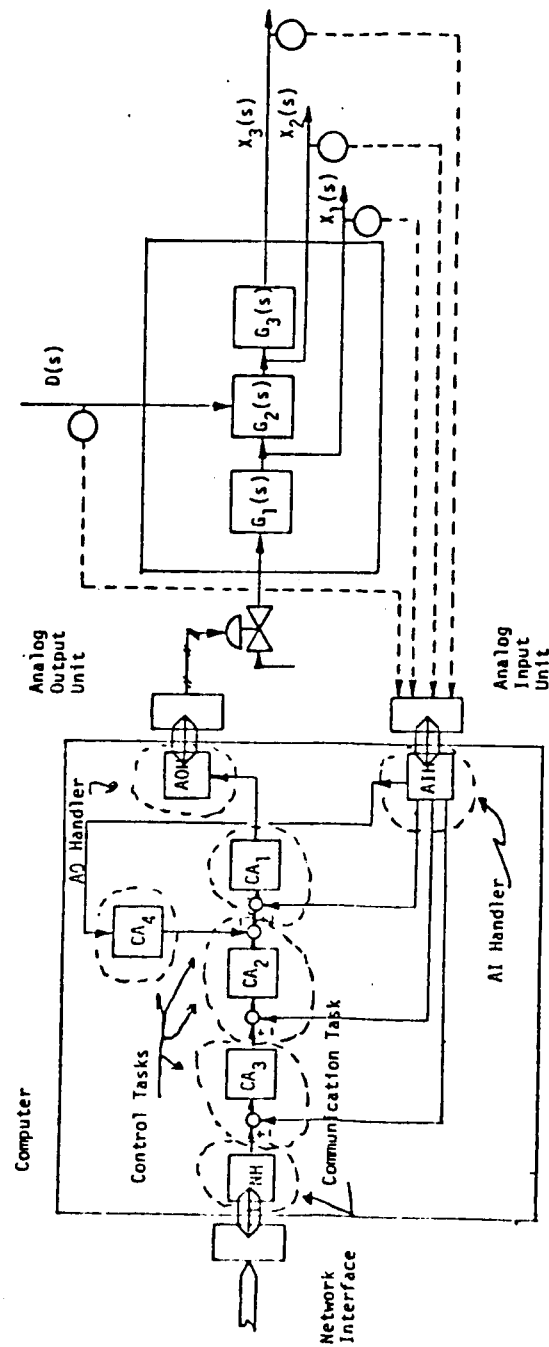


Figure 3-2. The cascade hierarchy structure applied to software.

approximately 2K words of memory. With the very simple requirements for the work done in this dissertation, a real time monitor was developed which requires only 250 words of memory and virtually no computer time.

Software Concepts

As in hardware, the software would become very complex unless some clear unifying concept is developed. From the standpoint of scope, modular programming (53,54) was used. Modular or functional block programming is a method in which the total design is split by function. Each function is then defined such that a segment of code using well defined input and output data (in this system organized as tables) can be written. The total project then becomes a task of defining functions, writing code to service those functions, and thereby maintaining the flow of information through the system. As can be seen in Chapter II, modular programming is also used in hardware in the form of printed circuit boards.

Another concept employed to meet the system design requirements is formalized code. In a multitasking system with shared code, synchronization problems can be eliminated by using reentrant code (55). Reentrant code involves writing a program which has no dedicated memory write locations. Each time the program is called, an unused portion of memory is dedicated to the program. Program sections requiring data base operations can be handled by a table driven reentrant code with the remainder serially reusable.

A final programming concept used in the development was necessitated by memory device selection. It is of particular importance in

an on line system to protect program integrity. The program should be safe from accidental erasure or loss from power failure. EPROM (erasable, programmable, read only memory) provides all these features (56). As such, it enforces a sharp segmentation between program memory (read only) and data memory (read write memory) development.

Computer Memory Organization and Information Flow

These software guidelines coalesce into a few rules best described in terms of computer memory organization and information flow:

1. Write all control algorithms as separate, table driven, reentrant (sharable) code for read only memory.
2. Store all information concerning each control loop in individual tables with identical formats in read write memory.
3. Develop interface handlers to directly write into and read from these control tables.

Control Table Format

The control table format is the fundamental basis of the data flow for the multiloop controller. As stated previously, a control table containing tuning parameters, current process values, and controller values is assigned to each control loop (see Figure 3-3). All tables have a common format.

The first parameter group in each table is concerned with the variable under control and its acceptable range:

1. The zero position in each table is assigned to the controller setpoint or target value for the variable.

<u>TABLE POSITION</u>	<u>FUNCTION</u>
0	SET POINT
1	PROCESS
2	HIGH LEVEL
3	LOW LEVEL
4	GAIN
5	GAIN 2 / THRESHOLD
6	CONTROLLER OUTPUT
7	SUM OF ERROR
8	SAMPLE TIME
9	UNUSED
10	BUFFER VALVES
11	
12	
13	
14	

Figure 3-3. Control table format.

2. The next position is assigned to the actual value of the variable as determined by the analog to digital converter.
3. The second and third positions in the table are assigned to high and low level checks for the variable. The control algorithm checks the setpoint against these limits and keeps the variable within these limits by appropriate control action. The limits are also used for reset wind-up elimination as described in the later section on the proportional plus integral control algorithm.

The second group of values is devoted to controller parameters.

1. The fourth and fifth positions in the control tables contain tuning parameters. They are the locations where the controller looks for proportional and integral gains, thresholds, etc., as required by the controller.
2. The sixth position is occupied by the controller output. For valve controllers, it is an up or down count to move a valve. For a proportional plus integral controller, it is a setpoint that is also cascaded into another control table.
3. Position seven is a scratch pad used by the software. In the proportional plus integral controller it is the integral sum.
4. Position eight is a cycle rate for the real time monitor. With it, the control algorithm is executed at a frequency in accordance with strategic requirements.
5. The rest of the positions (up to 20) are undefined and are used by custom controllers for various purposes.

Read Write Memory Organization

As stated earlier, there are two types of memory employed in the multiloop controller: read only, nonvolatile memory for program storage, and read write memory for dynamic information. The control tables contain changing information and as such are located in read write memory.

In addition to the control tables, there are other data paths which require dynamic storage (see Figure 3-4). The system status (identified in Figure 3-4 as STATUS) requires dynamic storage. It contains information about the system as a whole in a concise map of all loops showing which are active.

Another major block in data memory is the serial line buffer area (identified in Figure 3-4 as I/O BUFFER). All serial communication from the network or local teletype go through buffers. As characters are typed into the computer, they are stored in read write memory as a string until an end of line character (a carriage return) is typed. The buffer is then decoded and marked empty. Output is handled in a similar way. Any message to be sent from the computer is packed into a buffer and trickled out as the serial line becomes clear.

The real time monitor also requires some use of dynamic memory. The basic function of the monitor is simply to keep track of time. To accomplish this, the monitor keeps a time and date clock (identified as TIME in Figure 3-4). In addition, the monitor keeps a countdown clock for each task (identified as VECTOR CLOCKS in Figure 3-4) as will be described later in this chapter.

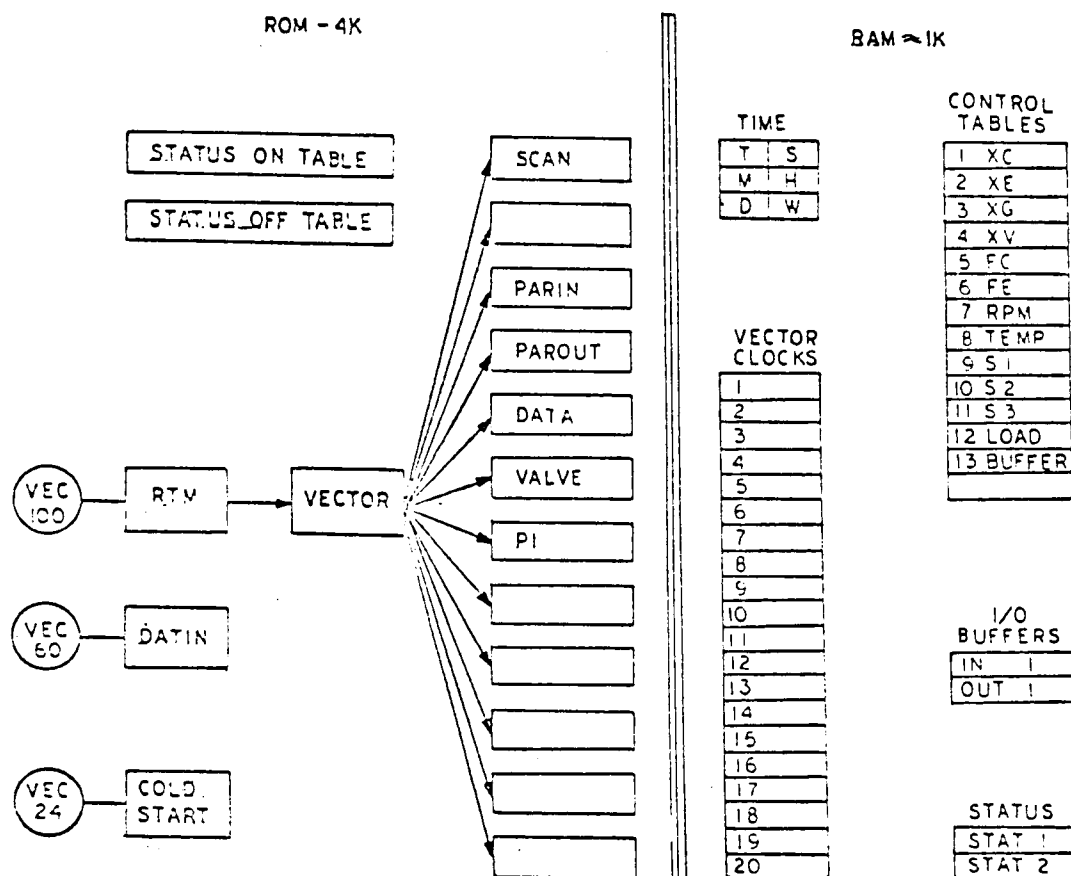


Figure 3-4. Software organization.

Functional Blocks

As stated earlier, the overall system is divided into small functional blocks. Figure 3-2 is one such division for a general cascade type hierarchy. Figure 3-2 also suggests that there are only three basic types of functional blocks in the system:

1. Control tasks
2. Communication tasks
3. Device handlers

Control Tasks

A control task defines some specific feedback or feedforward control function for the system. The task is defined by a pair of pointers and an appropriate control routine. The control routine is a functional block which contains a specific control algorithm and the pointers contain the address of an input and an output control table. The input table contains the process data, the desired setpoint, and the control parameters. The output table is the destination of the calculated results.

The task can use a conventional control algorithm (i.e., proportional plus integral) or it can augment these conventional routines for more sophisticated operations. For example, the control algorithms can operate on calculated variables instead of raw process data. This feature is accomplished by a separate block which performs the calculation used by the control algorithm as if it were raw data. An example of this would be the energy load on a heat exchanger calculated as a function of a flow rate and a temperature difference.

More sophisticated control strategies such as adaptive gain, dead time control, and multivariable decoupling can be handled in a similar manner using a combination of calculated variables and non-conventional algorithms.

Communication Tasks

In a distributed control system, by definition, external communications are required. Without such communication, the distributed system is simply a series of individual controllers, as indicated by the network interface in Figure 3-2, page 29. The purpose of the external communication is to allow the control strategy to extend beyond an individual controller. The communication task performs this function by linking the network communications with the control tables. This structure allows higher level network based control functions to merge with the control strategy in exactly the same manner as do all local control functions. Whether network based or local, all intercontroller communication is accomplished through the same set of control tables.

Device Handlers

The last type of functional block is the input/output (I/O) device handler. These handlers are defined not by the strategy but by the interface hardware which the system must drive. Their sole function is to be the front line software connection to hardware. There is one handler per device, which might typically include analog input, analog output, parallel input, parallel output, and the serial communication port. A handler must provide the link between the device and the various data tables associated with that device. An example is the

interaction between the communication task and the serial line handler. The communication task would, in response to a network request, fill an output buffer. The serial line handler would then, on a hardware interrupt basis, trickle out the characters in the buffer as the line becomes available.

Real Time Strategy

Now consider the total interactions between all the various components using an example from the energy management application. The example (shown in Figure 3-5) is a cascade strategy in which the top level considers the total plant operation and determines a desired setpoint for the chilled water temperature on a specific unit. The local chiller strategy then uses that setpoint to determine a desired turbine speed. The speed control strategy then determines a desired valve position and the valve position strategy determines the actual signal to the valve necessary to achieve that position. This four level cascade strategy can be logically implemented in the control system as shown in Figure 3-6. (Note the advantage of the cascade philosophy. This example clearly shows how the distributed nature of a cascade system is totally compatible with the cascade nature of a distributed computer system.)

The primary difference between the block diagrams for the strategy (Figure 3-5) and for the software (Figure 3-6) is the presence of the control tables in the software implementation. The control tables provide the only link between the various tasks and handlers. This link is by definition passive because these tables

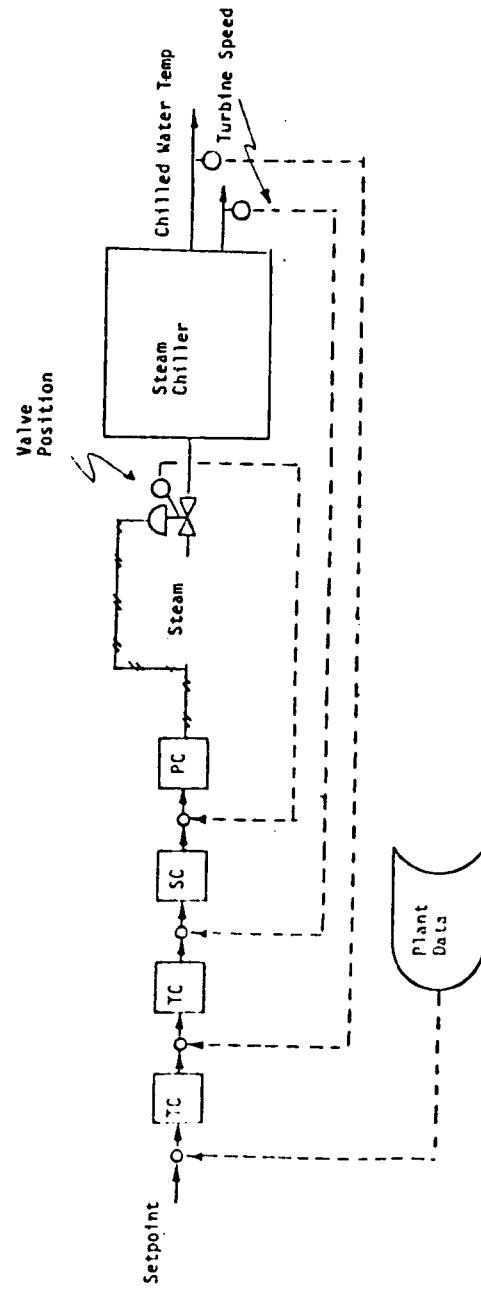


Figure 3-5. Steam chiller temperature control and optimization strategy.

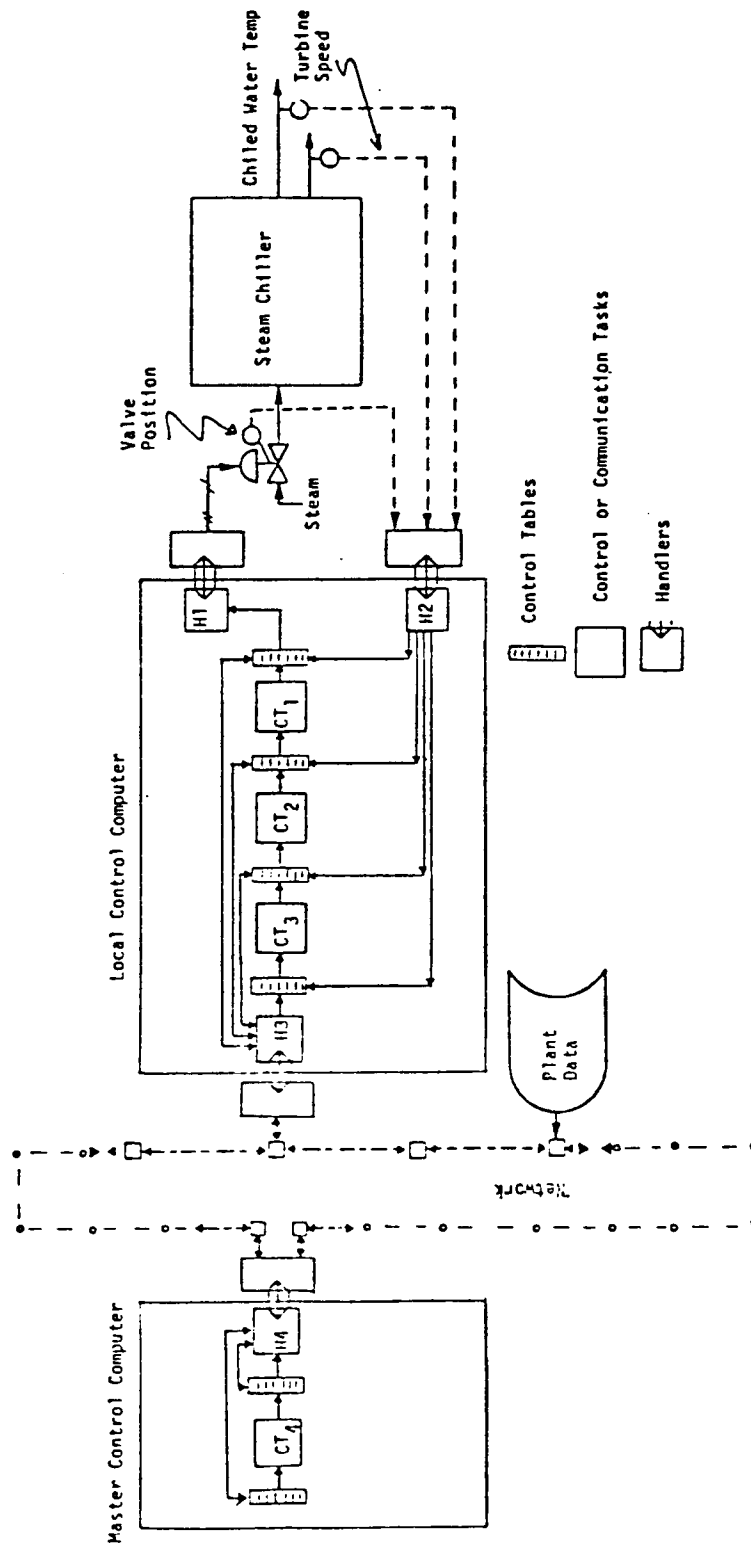


Figure 3-6. Software implementation of chiller temperature control and optimization strategy.

are just data. The passive nature of this link insures autonomy between the various blocks.

The software system described in Figure 3-6 indicates only the specific task and the flow of information between the tasks. Because this is a real time application, the "scheduling" of tasks and the orderly sequencing of program execution must be considered. Real time operation means the computer is performing calculations on the process values as they are being collected. The computer schedules programs (tasks) by executing them in a time ordered fashion. Here lies the simplicity of this organization. The cascade structure provides the autonomy to allow totally independent execution of each block of code. Each task can be executed on an independent clock cycle. It is only necessary to externally order the execution frequencies properly. That ordering is well defined from a process control point of view. The inner loop of a cascade must be executed at a higher frequency than the next outer loop. The third outer loop must be executed at a lower rate and so on for all loops, with the outermost loop operating at the lowest frequency of all. Implementation is very simply from a strategic base from high to low frequency while moving up the cascade architecture. Figure 3-7 indicates the ordering for this example using the clock cycle (in seconds) at which each block is executed.

Note that some blocks are executed on an external interrupt rather than a clock cycle. This feature is typically associated with communication tasks and is totally consistent with the autonomous structure. Each task and handler is asynchronous and is a function only of the interrupt which initiates the execution.

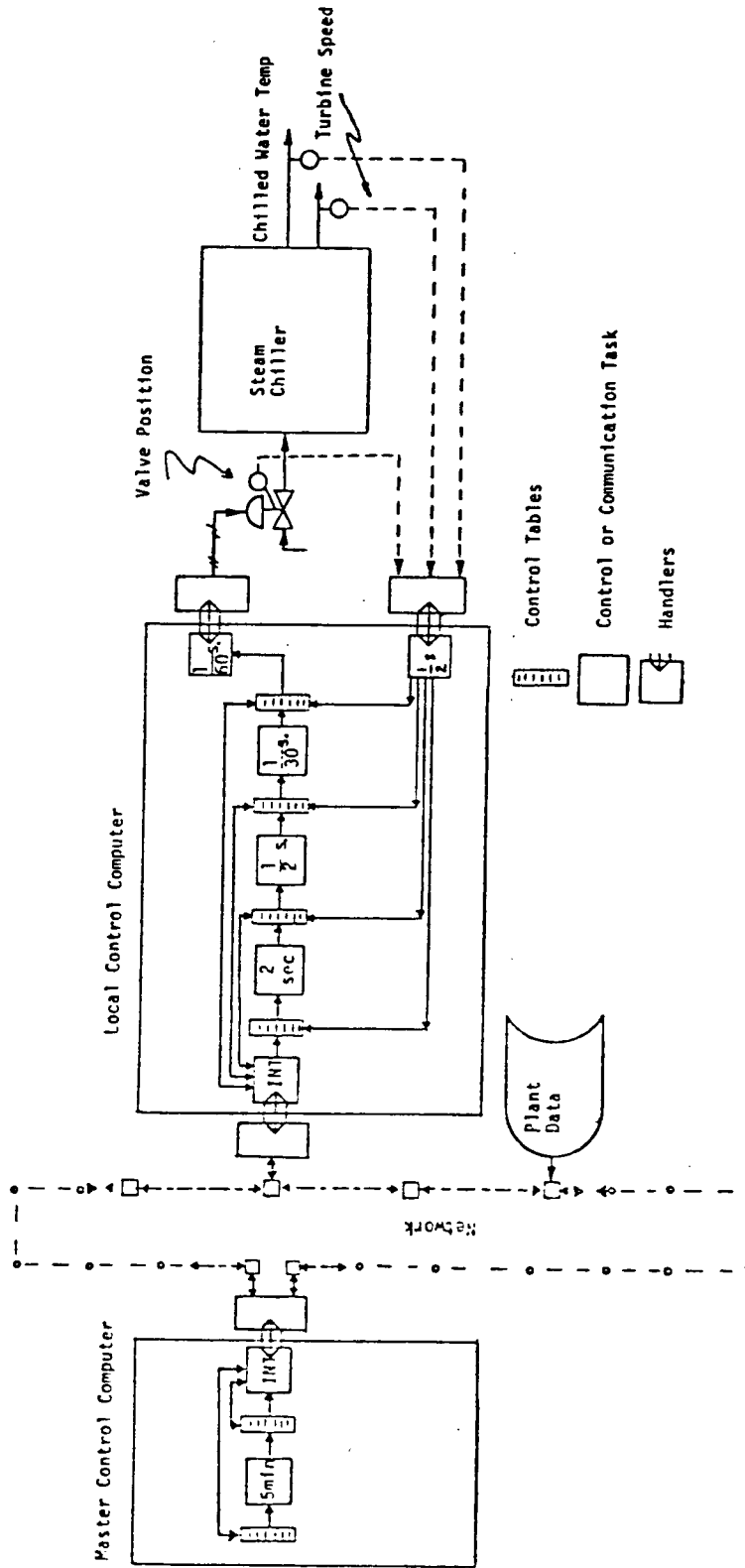


Figure 3-7. Software implementation execution cycles.

The Real Time Monitor

The real time monitor is the core of the system. It is the central heartbeat which schedules and monitors all time based activities within a single computer of the distributed system. The monitor must preserve the time integrity of the numerous parallel tasks which are continually being executed as well as to eliminate schedule conflicts which might saturate the limited capacity of the computer hardware.

Low "overhead" design. The real time monitor functions must be present in any nontrivial real time computer application and are the components which usually contain most of the so-called software "overhead" (e.g., the minimum code required to execute one simple real time task). The most distinguishing characteristic of this software organization is that the program requirements for the monitor are extremely low. The monitor size in terms of the storage requirements is a direct function of the number of tasks to be scheduled and the resulting time overhead is, for all practical purposes, insignificant.

Before examining the details of the structure and operation of the real time monitor, several factors should be considered which contribute to the short, crisp nature of the monitor functions in this system.

1. The only program code which must be directly supported by the monitor are the various clock routines. All other programs are identified as either tasks or handlers, both of which are executed on an interrupt basis.

2. The monitor performs no direct program synchronization.

Such requirements are satisfied implicitly by the autonomous nature of the cascade structure and the asynchronous nature of the reentrant, interrupt driven code.

3. Since it is assumed that for these small systems the software will be generated off-line and on a higher level machine, the monitor contains no program development facilities (i.e., editors, assemblers, etc.).

The clock structure. The main function of the monitor in this organization is to keep track of time. This is accomplished by a short interrupt driven service routine. This routine (which is the complete monitor code) responds to an external "clock" interrupt which occurs once every 1/60 of a second (a duration of time defined in this system as a "tick"). This time unit was chosen as a result of hardware availability (the interrupt is generated by the power line AC). In addition, the time unit has a high enough frequency for the fastest process control loops in these applications. During each execution, the monitor performs only two functions, both of which are associated with the various software clocks in the system.

First, the monitor updates the system time and date clock which consists of six coupled counters in which the time is stored in units of "ticks," "seconds," "minutes," "hours," and "days."

Second, the monitor decrements the various task schedule clocks. There is one task schedule clock for each real time task in the system. Each clock is an integer counter which can be defined to have units

in either ticks, seconds, minutes, or days and must be initially set to a nonzero value. Associated with each task schedule clock is a "pointer" which links the clock to its specific task definition.

Task scheduling and execution. The task schedule clocks and the associated pointers provide the structure in which all tasks are scheduled and executed. As the real time monitor decrements each schedule clock, it tests for a zero result. If the result is zero, the monitor transfers control to the address contained in the associated pointer. The pointer address is that of a Task Vector routine which sets the input table pointer and output table pointer to the appropriate control tables and transfers program control to the specific task to be executed (as shown in Figure 3-8).

This organization provides considerable flexibility in scheduling and rescheduling tasks. If the task is cyclic in nature (i.e., a control task), it reschedules itself by setting the schedule clock with the Sample Time Value from the control table.

Note the simplicity of this organization. The monitor updates the various clocks and ignores the status and nature of the task. The control of the schedule is independent of the monitor and is accessed externally through the task schedule clock and the execution cycle parameter in each control table. If a task is one time or noncyclic in nature (i.e., a limit violation warning), the task is requested by setting the clock to one "tick" and is not rescheduled after its execution. Tasks which are normally cyclic can be terminated (i.e., a control loop can be put into manual) by setting its clock to zero.

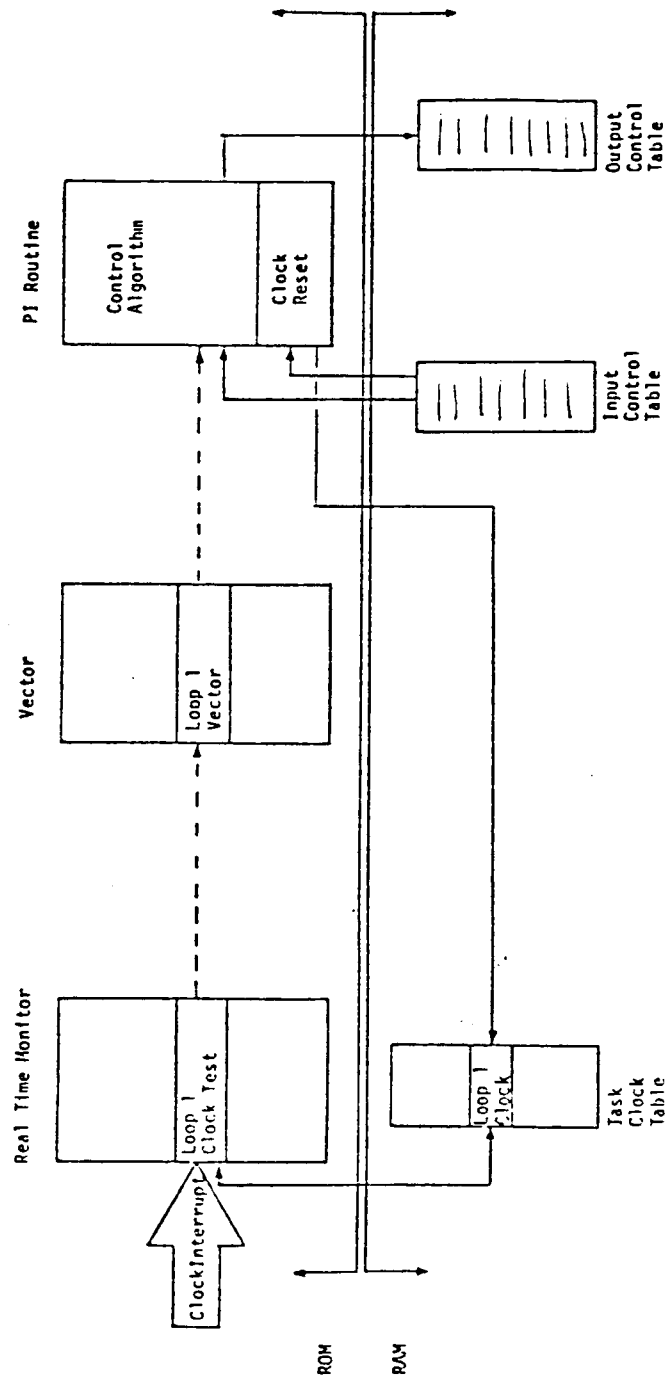


Figure 3-8. Real time task execution.

An inactive program can be activated by setting its clock to any non-zero value.

Conflict resolution. A subtle point in this organization concerns the rescheduling of cyclic tasks. If the schedule clock is reset at the end of the task execution instead of the beginning, the hierarchical nature of interrupt programming will function to eliminate all schedule conflicts. Suppose, for example, that two cyclic programs are originally on the exact same cycle (i.e., to be executed once a second, on the second). In the first cycle there will be two simultaneous requests; however, one will necessarily be executed before the other. The first one executed will be the first one rescheduled for execution. This procedure results in an initial "phase shift" and eliminates the conflict in all future cycles and insures that all rescheduled time intervals are accurate. This basic principle holds no matter how many tasks are active in the system and will always result in an operation which meets the required schedule of each task (provided, of course, the capacity of the computer is not exceeded).

Load calculation. A very convenient (but not essential) feature to add to the function of the monitor is a load calculation which continually calculates the "free time" of the system. This is done by adding a "background" program to the system which continually performs a floating point addition when there is no task or handler activity in progress. At each clock tick, the monitor reads the value and sets it to zero. The count is directly proportional to the

amount of "idle" time and can be scaled to represent "percent loading." Since the loading varies from tick to tick, it is also helpful to filter the result so that it is representative of the average loading on the system.

Functional Description of Software Modules

Now that an overview of the complete system has been presented, it will be of benefit to look in more detail at the functional description of each software module. First, however, a brief discussion of the programming conventions is in order.

1. Each module is written as a complete software routine with its own entrance and exit points.
2. All routines are written for an interrupt environment by saving all scratch registers on entry and restoring them on exit from the routine.
3. All routines driven directly by external interrupts (the monitor and serial line handler) are written as interrupt service routines with a standard interrupt return terminator. (The cold start module, though technically a routine, does not have an exit, but rather branches into the endless background load calculation loop.)
4. All other routines are written as subroutines with standard subroutine return terminators.

These programming conventions allow an interrupt to occur at any time, permit the service routine to call any module or modules in any order, and provide a retrace path back to the program location which was interrupted.

Control modules. The monitor schedules a control task when its task clock has counted down to zero by calling its task vector as a subroutine. The task vector sets two pointer registers, one at the input control table for the process value, the desired setpoint, and controller parameters, and the other pointer at the output control table for the calculated result. The task vector then makes an additional subroutine call to the appropriate control algorithm. After making its calculation, the control algorithm stores its answer in the output control table and returns to the task vector. The task vector then resets the task clock with the sample time (see Figure 3-1, page 27) from the input control table and returns control to the monitor.

Proportional Plus Integral Controller

The proportional plus integral plus derivative (PID) controller and the proportional plus integral (PI) controller, labeled PI in Figure 3-4 (page 35), are the primary control tools of the process industry, and papers have been written about their digital implementation (57,58,59). While the microprocessor has the computational power to allow much more sophisticated algorithms, the conventional PI controller has remained the standard of the industry for the following reasons:

1. Maintenance and operations personnel are familiar with the equipment.
2. Tuning is simple and straightforward.
3. It is well suited for a feedback, feedforward cascade structure.

While the PID approach is conventional in terms of analog control, its application is not entirely straightforward and obvious in terms of computer control. While many commercial systems are available which offer PID control, the literature does not adequately document the subtleties for the person who is developing the system inhouse. With that caution in mind, it may be helpful to look at the system implementation and some important features.

Perhaps the most common control algorithm in use today, the discrete equivalent of the proportional plus integral controller, is formed by replacing the integral by a sum. Using rectangular integration, the algorithm is:

$$M_n = K_p * Err + K_i * \int Err + M_r \quad (3.1)$$

Where M_n = Controller output

K_p = Proportional gain

Err = Setpoint error

K_i = Integral gain

M_r = Initial value of calculated result

This form is known as the position form of the algorithm, since an actual setpoint is calculated from the error sequence.

The second version of the algorithm is known as the velocity form. Instead of calculating a setpoint, it generates a change for the cascaded setpoint. The algorithm is:

$$\Delta M_n = K_p * Derr + K_i * Err \quad (3.2)$$

Where ΔM_n = Controller output

K_p = Proportional gain

Derr = Difference between current and last setpoint error

Err = Current setpoint error

Ki = Integral gain

The velocity form does not need initialization for the summation; however, this form needs initialization for the setpoint and error deviation as will be discussed shortly.

The two forms are mathematically equivalent; however, the position form was chosen since its output can be used not only for setpoint control but also as a multiplier (for ratio control) without an additional function block.

PI controller flow diagram. Consider now Figure 3-9, the flow diagram of the PI algorithm used in the multiloop controller. (The software listing of this algorithm is in Appendix G.)

The algorithm starts with a check of the setpoints for setpoint violations. Normally the setpoint would never exceed these bounds; however, this feature is very helpful during startup and other abnormal operation. If the setpoint is found outside one of the saturation limits, it is reset to the appropriate value.

The next two computational blocks calculate the error term and compensate for a negative acting process. Feedback control requires the error term be recalculated by subtracting the process value from the setpoint to produce an error in a process with a positive feedforward gain. Most processes have a positive feedforward path; however, some, such as a heat exchanger used for cooling, do not. As the flow of coolant increases, the temperature of the exchanger decreases.

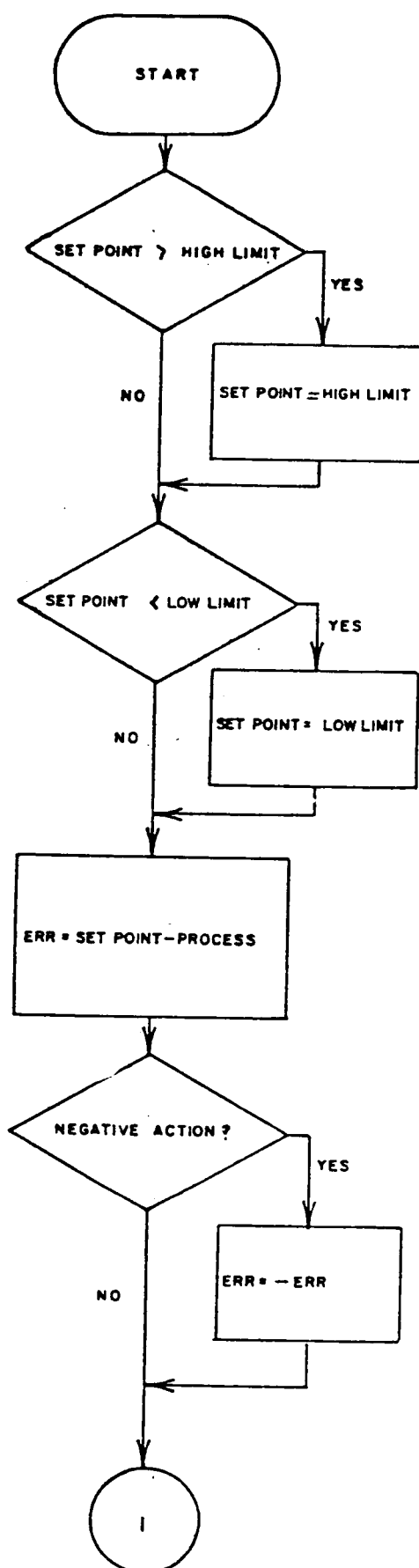


Figure 3-9. Proportional plus integral (PI) controller.

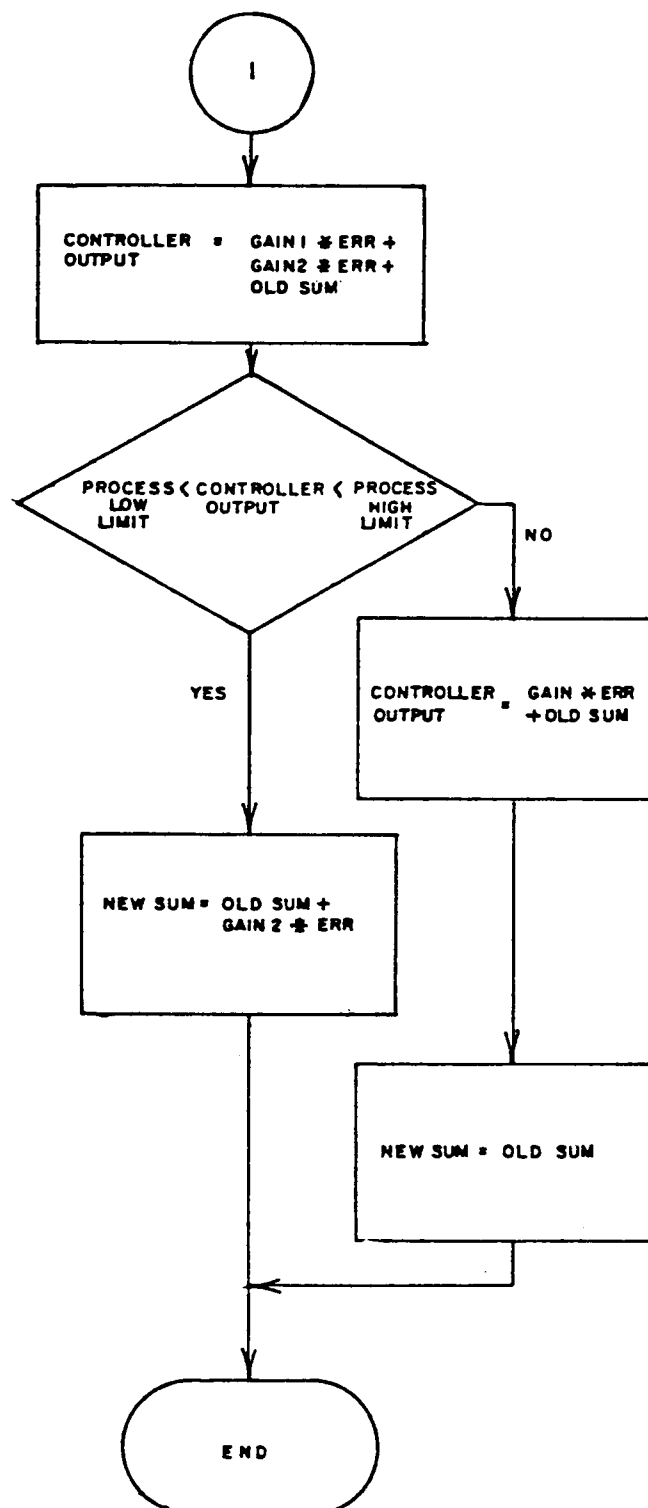


Figure 9 (continued).

This is an example of a negative acting process. To compensate, the controller must have an additional negative sign. The algorithm checks a status bit in the control table and reverses the sign of the error if the process is negative acting.

The next calculation produces a trial controller output. The trial calculation involves adding the proportional term to the updated summation.

In the next decision block, the algorithm looks ahead to the output control table for its saturation limits. If the trial calculation is outside the limits, it is thrown out and a new calculation is made without updating the summation to prevent reset wind up.

Bumpless transfer. Several points should now be considered--the first involves initial conditions and "bumpless transfers." Bumpless transfers means a smooth transition when going from manual to automatic operation. From an algorithmic point of view, this involves correctly setting initial conditions before the first pass through the software. For the position form of the algorithm, the sum must be initialized to the current value of the cascaded setpoint. Secondly, the setpoint for every loop except the outermost must also be reset to the current process value. This requirement applies to both the position and velocity forms of the algorithm. In the multiloop controller, this function is performed by the parallel input handler in response to an operator request for automatic control.

Consider the example shown in Figure 3-5, page 39. During startup the operator brings the chiller up to operational speed at

close to the final conditions. As he turns on the control loops, the chiller should remain in a static operating condition, modifying valve positions only to maintain performance, and not move to some previously defined setpoints until the outermost loop is engaged and moves the chiller to a different operating condition.

Bumpless tuning. If the position algorithm is implemented literally as in Equation 3.1, a change in the integral gain will cause a disturbance or "bump" in the controller output. However, if the sum is computed:

$$NSum = OSum + K_i * Err + Mr \quad (3.3)$$

$$Mn = K_p * Err + NSum \quad (3.4)$$

$$OSum = NSum \quad (3.5)$$

there will be no disturbance and tuning will be bumpless.

Interacting tuning parameters. A common form of the algorithm links the tuning parameters together. The equation, including the derivative mode, has the following form:

$$Mn = K_c (Err + (T/Tr) * Err + (Td/T) * Derr) \quad (3.6)$$

Where K_c = Overall controller gain

T = Sample time (for rectangular integration)

Tr = Reset time

Td = Derivative time

This form is the conventional analog form of the tuning parameters. It ties the integral and derivative mode gains to the proportional action, but the noninteracting form does not. During the course of installing and tuning several multiloop controllers, it has been found occasionally effective to completely tune out the proportional action for smooth control on very sensitive loops with noisy signals. As a result, the noninteracting version is used in the multiloop controller.

Reset wind-up. "Reset wind-up" is the phrase which means the continued integration of the error after the cascaded variable has reached saturation. This condition is not desirable. To eliminate wind-up, the PI controller needs to be aware of the saturation (high and low) limits of the next variable down in the cascade and stop the integration when its calculated result (setpoint) reaches that limit. In the multiloop controller, one of the pointers is to the output table and the algorithms checks the limits as shown in Figure 3-9.

Saturation limits. Incorporating these high and low limits in the control tables proved not only useful for anti-reset, but also of great benefit during startup. As shown in Figure 3-9, the saturation limits are also used as an initial check of the setpoint and are an acceptable value of the process. During startup, this acceptable range can be reduced to a very narrow band to eliminate wide fluctuations in the variable until the controllers are properly tuned.

Lead Controller

The lead controller is the basic feedforward element used in the multiloop controller. The feedforward controller, shown in

Figure 3-1 (page 27), is used to compensate for measured disturbances entering the process. Complementing the feedback controller, the feedforward bases its control action on the measurement of a disturbance and attempts to prevent process upset by acting before the controlled variable deviates from its setpoint. Because the controller is a feedforward element, it has no information to determine if it was successful and must be "trimmed up" by a feedback element.

The two elements, feedforward and feedback, should be used in pairs and, as a result, the feedforward element "adds" into the summation term of the trimming PI controller.

The feedforward element for most applications in a cascade structure need not be complicated because any error will be removed by the feedback loop. The element used in the multiloop controller is the classical lead compensator (61,62) which has the Laplace transfer function as follows:

$$(1 + \text{Tau1} * S) * Kf / (1 + \text{Tau2} * S) \quad (3.7)$$

where Tau1 = Corner frequency, adjustable

Tau2 = Corner frequency, adjustable

S = Laplace operator

Kf = Gain

A bilateral transformation (63) of Equation 3.7 produces a discrete form as follows:

Mn =

$$Kf * (In * (1 + \text{Tau1}) + In1 * (1 - \text{Tau1}) - Mn1 * (1 - \text{Tau2})) / (1 + \text{Tau2}) \quad (3.8)$$

Where: M_n = Current output

M_{n1} = Last output

K_f = Gain

$Tau1$ = Corner frequency

$Tau2$ = Corner frequency

I_n = Current input

I_{n1} = Last input

Valve Controller

The last standard control routine discussed (labeled VALVE in Figure 3-4, page 35) is the valve position controller. Position control of valves is usually not done in analog control systems because of the cost of the additional controllers. In a digital controller such as the multiloop controller, the only additional cost is that of the valve position transducer, memory, and computer time. It is well worth the effort to include positioning routines as standard practice to eliminate common valve nonlinearities such as hysteresis or sticking.

This algorithm is designed for air or electric driven valves. For electric motor-operated valves, the valve station is connected to relays on a reversible motor. The control output is a tick count or time duration during which one of the relays should be closed. For the air-driven valves, the valve station is connected to two solenoids on the valve bonnet. One is connected to 20 PSI supply air and the other to a vent. The solenoids then control the valve by adding or venting air in a manner similar to the electric version.

The algorithm is basically an integral mode controller in a velocity form:

$$V_n = K_s * Err \quad (3.9)$$

where

V_n = Controller output (change in position)

K_s = Controller gain

Err = Error

The controller output is a positive or negative tick count used to close the appropriate relay. The listing for this algorithm is in Appendix F.

Valve controller flow diagram. Consider Figure 3-10, the valve controller flow diagram. The algorithm starts by checking first the valve position and then the setpoint for saturation limit violations. The algorithm then checks the valve position and will attempt to move the valve if it is outside the saturation limits. Similar to the PI algorithm, the setpoint is checked and if found outside it is reset to the appropriate limit.

The error is calculated by subtracting the valve position from the desired setpoint, and the controller output is calculated by multiplying the result by the controller gain.

The next block, a decision block, tests the result against a second tuning parameter, the "threshold." If the controller output is less than the threshold, it is set to zero, and the algorithm is ended.

Bumpless transfer. A few points should now be discussed. As described earlier, bumpless transfer refers to a smooth transfer from manual to automatic control. From an algorithmic point of view this

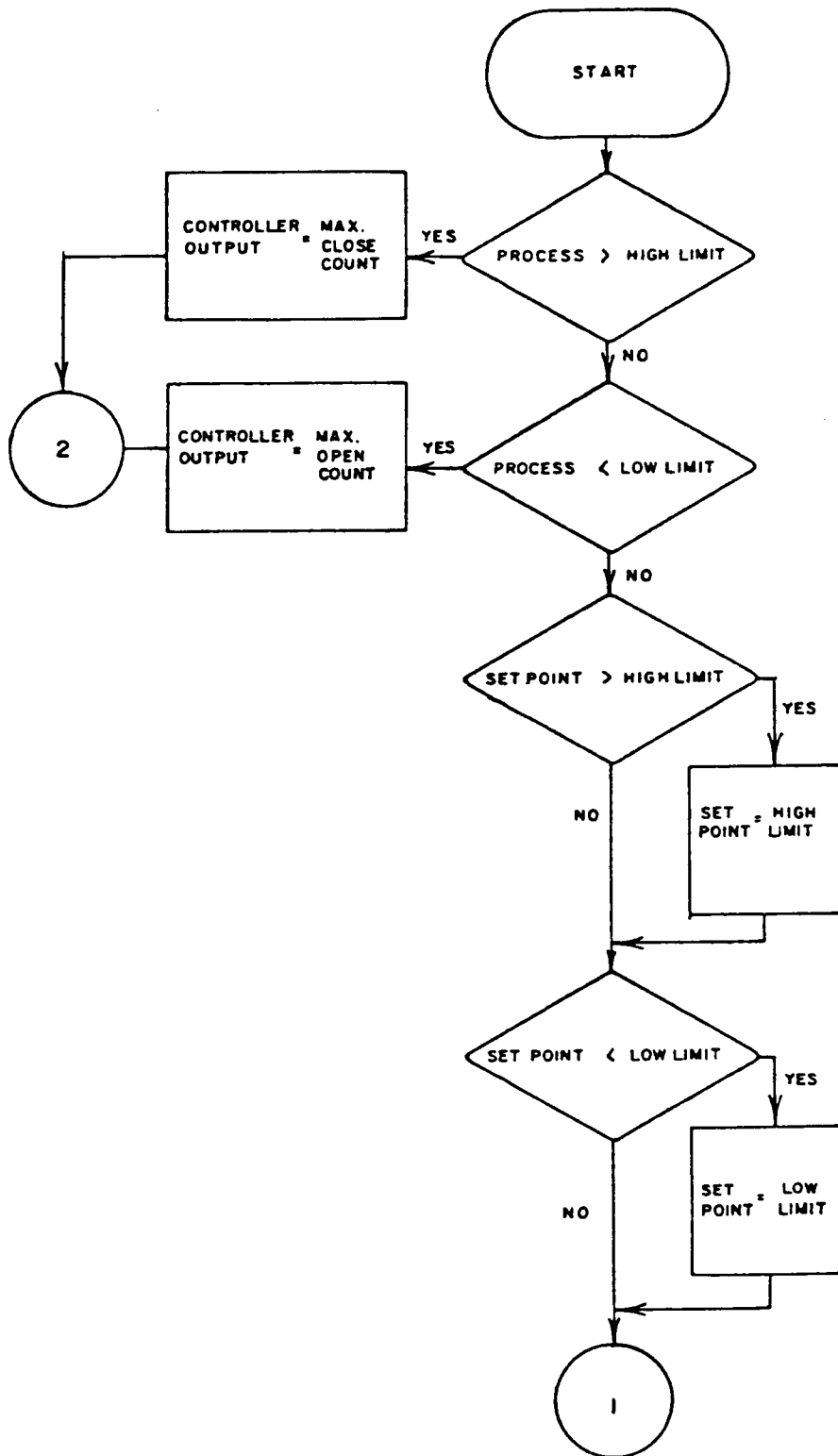


Figure 3-10. Valve controller flow diagram.

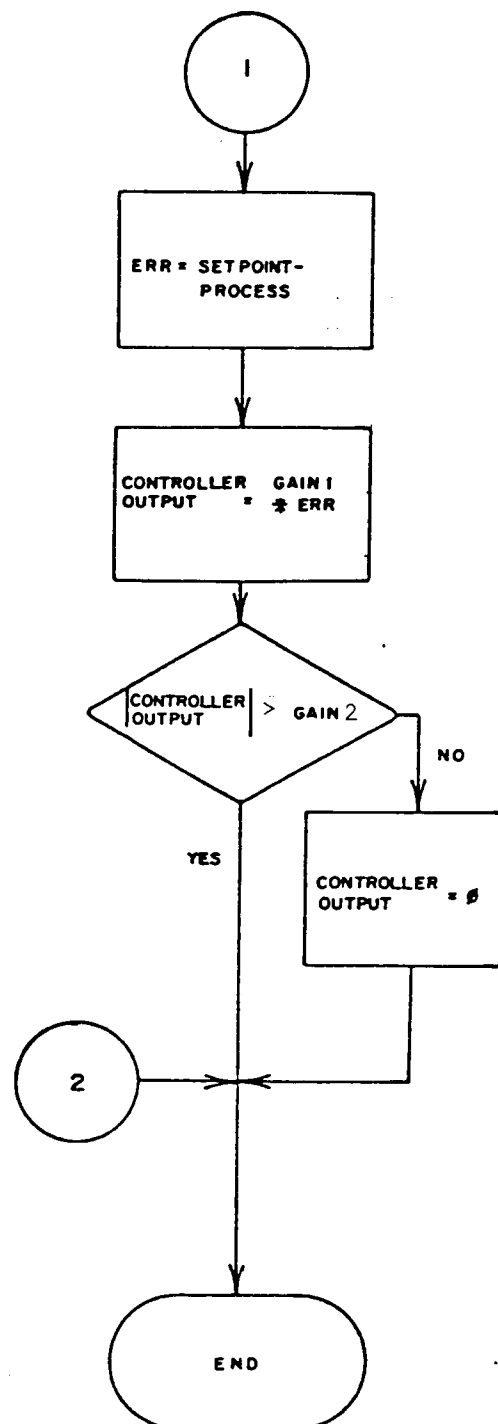


Figure 3-10 (continued).

involves setting initial conditions before the first pass through the software. Since the valve control algorithm is in the "velocity" form (its output is not a setpoint but rather a requested change), there is no summation term to initialize. However, the setpoint must be initialized to the position of the valve as the transfer occurs for a smooth transition.

Consider the alternatives. Suppose the operator has fired up a boiler and manually adjusted it for a low load. Suppose further that the last time the boiler was used it was at full capacity with high setpoints in all control loops. When the operator switches to automatic, if the setpoint is not reset, the boiler will unexpectedly open to full load and unbalance the system. The setpoint must be initialized in all loops except the outermost. For the boiler, this would be the final pressure controller, set at 150 PSI. This initialization is done in the multiloop controller by the parallel input handler in response to the operator pushing a button requesting automatic control.

Saturation limits. As in the PI controller, the strategy should never change a setpoint beyond a saturation limit. However, this feature has been found to be very useful when tuning controllers by preventing process swings while on line.

Threshold. Due to noise and measurement error, it is very unlikely the valve position will ever be exactly equal to the setpoint for very long, but it may be very close. The addition of an adjustable threshold to prevent relay chatter provides an adjustable gap around the setpoint to account for noise in measurement.

Communication Task

The next major software module to be discussed is the communication task. The two functions of the communication task are to provide an information link to a higher level network controller via the communication network and to provide access (via a high speed teletype) to the controller data bases during startup operations.

The communications monitor (identified as DATA in Figure 3-4, page 35), services the serial line connected to the network or local terminal. Communications with the system are handled on an interrupt driven command/response basis. A command is entered and the controller responds. There are four types of commands: display the value of a variable, update the value of a variable, display the system status, and update the system status.

System status. The status is a concise map depicting which control loops are active and the state of the run/detach switch on the valve station. The command requesting the status is the letter S followed by a carriage return. The communication task would then respond with two 16 bit integers. Each control loop is represented by a single bit. When a loop is in automatic, its bit is set indicating it is on. This map of the controller's state is important for the higher level controller to determine how much flexibility it has in modifying the plantwide operation. The network can request a change in the status (i.e., request a loop be turned off or on) by sending the S command followed by the bit for that loop. For example, to activate the loop corresponding to the third bit in the second status

word, one would type S2,4<carriage return>. If allowed, the system would then turn on that loop. Repeating the same command would turn the loop off.

Current value. To display the value of a particular variable, a similar method is used. Each variable in a control is assigned a number (see Figure 3-3, page 32). In addition, each control loop is assigned a number. To display the values of two variables, type the letter C, a loop number and variable number, followed by a comma and another loop and variable number. For example, to display the values of the first and second variable of the fourth loop, type: C41,42<carriage return>. The controller would then respond with the values.

Updating a value is handled in a like manner. The same variable addressing system is used. To update a variable in the control tables, type a key letter, the address, and the new value. For example, to change the setpoint in the fourth table to 75.5, type: U40,75.5<carriage return>. The communication task would then respond with another carriage return to signify its acceptance.

Serial Communication Handler

The serial communication handler (identified as Datin in Figure 3-4, page 35) is the software module which operates the hardware serial interface. It is the link between the communication task and the serial communication port.

The functional division between the communication task and serial communication handler is important for two reasons:

1. Hardware independence. While computer manufacturers attempt to standardize their product line, variations in products do occur as new models appear. A software system divided into several functional blocks allows changing only the affected segments.
2. Time independence. The serial communication hardware operates several orders of magnitude slower than the processor. By allowing the communication task to write into and read from a buffer in read write memory, the communication task can operate at full speed. The handler can then unpack that buffer by sending the message character by character as the hardware is available on an interrupt basis.

Analog to Digital Handler

The analog to digital handler drives the analog to digital converter to bring the various process values into the computer. Converting the analog to digital reading to engineering units is useful in giving meaning to the values in the various control tables. Both of these tasks are performed by the analog to digital handler.

After the analog signals are conditioned in the valve station to a 10 volt range, they enter the computer via a 12 bit analog to digital converter. The converter used in this work is capable of handling up to 32 input channels and is software supported by the functional block called SCAN (see Figure 3-4, page 35).

The scan routine has a table with four entries for each analog signal:

1. A status word which contains a bit to indicate if the signal requires the square root function, and a filter constant if the signal requires filtering.
2. Two constants for linear extrapolation to convert a binary count to engineering units.
3. A destination address for the result to be stored.

In the scan routine, the converter operates on all active channels. Each binary count is converted to internal floating point representation and, if necessary, digitally filtered or taken to the $1/2$ power (flow signals are usually related to the square of the flow). The values are changed to engineering units by linear extrapolation and stored in the proper control table or buffer.

Parallel Output

Since the process valves in this work are relay driven, the control signals emerging from the computer are digital. The digital out lines are buffered in the valve station and close appropriate increase or decrease relays. The parallel out handler (identified as PAROUT in Figure 3-4, page 35) collects the digital output from the valve control loops, packs them into the correct orientation, and sends them out of the computer via the parallel I/O card. A second feature of the parallel out handler is local display of the system status. As mentioned earlier, the system status is the composite of the auto/manual status of all control loops. The status is displayed on the valve station by appropriate LED's. For example, each valve control cell operates in manual until the auto bit is set by the handler.

This operation activates the auto light and engages the automatic circuitry. Higher level control loops are also indicated on the valve station by LED's in a similar manner.

Parallel Input

The operator can affect the auto/manual status of individual loops locally with pushbuttons incorporated in the LED's. By pressing a particular button, the operator can request a change in the status of that particular loop. The parallel in routine (identified as PARIN in Figure 3-4, page 35) handles these requests. Whenever a button is pushed, the handler checks the status of that loop. If the loop is currently active, the routine turns not only that loop off, but also any other appropriate loop. For example, if the operator turns off a valve control loop, the routine will also turn off the associated flow control. Alternately, if the loop is currently off, the routine checks for necessary active loops (i.e., active valve control before flow control) before setting parameters for bumpless transfer and turning the loop on.

Cold Start Routine

There is one software module which does not appear as a task or a handler since it is executed only once--the "cold start" routine. Cold start refers to activating a computer from a power off condition. For the multiloop controller, a cold start involves resetting all volatile memory and hardware devices.

Control tables. When the power is off, the read write memory used in the multiloop controller is lost. The cold start routine must

reset the control tables by copying a list of saturation limits, tuning parameters, and setpoints from nonvolatile memory into the control tables.

Status and I/O buffers. The system status, also in read write memory, must be reset.

Time and vector clocks. The real time monitor functions must also be reset for a smooth return to operation. The date and time clock must be zeroed and the vector clocks must be given a nonzero value dispersed over the first few seconds to start all tasks without immediately overloading the entire system.

Registers and hardware. Finally, the registers and hardware devices must be initiated as required in preparation for going on line.

When all initialization is complete, the cold start routine branches into the background load calculation and allows the real time monitor to take control.

CHAPTER 4

STEAM DRIVEN CHILLER CONTROLLER

This chapter describes the development of the control strategy for a 1000-ton steam driven chiller. As mentioned earlier, this is an application located in an IBM facility in Kingston, New York. The entire chilled water load, required for office space conditioning, is provided by three electric and four steam driven chillers located in the boiler house. The steam driven chillers are a main consumer from the boilers, provide chilled water for the plant, and reject waste heat to the cooling tower. The chillers were selected as an initial subsystem to be controlled because of the relatively high dollar/energy savings potential and because their operation affects so many other energy subsystems.

The chillers are conventional mechanical refrigeration units, as shown in Figure 4-1. In operation, the chillers must provide a required flow of 44° F chilled water while avoiding physical constraints.

The chiller operation can be described as follows: water returning from the plant is chilled by evaporating refrigerant in a heat exchanger. To convert the refrigerant back to a cool liquid for reuse after evaporation, it is compressed into a hot gas, condensed into a warm liquid with water from the cooling tower, and further cooled by evaporating a small portion of itself in passing through an expansion valve. The two physical constraints which must be

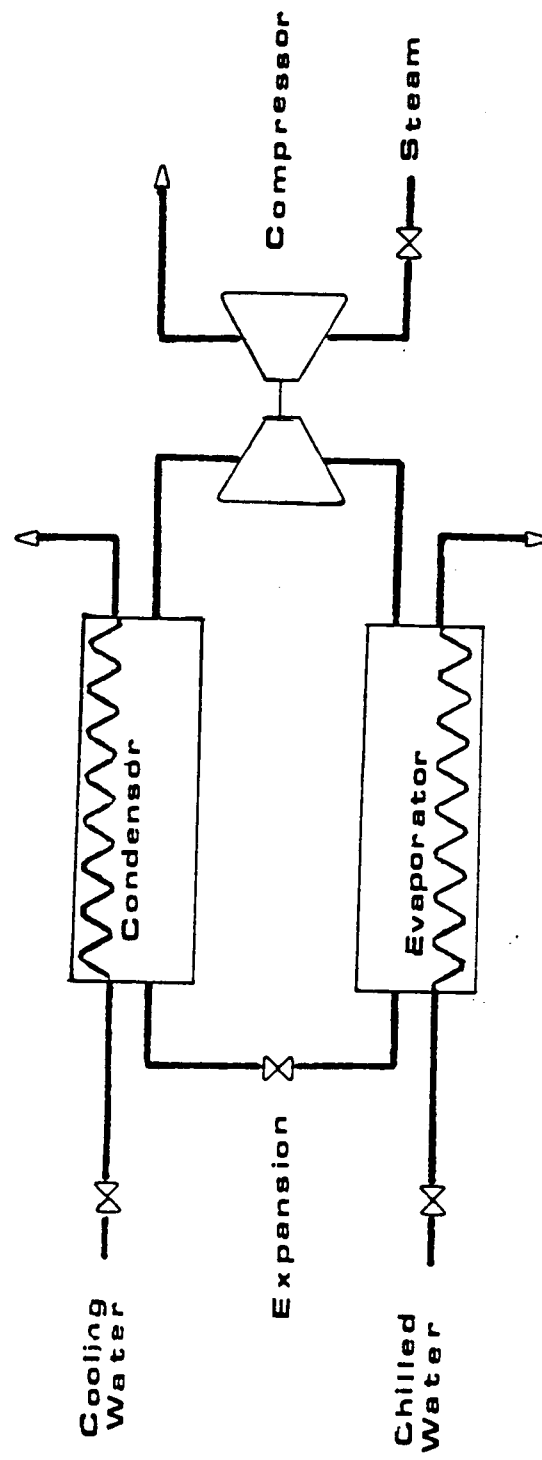


Figure 4-1. Steam driven chiller.

avoided for safe operation are freezing conditions in the evaporator and the gas flow instability characteristic of all axial flow compressors, called surge.

The primary goal of the control strategy was the development of a combined chiller control/energy management control system to reduce the energy consumption (i.e., the steam required for a given load) while meeting the demands of the operating environment.

The Initial Control System

The initial chiller control system (see Figure 4-2) was very simple. A conventional analog controller adjusted the turbine governor to maintain a desired chilled water temperature. This arrangement effectively controlled the temperature and therefore met the load demand. However, normal operation resulted in considerable waste. To eliminate freezing conditions under high loads in the evaporator, the compressor inlet damper was operated partially closed. While this feature offered safe, stable operation, it resulted in an irreversible energy loss at all load conditions.

The Control System Design Principles

The fundamental thesis through all phases of this project is "the control problem and the control solution can both be understood entirely in terms of the structure of the process itself." The chiller control strategy and its development is an excellent example of this principle which can be restated as: a clear understanding of a process will illuminate both the method of its control and a control strategy to implement that method.

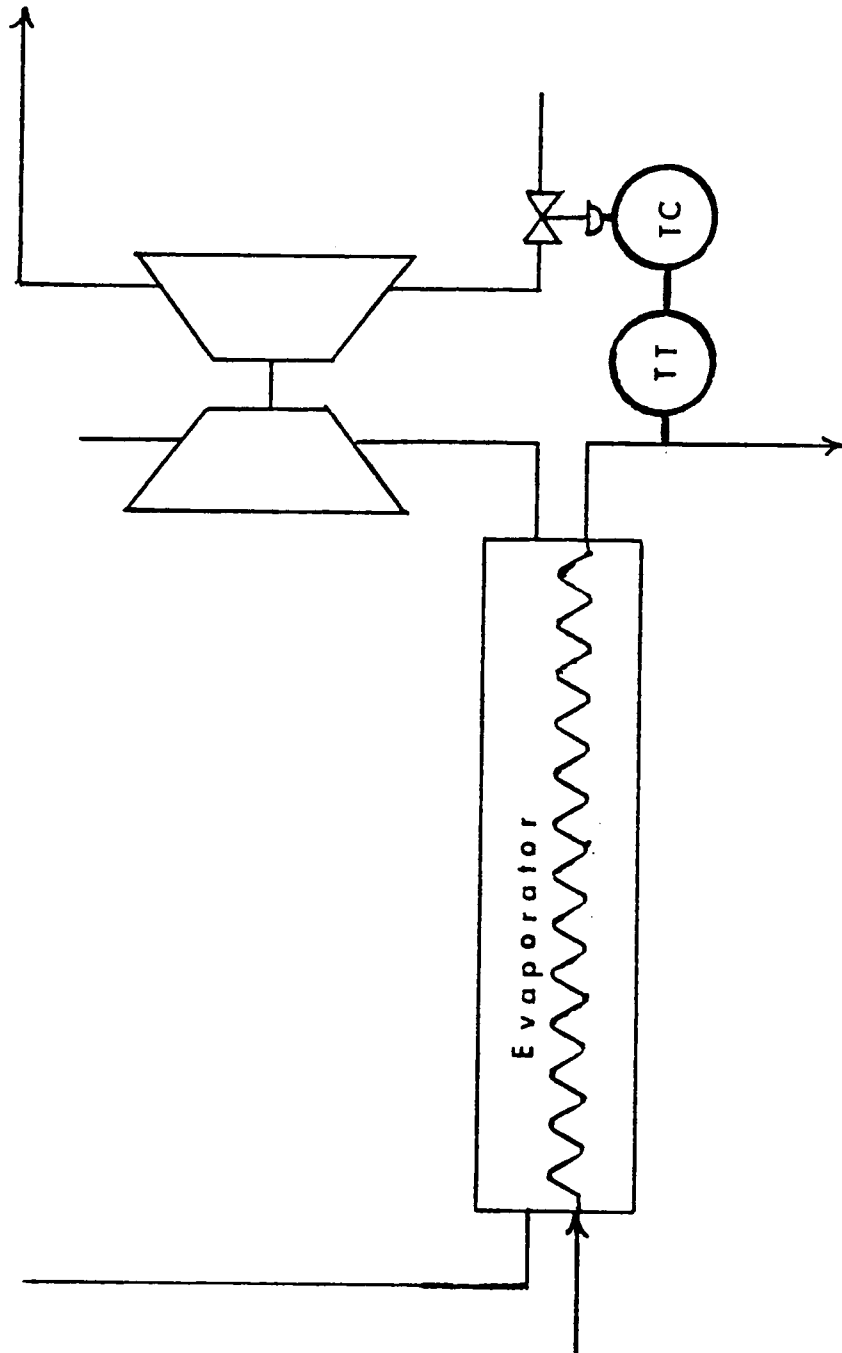


Figure 4-2. Initial chiller control.

The Overall Structure of the Process

From a control point of view, the process analysis is basically a systematic procedure for classifying the known process variables. The general framework of any process variable is as follows.

Dependent Variables

The dependent variables are only those process sensors which respond to changes in the actuator positions. They include:

Control variables. The control variables are sensor signals which are of prime interest in the control study. They are the variables which must be controlled to desired target values or set-points. For the chiller they are:

1. The chilled water temperature.
2. The chiller BTU load.
3. The refrigerant temperature.
4. The compressor load.

Intermediate variables. The intermediate variables are sensor signals which are of secondary importance to the control strategy. These variables represent the state or internal condition of the process. For the chiller they include:

1. The compressor RPM.
2. The cooling water flow.

Independent Variables

The second major class of signals are the independent variables, including the variables directly manipulated by the control strategy

and those variables which are unaffected by the operation of the process. These variables include:

Manipulated variables. The manipulated variables are signals which can be directly adjusted by the computer such as control valves, solenoids, stepping motors, and heaters. These variables represent the "handles" the operator or control system uses to effect changes in the process operation. For the chiller they are:

1. The signal to the chilled water valve.
2. The signal to the cooling water valve.
3. The signal to the steam valve.
4. The signal to the compressor inlet damper.
5. The signal to the hot gas bypass damper.

Disturbance variables. The disturbance variables are those sensor signals which measure variables external to the process and do not respond to changes in any of the actuator positions. These signals can be used to indicate changes which will disturb the operation of the unit. For the chiller the disturbance variables include:

1. The chilled water return temperature.
2. The cooling water inlet temperature.

This signal information about the chiller is summarized in Figure 4-3 which divides the variables into the four groups: control variables, intermediate variables, manipulated variables, and disturbance variables.

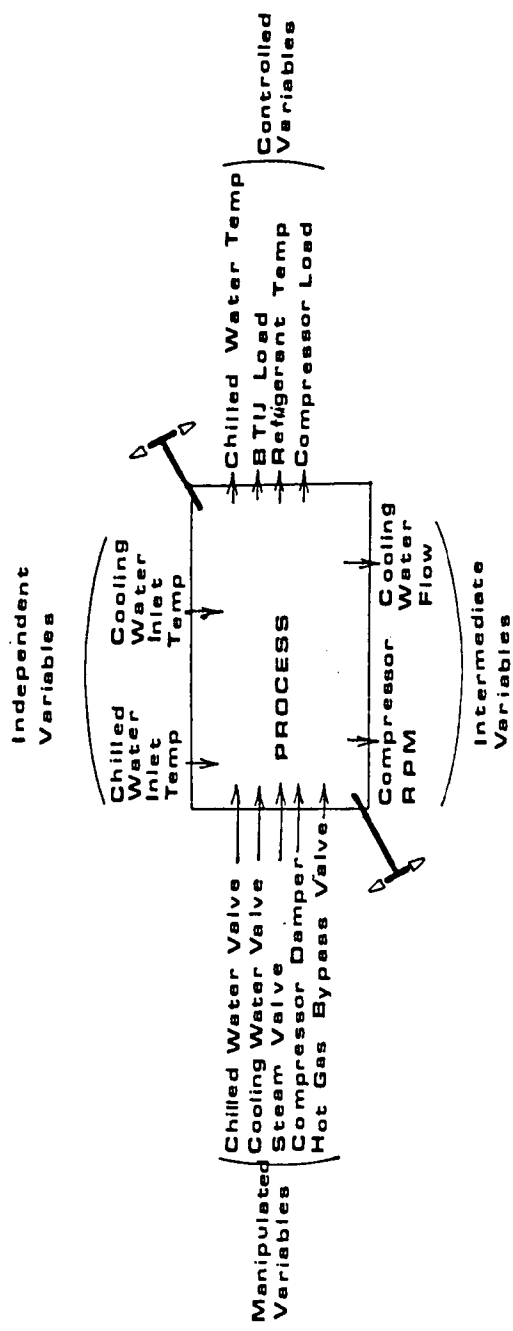


Figure 4-3. Chiller variables.

The Overall Structure of the Strategy

An overall structure of the strategy can also be defined independently of any specific application or theoretical foundation on which the strategy is based. The overall structure consists of only two components:

1. Feedback control. The control action of a feedback element is based on a dependent process variable. The control strategy adjusts the manipulated variable in an attempt to eliminate a measured control error.
2. Feedforward control. The control action of a feedforward element is based on an independent process variable. The control action is an attempt to adjust the process before an error in the control variables appears.

All control strategies can be described within this two-component structure. Note that both the feedback and feedforward elements adjust the manipulated variables; the difference between the two is the process information used to make the adjustment.

The feedback elements use the dependent variables, as follow, for the chiller:

1. The chilled water temperature control.
2. The BTU load control.
3. The refrigerant temperature control.
4. The compressor load control.
5. The cooling water flow control.
6. The chilled water flow control.
7. The compressor RPM control.

The feedforward elements use the independent variables which, if required, would be associated with:

1. The manipulated variables.
2. The chilled water inlet temperature.
3. The cooling water inlet temperature.

The Internal Structure of the Process

The next level in the process analysis is understanding the internal structure of the process, or how the independent and dependent variables are related. This internal structure can be understood in terms of hierarchy of actions and interactions. That is, if a change is made in an independent variable (i.e., a control valve), the interactions are the response of the dependent variables in terms of relative speed and amount of change. A study of the interactions was done and reported by Moore and Green (19). The findings of that report along with relationships found since are summarized here:

1. Modulating the compressor inlet damper can be used to modulate the chiller load and eliminate freezing conditions in the evaporator; however, the damper should not be used because it represents an irreversable energy loss. (The inlet damper is not an inlet guide vane as found on electric driven compressors.)
2. Modulating the steam valve on the steam turbine can be used to change the chiller load. An increase in steam flow drives the compressor harder, reducing refrigerant pressure in the evaporator. The refrigerant evaporates at a lower

temperature and chills the water to a lower temperature as shown in Figure 4-4.

3. Reducing the cooling water flow increases its exit temperature as well as the refrigerant temperature in the condensor and the evaporator. This feature can be advantageously used to prevent the chilled water from freezing in the evaporator as shown in Figure 4-5.
4. Opening the hot gas bypass both increases gas flow through and reduces pressure drop across the compressor. At very low loads, this operation is necessary to prevent surge in the compressor. This need has not been verified on this chiller since it does not operate at such low loads; it has, however, been well documented on other systems.

Internal Structure of the Control Strategy

The fundamental characteristic of a well designed control strategy is that it directly reflects the internal organization of the process. In very general terms, the internal organization of the strategy should be a "mirror image" of the internal structure of the process (50).

Cascade control. The most important concept in designing the internal structure of a strategy is that of cascade control. It ties the two (or more) control systems together so that the output of one controller changes the setpoint of another. The cascade organization provides a very simple framework to connect the process information into the strategy in a mirror image to the process and move the actuator.

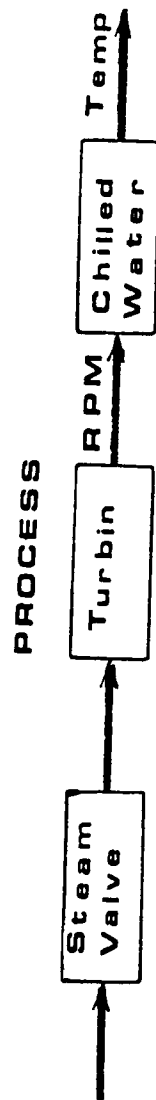


Figure 4-4. Chilled water temperature.

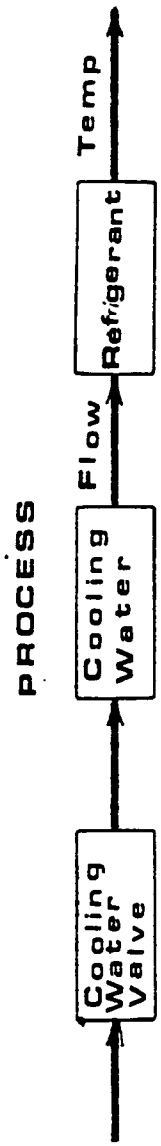


Figure 4-5. Refrigerant temperature.

The mirror image cascade is the guiding rule for developing the internal structure of the control strategy in this work. The basic control structures as reported by Moore and Green (19) reflects this and are summarized here along with modifications developed during startup:

Chilled water temperature control. The chilled water temperature control is a three level cascade strategy as seen in Figure 4-6. The strategy is a mirror image of the process. The outer loop measures and controls the chilled water temperature. The controller, a negative acting PI algorithm, cascades a setpoint into the RPM controller. (It is a negative acting controller since the RPM must be increased to reduce the chilled water temperature.) The RPM controller, a direct acting PI algorithm, controls the speed of the turbine by cascading a setpoint into the steam valve positioner. (It is a direct acting controller since increasing the steam valve position will increase the turbine speed.) The valve positioner adjusts the valve by closing increase or decrease relays in the valve station hardware. There is a direct one-to-one relationship between the internal structure of the process and the internal structure of the strategy.

Refrigerant temperature control. During normal operation, the refrigerant in the evaporator must be kept above 32° F to prevent the chilled water from freezing. The strategy in Figure 4-7 was developed to control the refrigerant temperature. The outer loop, a negative acting PI, has a setpoint of 34° F. During normal operation with the refrigerant temperature above this setpoint, the controller "winds up"

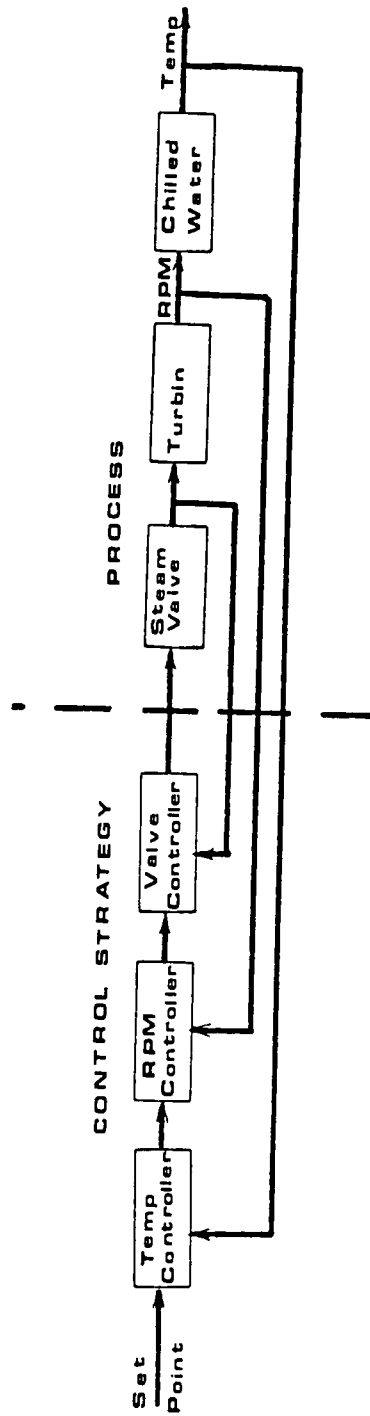


Figure 4-6. Chilled water temperature control.

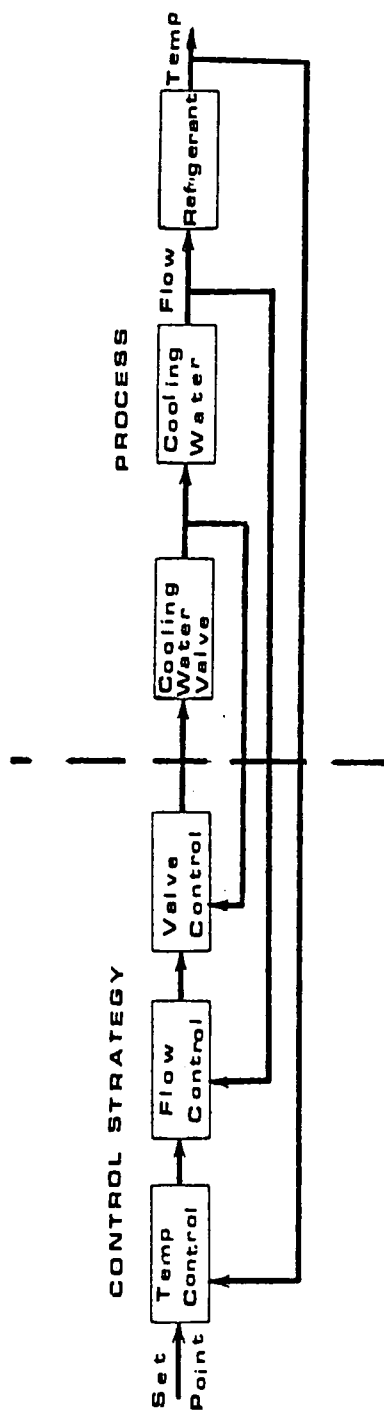


Figure 4-7. Refrigerant temperature control.

to the flow controller high limit and allows free coolant water flow through the condensor. As the temperature dips below 34° F, the controller reduces the cooling water flow setpoint until the 34° F setpoint is achieved. The next inner control element is a flow controller. It is a direct acting PI algorithm and cascades a setpoint to the cooling water valve positioner.

Chiller load control. The chiller load is a measure of the energy removed from the chilled water as it flows through the evaporator. It is important to have a local load controller as a handle for the higher level controller to distribute the overall load among the various chillers as seen in Figure 4-8. The first control element is a steady state feedforward controller. It measures the water temperature across the evaporator and calculates a flow rate for the exact energy loading according to the following equation:

$$\text{Flow} = \text{Load} / (K * \text{Delt Temp}) \quad (4.1)$$

Where Flow = Calculated flow (gpm)

Load = Chiller load (tons)

Delt Temp = Temperature across the evaporator

K = Constant, .0415 tons / (gpm * degree)

The flow controller, a direct acting PI, then adjusts the flow by cascading a setpoint into the chilled water valve positioner.

Compressor surge control. The compressor used in this chiller is an axial flow design. Inherent in all axial flow compressors is a

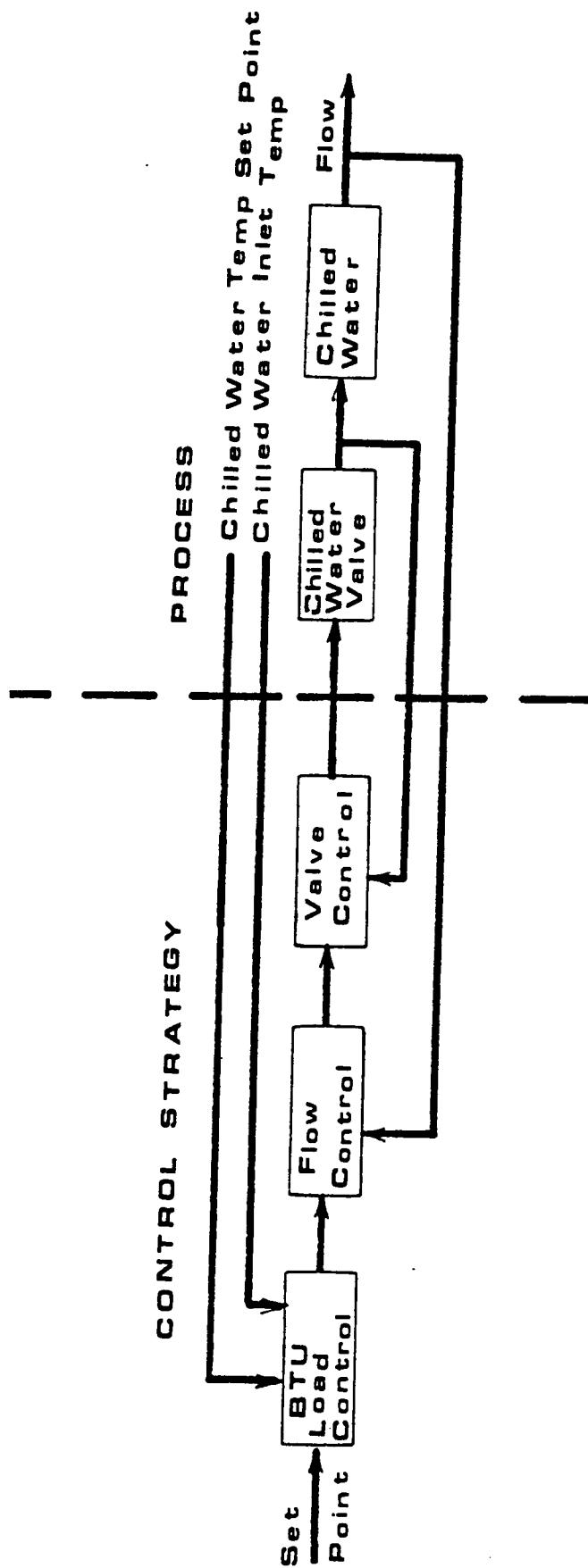


Figure 4-8. Chiller load control.

phenomenon known as surge. Surge is a temporary gas flow reversal at low loads and must be avoided. The principle cause of this behavior lies in the shape of the pressure capacity curve in Figure 4-9 for this type compressor.

As refrigerant flow is reduced for a given RPM, the pressure differential across the compressor is increased until, after reaching a maximum, it begins to drop. When this event occurs, the pressure in the discharge piping exceeds that produced by the compressor and flow is momentarily reversed. Such pulsations can be magnified to destructive proportions very quickly by the response of the compressible gas in the discharge system.

High pressure differential and low flow, characteristics which indicate surge, are representative of low-load operation accompanied by high condenser temperatures or low chilled water temperatures. Since the chiller these tests were run on does not normally operate under those conditions, the surge curve has not been experimentally characterized.

Surge control is straightforward and is based on the estimated surge curve. Surge is indicated by low flow of refrigerant and high pressure drop. Surge can be prevented by increasing the flow and reducing the pressure. The flow and pressure adjustment is accomplished by a bypass which, when opened, allows discharge gas to circulate directly back to the intake.

The surge control strategy, though found not to be required for this chiller, is included for completeness in Figure 4-10. The outer control element has two functions: it calculates a minimum setpoint

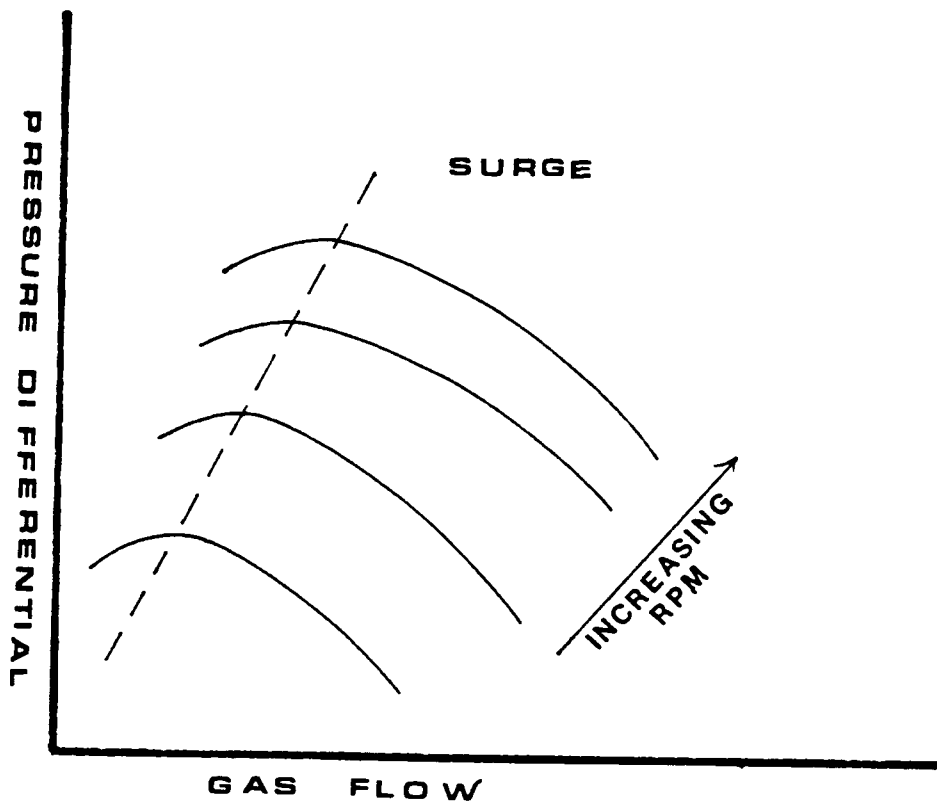


Figure 4-9. Compressor surge line.

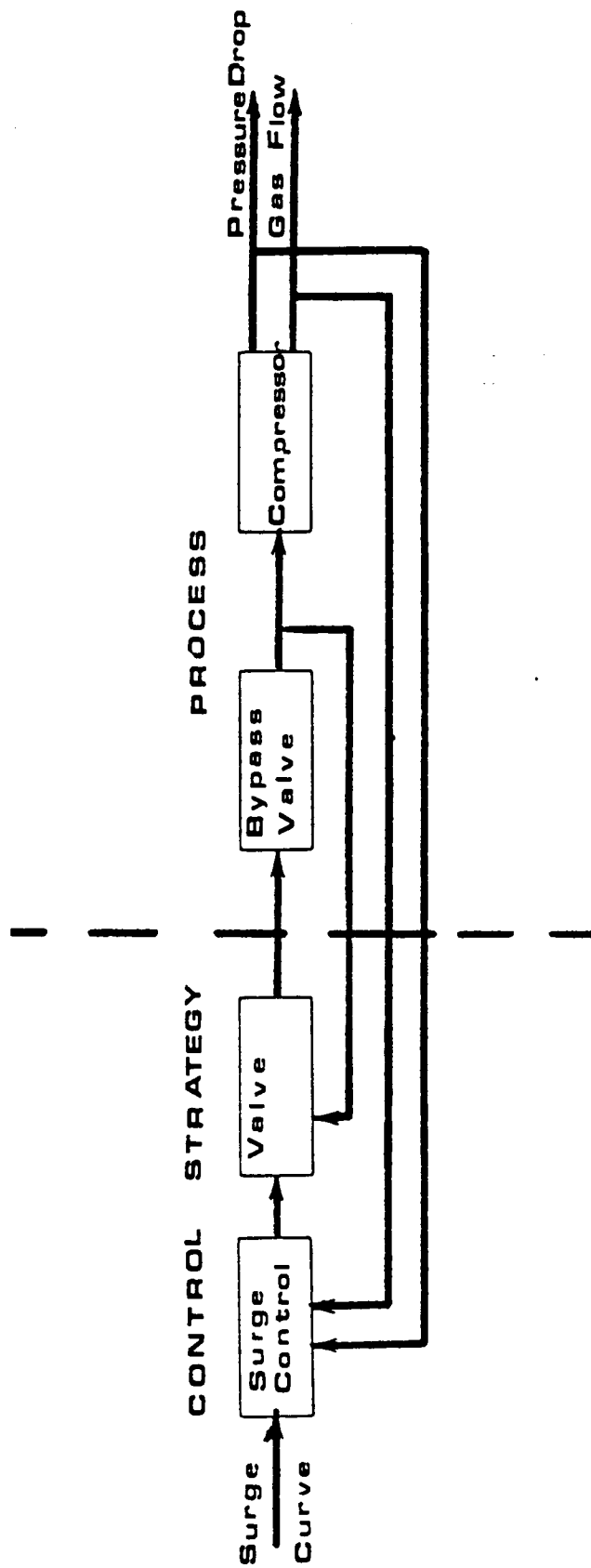


Figure 4-10. Control strategy for compressor surge.

for the gas flow based on the measured pressure drop using a curve fit for the surge line. It then uses a direct acting PI algorithm to control the flow by cascading into the hot gas bypass valve positioner. Under normal conditions, the gas flow is greater than the minimum setpoint, and the controller is "wound down," keeping the hot gas bypass damper closed. As the gas flow dips below the calculated minimum, the PI controller gradually opens the hot gas bypass valve to maintain safe operation.

Complete Chiller Control Strategy

The complete strategy combining all the subprocess controls can best be understood in levels. The lowest level is used for positioning the electric motor driven valves in Figure 4-11. The next level provides flow control on the chilled and condensor water along with RPM control on the turbine in Figure 4-12. The next level adds the higher level controls including water temperature control, refrigerant temperature control, BTU load control, and surge control as shown in Figure 4-13.

Conclusion

The control strategy described in this chapter has been successfully applied to all steam chillers at the Kingston IBM facility. The improvements which have been achieved are tighter control and a reduction in steam usage. Prior to this study, the chiller required around 15 KLBS per hour steam for a nominal full load of 1000 tons. With this control system in operation, the steam

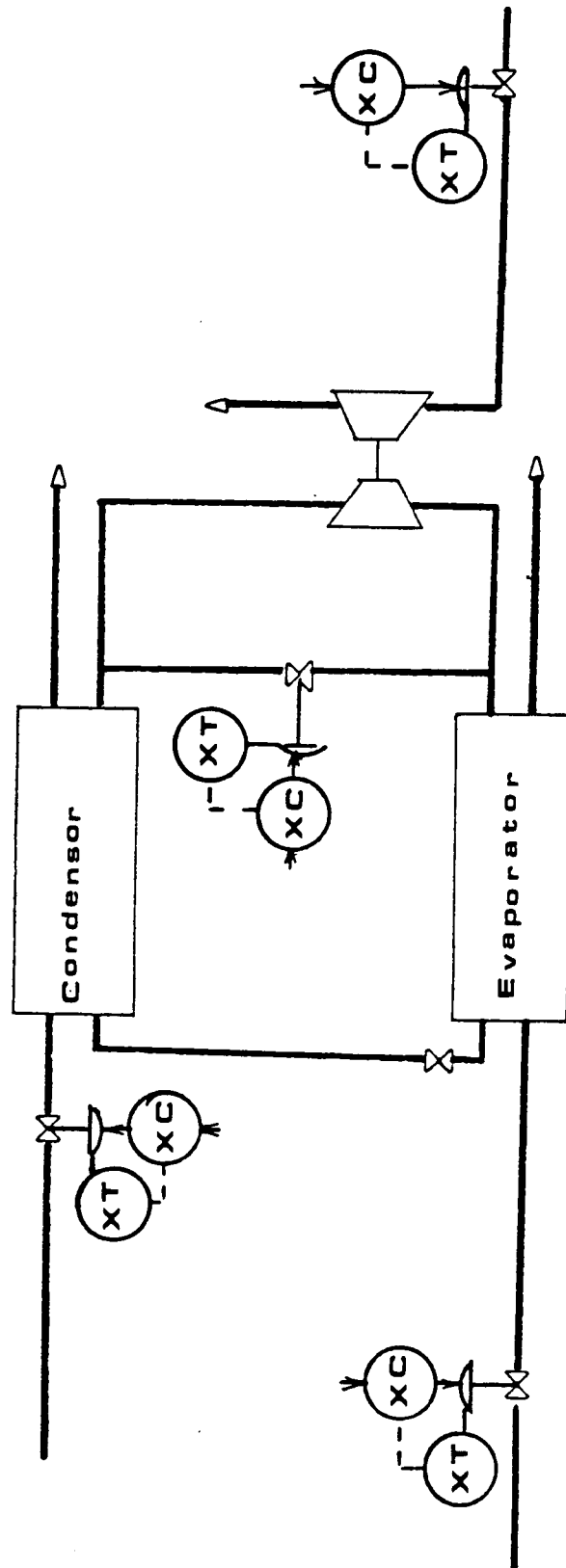


Figure 4-11. Valve position control.

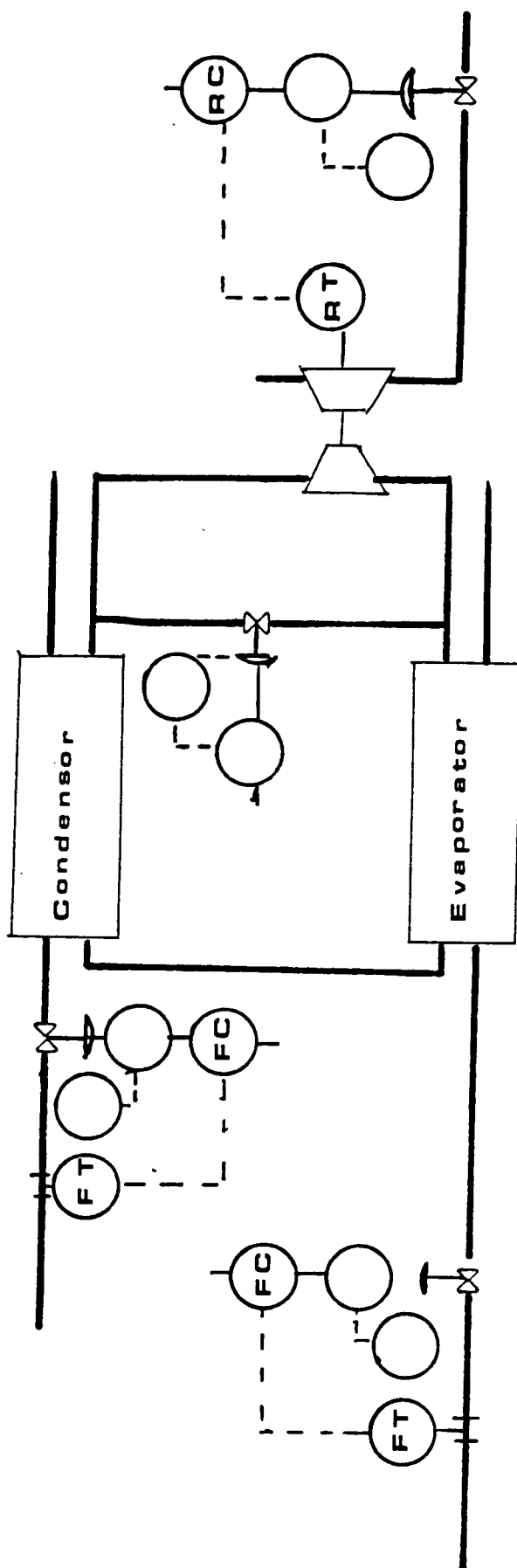


Figure 4-12. Flow and speed control.

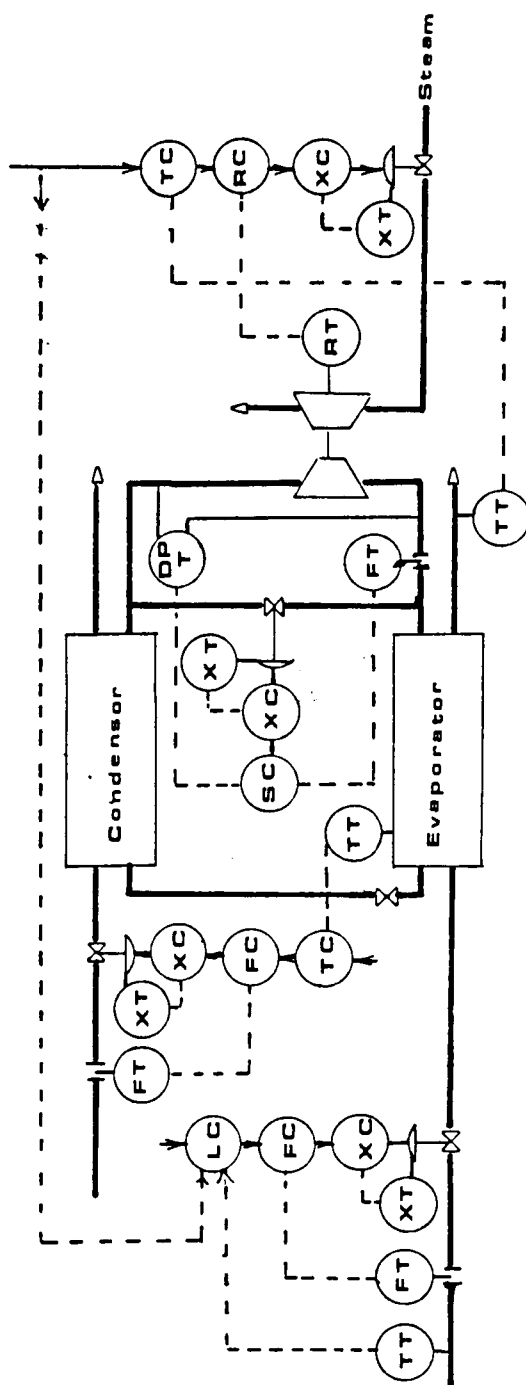


Figure 4-13. Complete chiller control.

usage has been reduced to 6 to 8 KLBS per hour for a full load, depending on the outside conditions and cooling tower loading.

This dramatic reduction in energy consumption is a direct result of:

1. Designing a strategy with attention to the complete thermodynamic condition of the refrigeration cycle. The strategy incorporates the constraint control into the overall system.
2. The strategy provides tighter controls and can be operated closer to the optimum.

It should be noted that tight control may not be necessary or even important; however, when combined with a total strategy which continually moves the process to an ultimate constraint, it becomes very important. In the case of the chiller, the chilled water temperature control was improved from $\pm 3^{\circ} \text{F}$ to $\pm 0.5^{\circ} \text{F}$. In terms of the chilled water users, the improved control was hardly noticed, but in terms of safe, consistent operation of the unit, it translates into a considerable improvement.

CHAPTER 5

OIL FIRED BOILER CONTROLLER

This chapter describes the control strategy developed for a 40,000 pound per hour steam boiler. There are many commercial combustion control systems available today, and several papers have been devoted to various control strategies (64). As a result, the work described in this chapter does not attempt to be a complete description of the boiler control field. Instead, the work is aimed at combining some conventional strategies with a few refinements tailored to the operation of a medium small boiler in a distributed system.

As mentioned previously, there are four identical boilers in the Kingston boiler house. They are connected to a common header and supply 150 pound per square inch steam. The steam is used for office space heating in the winter and operation of the steam driven chillers in the summer. The units are conventional, oil fired boilers (see Figure 5-1).

The boiler's operation can be described as follows: inlet air is drawn in with the forced draft fan and is heated by a steam heat exchanger. The air can then either flow through a chamber in the back wall of the boiler to pick up additional heat or be bypassed directly to the inlet air register. For energy recovery, it is desirable to use as much heat as possible from the walls of the boiler, but not so much as to cool the flue gas below its dew point, with resultant corrosive condensation.

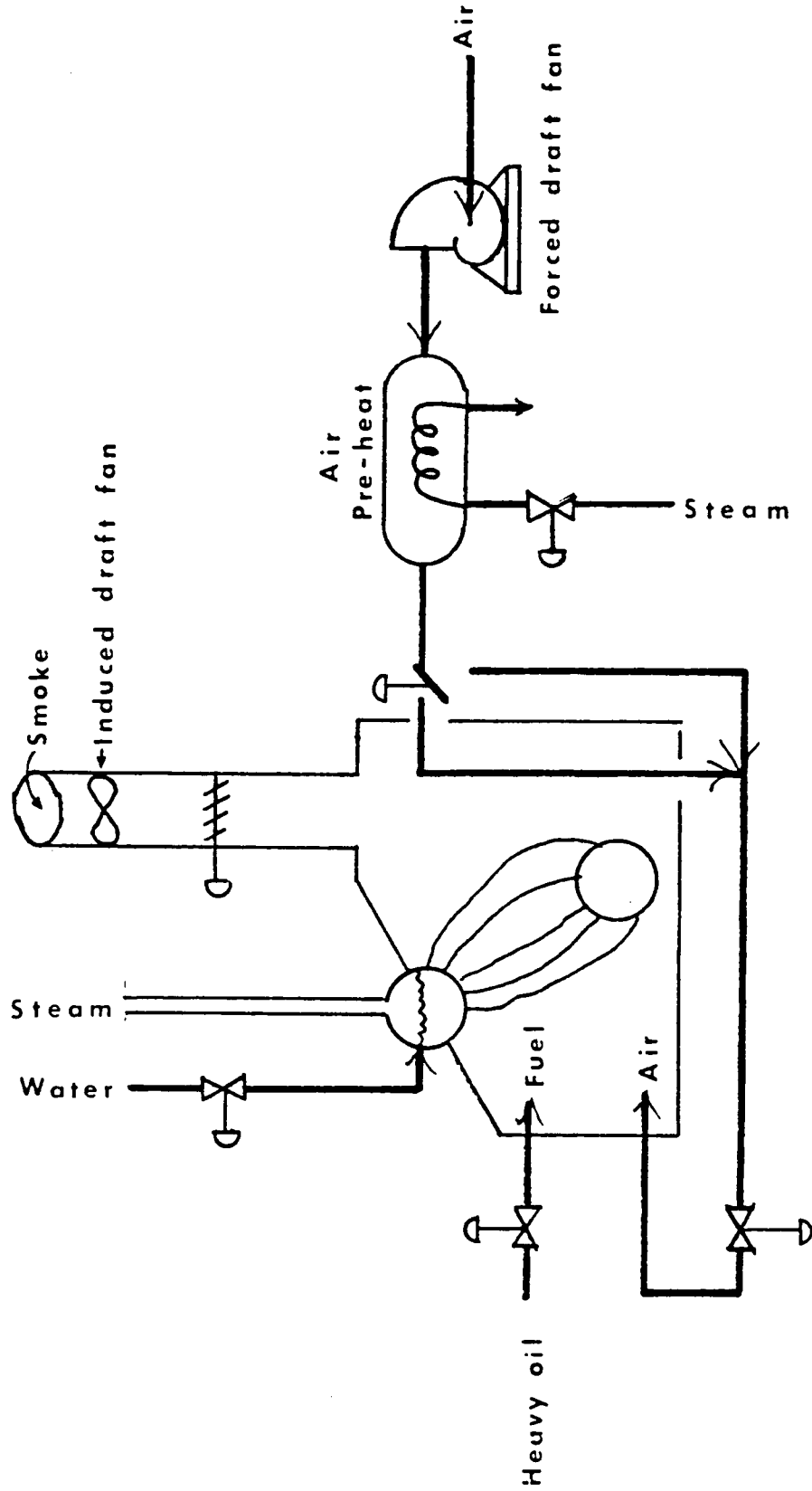


Figure 5-1. Boiler.

The fuel is mixed with the air stream and atomized by a steam blast in the inlet air register. The fuel ignites and passes up around the generation tubes, exchanging heat and producing steam. The combustion gasses are further cooled as they strike the back wall of the boiler and flow up and out of the boiler. Transducers in the stack measure temperature, opacity, and excess oxygen. The ideal flue gas temperature would be just above its condensation temperature, having lost all its heat producing steam and heating the inlet air. The gas should have an oxygen content sufficient to allow complete combustion, denoted by an opacity level in accordance with the environmental protection agency regulations.

Water is valved into the steam drum which is connected to a second drum (the mud drum) by the generating tubes. As the water flows in the generating tubes, it is converted to steam and flows back to the steam drum and into the steam header.

Initial Control System

The boilers at the Kingston facility have, in the past, been the focus of a control study to improve efficiency and reduce stack emissions. That study, performed by IBM personnel, resulted in a hybrid (part analog, part digital) control system using an IBM System 7 minicomputer connected to four boilers as shown in Figure 5-2. In the initial control system, the air/fuel ratio was under direct digital control. Conventional analog controllers were used to maintain the steam drum level and fire box pressure.

While the study did provide a fundamental understanding of the firing control and a control strategy, its implementation was not

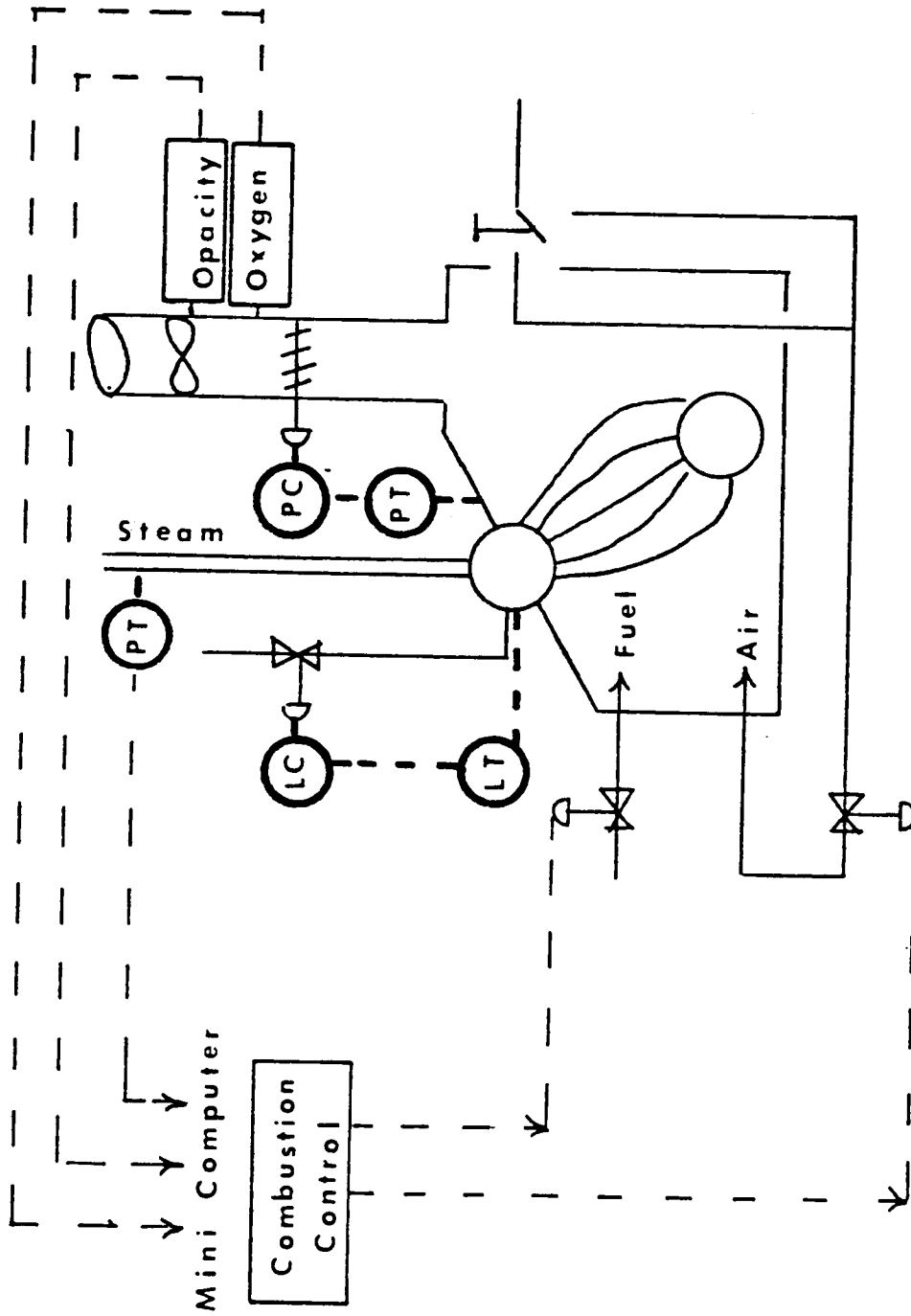


Figure 5-2. Initial boiler control.

completely effective. The control system was representative of those designed before the availability of mini and microprocessors and utilized a rather large central digital control system connected to a number of processes. In addition to the boiler firing control, the entire boiler house was instrumented and connected to the computer for monitoring and logging the total plant operation. The software system was quite large and required extensive disk operations consuming a major portion of the computer's time. As a result, the execution rates for the control algorithms were slow. The fastest rates were on the order of 30 seconds and coupled with a boiler that could become unstable in 5 seconds. As a result, the system had to be operated with an even greater margin of safety (and reduced efficiency) except at fixed firing rates.

Goals of the Control System

The overall objective of the new controller is to further reduce the energy consumption (fuel oil) for a given load while avoiding operational constraints. The specific goals include:

1. Develop a boiler control system which, through its dedicated nature, would have faster response and tighter control (i.e., be able to operate closer to its constraints).
2. Incorporate all boiler controls within the same digital controller to upgrade the steam drum level and fire box pressure controls for more uniform operation.
3. Develop and include a strategy for minimizing the amount of steam required by the air preheater, while avoiding flue gas condensation in the stack.

Process Variable Analysis

As in the previous chapter, the control strategy description will start with a listing by categories of the process variables.

Control variables. The control variables are sensor signals of prime interest in the control study. They are the variables which must be controlled to desired target values or setpoints. For the boiler they are:

1. The steam header pressure.
2. Flue gas opacity.
3. Flue gas temperature.
4. Steam drum water level.
5. Fire box pressure.
6. Inlet air (at the combustion register) temperature.

Intermediate variables. The intermediate variables are sensor signals of only secondary importance to the control strategy. They represent the state or internal condition of the process. For the boiler they include:

1. Fuel flow.
2. Air flow.
3. Water flow.
4. Steam flow to air preheater.
5. Flue gas oxygen content.

Manipulated variables. The manipulated variables are signals which can be directly adjusted by the computer such as control valves, solenoids, stepping motors, heaters, etc. They represent the "handles"

the operator or control system uses to effect changes in the process operation. For the boiler they are:

1. The signal to the fuel valve.
2. The signal to the inlet air (forced draft) damper.
3. The signal to the stack (induced draft) damper.
4. The signal to the water valve.
5. The signal to the steam preheat valve.
6. The signal to the air bypass damper.

Disturbance variables. The disturbance variables are those sensor signals which measure variables external to the process and do not respond to changes in any of the actuator positions. These signals can be used to indicate changes which will disturb the operation of the unit. For the boiler the only variable is:

1. The steam flow.

Internal Structure of the Process

At the next level of description, the process can be studied along lines of action and interaction between the variables.

Air-fuel combustion control. In a conventional strategy (see Figure 5-3), the steam pressure is controlled by modulating the air flow and steam flow in unison. This coordination is accomplished with one controller whose output goes to both the fuel flow and air flow controllers. Quite often, for boilers with varying feed quality, an oxygen analyzer in the stack is used as a signal to trim the air flow to reduce the amount of excess air. Sufficient excess oxygen

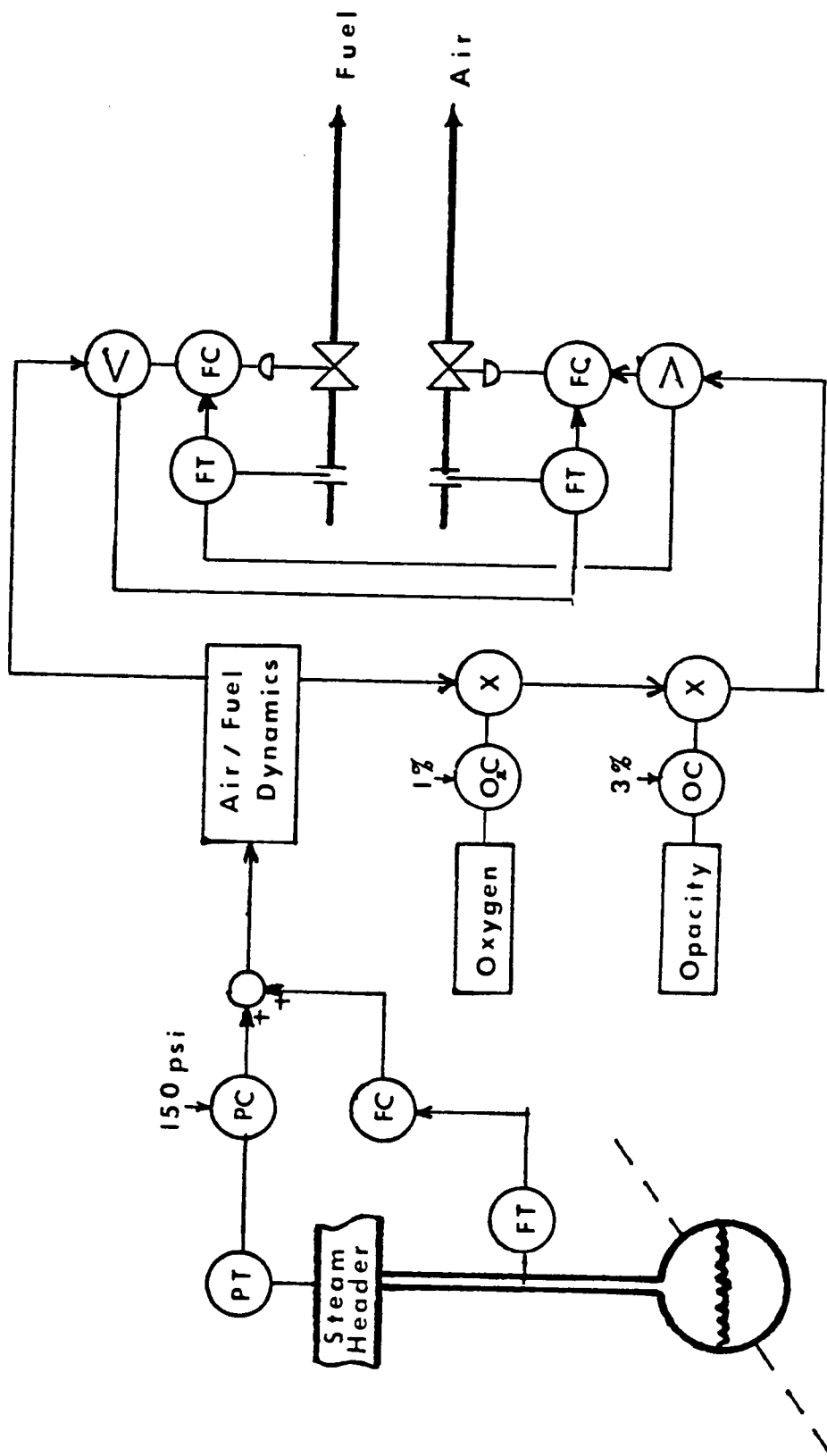


Figure 5-3. Combustion control.

(1 to 3 percent in the flue gas) is required for complete combustion (no smoke); however, too much excess oxygen results in a cooler flame and energy loss to gas dilution. In this system, a PI controller is used to control the excess oxygen. The controller output is not a setpoint, but rather an additional multiplier to the air flow controller setpoint.

Because a demand change from the plant is first noticed as a change in the steam flow rate, a feedforward controller (a lead algorithm in this application) is usually added by using the steam flow signal to alter the firing rate. This feedforward element closes the energy balance. The steam flow is a measure of the energy output which the controller relates to the energy (firing rate) input.

Combustion constraints are usually incorporated into the controller by "flow tie back" elements. The air flow and fuel flow signals are calibrated with a function generator to provide identical readings at steady state operation with minimum excess air. The flow tie back elements use the calibrated signals as setpoint limits to insure the air flow controller never reduces the air flow below the measured fuel flow or the fuel flow controller never increases the fuel flow above the measured air flow.

Additions to the Combustion Control Strategy

Three additions were made to the conventional control strategy: 1) an opacity controller, 2) an air lead on the air-fuel ratio controller, and 3) a slave pressure controller for multiple boiler operation.

Opacity control. The opacity control provides a final air trim to control the opacity (smoke) in the stack gasses. Its implementation was performed by IBM personnel. The opacity controller works similarly to the conventional excess oxygen controller. The sensor is placed in the stack and measures the attenuation of a light beam passing through the stack gasses. The signal is fed to a PI controller with a setpoint in accordance with EPA requirements. The output of the controller is a ratio multiplier for the air flow setpoint. The opacity controller offers these advantages over excess oxygen control:

1. The opacity is one of the measurements required by the EPA for emissions control.
2. Compared to the opacity transducer, the conventional excess oxygen analyzer has a slow response. The faster responding opacity signal allows improved control through quicker action and permits operation closer to the constraints.

Air lead. A second addition to the combustion control was an air lead element to the air-fuel ratio control. In a conventional strategy, the pressure controller produces a common signal to both the fuel flow and the air flow controllers. It is a strategy well suited for boilers with fixed or slowly changing loads. Many small boilers, including those at Kingston, operate over a wide range and must be able to rapidly change firing rates. For example, it is quite common for a boiler at Kingston to change from a 20 percent load to full load within one half hour each morning. To accommodate these load changes, some dynamics were added to the air-fuel ratio

to insure safe (i.e., excess air) combustion during these load shifts.

As the demand of the plant increases, the steam pressure controller increases the firing rate (i.e., increases the air and fuel flow). A conventional strategy would ramp the air and fuel flow setpoint at an identical rate; however, a common ramp rate may cause incomplete combustion if the air flow increases slower than the fuel flow. To insure safe combustion, the air lead element adds an offset (an operator adjusted amount) to the ramp to increase the air flow setpoint. (To smooth out the transition corners, the offset is first passed through a filter as shown in Figure 5-4.) The air lead offset is continued as long as the setpoints are ramped. When the new demand is reached, the offset is zeroed and passed to the filter to provide a smooth transition into the new steady state firing.

A demand decrease is handled similarly. The fuel flow setpoint is ramped down at an operator adjusted slope. As before, the air flow setpoint is calculated by adding the filtered air setpoint "boost" to the fuel setpoint ramp.

The steam pressure "gap" controller. The third addition to the conventional combustion control strategy involves multiple boiler operation. The Kingston facility, like many others, has several (four) boilers connected to a common header. This arrangement allows a greater range of operation and ease of maintenance; however, it does present an interesting control problem. Unlike the chillers which can independently control their chilled water temperatures, the boilers must be controlled to a common steam pressure signal. Using multiple feedback

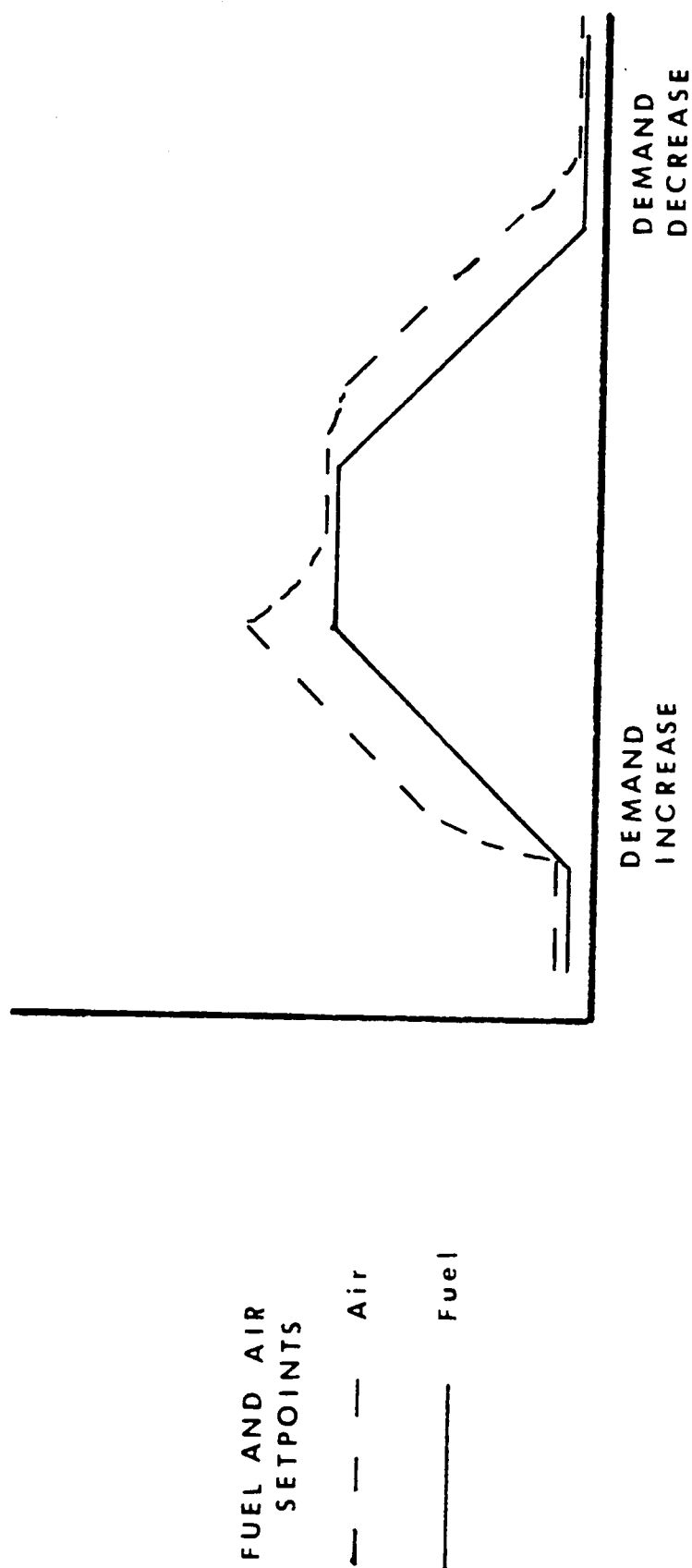


Figure 5-4. Air lead.

control loops (in this case four separate combustion controllers) to maintain a single control variable (i.e., the steam pressure) can produce an unstable overall control system. Each individual control system will fight for control of the variable.

To accommodate multiple boiler operation, a "gap" controller was developed. The gap controller uses a conventional PI algorithm with one exception. Instead of controlling at a unitary setpoint, it controls at a gap or band around a setpoint. In multiple boiler operation, the gap controller is used along with a conventional PI algorithm. One boiler is assigned the master by using the standard PI. The remaining boilers are assigned slaved operation by using the gap PI algorithms.

For two unit operation, the slave boiler is given a small gap. The gap must be wide enough so that during operation the slave controller makes no correction for the small errors in steam pressure, while the master controller adjusts for changing plant loads as seen in Figure 5-5. When the master unit reaches full load or minimum load, the steam pressure will exceed the band. The gap controller then exceeds its band and adjusts the slave unit firing demand until the steam pressure returns to the center, or setpoint, of the gap.

Multiple unit operation is handled similarly with a slave hierarchy (see Figure 5-6). By assigning the first slave a narrow band and each succeeding slave an increasingly larger band, the system responds smoothly to overall load changes.

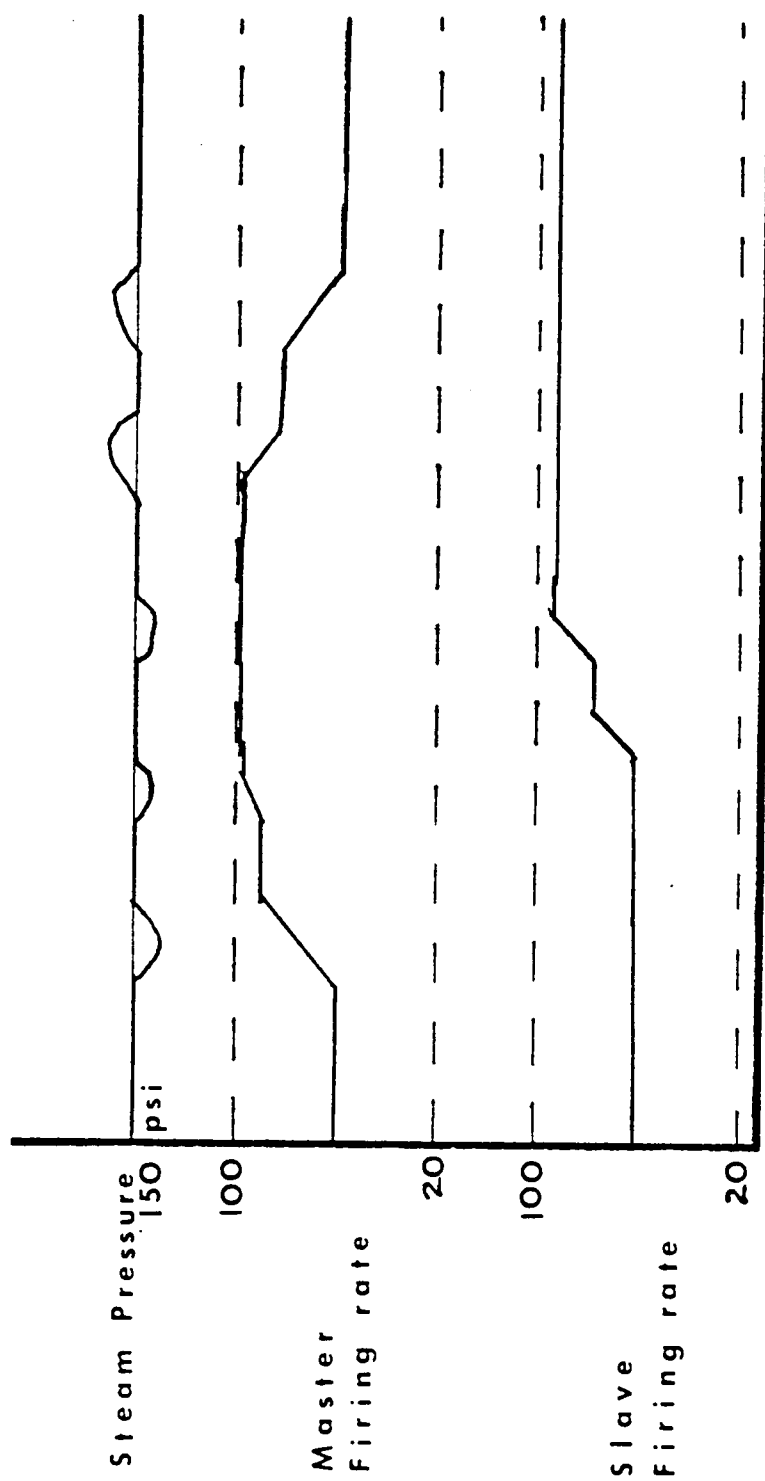


Figure 5-5. Master-slave pressure control.

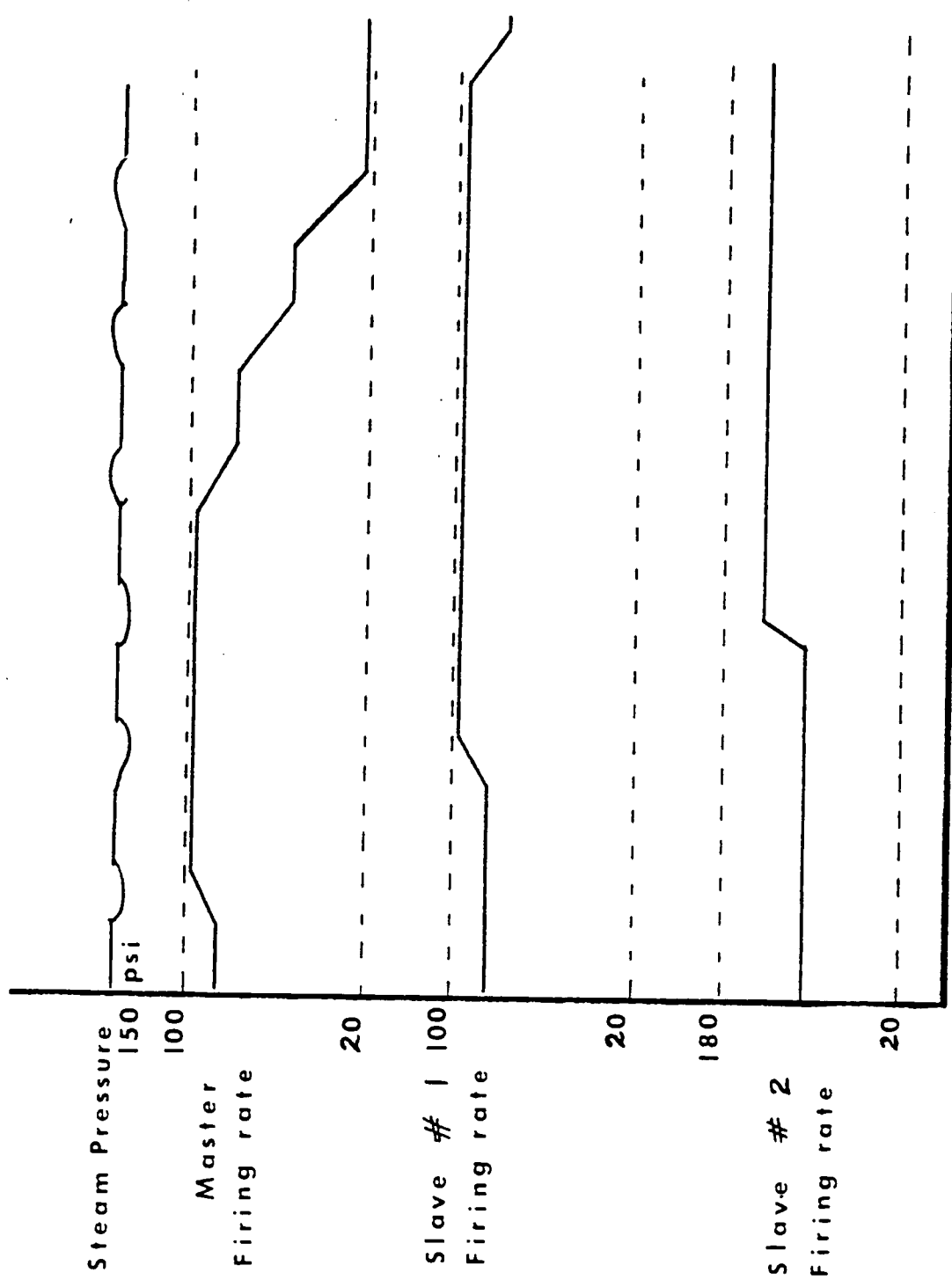


Figure 5-6. Multiple boiler pressure control.

Steam Drum Level Control

The second major control description involves the steam drum level control. As stated earlier, the steam drum is located in the top of the boiler fire box. Feedwater is continually added to keep the drum approximately half full. Water flows down the generating tubes where it is heated and converted to steam. The generating tubes are connected to the mud drum in the bottom of the boiler. The mud drum collects inorganics from the feed water and must be periodically blown clean. The steam flows up to the steam drum, heating the fresh feed water, and into the steam header.

The steam drum level control poses an interesting problem not found in ordinary liquid level control. Because the steam is produced below the liquid level sensor in the steam drum, a percentage of the water below the sensor is in the vapor phase. As the plant demand increases, the steam pressure drops. The reduced pressure causes an increase in the vapor volume and an increase in the drum level. A standard liquid level controller responds by reducing the water flow. In a boiler, however, the combustion control will turn up the firing rate in response to a reduction in pressure and the water flow should be increased to maintain a water balance on the increased steam production. The standard boiler control strategy, therefore, adds a feedforward element to the steam flow signal as shown in Figure 5-7. The steam flow is a direct measurement of the plant demand, and a feedforward element provides a means of closing the material balance based on that demand by relating the water flow to the steam flow. A simple lead controller is used in this application.

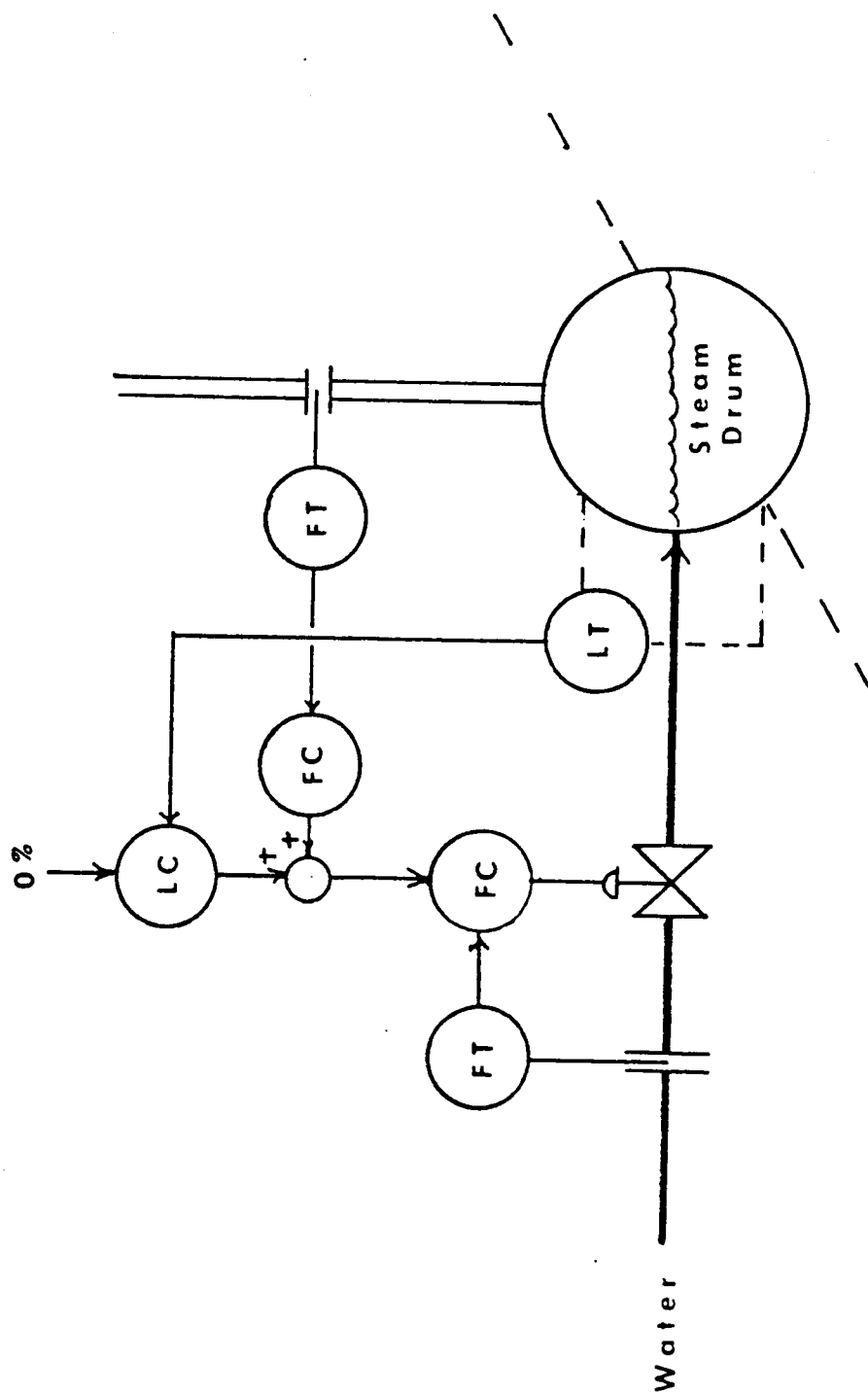


Figure 5-7. Steam drum level control.

As with all feedforward controllers, an outer loop feedback (a PI algorithm) controller is required to trim the result. In this case a level controller (a PI algorithm) provides final adjustment of the water flow to maintain the steam drum level.

Air Temperature Controller

The next major control subsystem provides a means of heating the inlet air to combustion temperature with a minimum of steam. For uniform combustion, the air must be heated to the fuel flashpoint. As mentioned earlier, the air can be heated by either a steam heat exchanger or by removing heat from the flue gas. From a recycle standpoint, it is desirable to remove as much heat as possible from the flue gas. However, excessive cooling of the gas will cause sulfuric acid to condense in the stack.

A two-level cascade was developed to control the inlet air temperature without cooling the stack gas below its dew point as shown in Figure 5-8. During operation, the stack temperature is controlled (a PI algorithm in this application) by modulating the air bypass damper. When the damper is closed, all the air sweeps through a chamber in the wall of the boiler, removing heat. When the bypass is opened, the air bypasses the firebox and flows straight to the inlet register. Another controller (a PI algorithm) monitors the inlet air temperature and cascades into a steam flow controller to trim the final inlet air temperature.

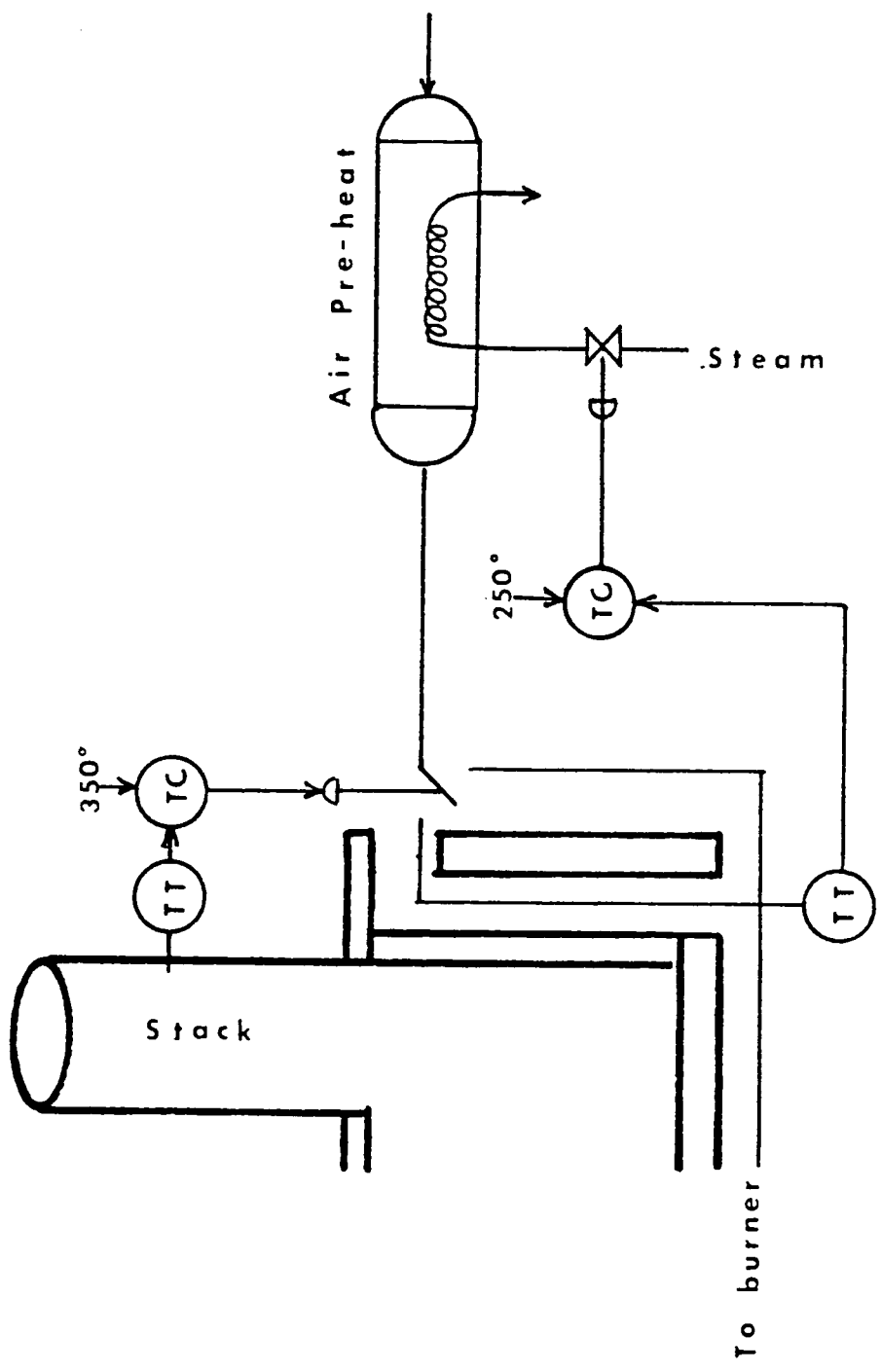


Figure 5-8. Air temperature control.

Firebox Pressure Control

The last major control subsystem is the firebox pressure control. The firebox, because of thermal considerations, is not gas tight. The firebox is operated at a slight negative pressure (minus 1 inch of water) to prevent outflux. To move the gasses through the firebox there are two blowers with dampers. The forced draft blower and forced draft damper are located on the air inlet. The induced draft blower and induced draft damper are located on the stack outlet. The forced draft damper is used by the combustion control and the induced draft damper is used to maintain firebox pressure. The firebox pressure is strongly affected by either damper and the control system accounts for this with a combination of feedback and feedforward controllers in Figure 5-9. Since the air flow into the firebox is controlled by the firing rate, the air flow must be completely independent of the boiler pressure control. As a result, the air flow is treated as an independent variable and assigned a feedforward (lead algorithm) controller. An overall feedback (a PI algorithm) is then used to maintain the pressure.

Conclusion

The control strategy described in this chapter has been successfully applied to a boiler at the Kingston IBM plant site. In terms of implementation, the strategy was assembled using the same hardware-software system as described earlier for the chiller to reduce spares.

In terms of performance, all the control goals were met. Those goals were the tight control of:

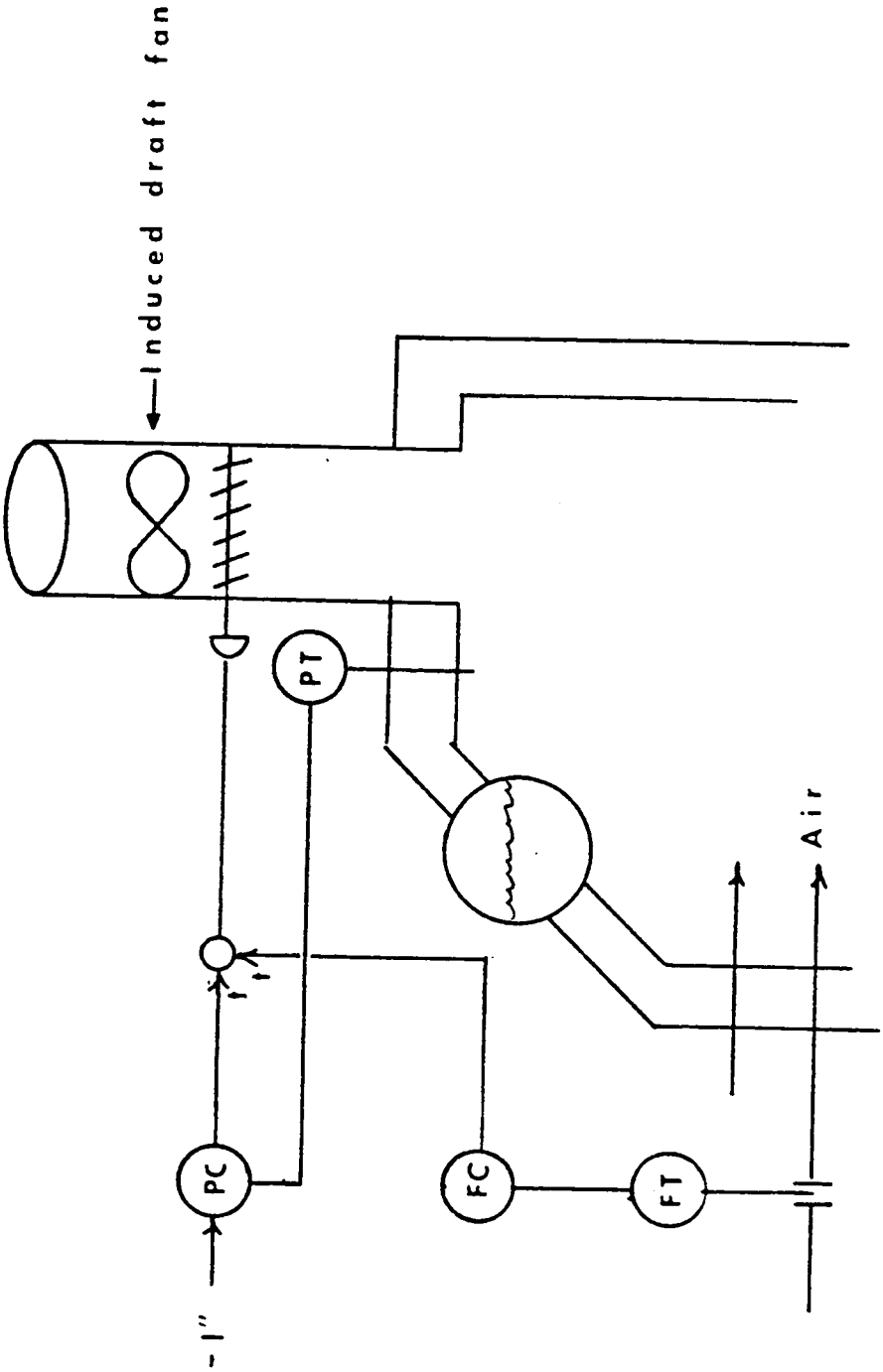


Figure 5-9. Firebox pressure control.

1. Steam pressure. Under moderate load shifts (1 percent change in firing rate per minute or less) the steam pressure is maintained within ± 0.3 PSI. Under major load shifts (i.e., sudden loss of a chiller) the pressure is maintained within ± 2 PSI. This arrangement compares with the initial system which, as a routine practice, was put in manual operation for stability when taking a chiller off line.
2. Opacity. At all loads the opacity (smoke) could be maintained at 3 percent attenuation. A new combustion register to be installed should be able to reduce the opacity for a given excess oxygen content.
3. At all but the lowest firing rates the excess oxygen content in the stack gasses could be controlled within ± 1.5 and 2.5 percent. This oxygen content insures minimization of heat loss up the stack due to gas dilution. At low loads (less than 30 percent firing rate), the excess oxygen setpoint was ramped to 3.5 percent to achieve complete combustion (no smoke).
4. Inlet air temperature. The inlet air temperature was tightly controlled ($\pm 1^\circ$ F) with a minimum use of steam for heating. During startup, the bypass remained full open to allow the stack gasses to quickly rise above the acid condensation point. The bypass would then close, allowing the inlet air to pick as much heat as it could. The heat transfer rate, however, was not sufficient to completely

heat the air, and steam was required to trim up the temperature.

5. Explosion conditions. The air-fuel dynamics and flow-tie back elements successfully prevent explosive conditions due to insufficient oxygen or loss of flame.

As described in the last chapter, the success of the micro based strategy lies within a consistent control strategy sensitive to the overall goal of energy use minimization.

In the boiler control strategy, the need for tight control is even more important than for the chiller described in the previous chapter. As before, tight control in itself may not be necessary or even important. However, tight control is valuable when coupled with a strategy that constantly adjusts the process to an ultimate constraint in performance.

CHAPTER 6

SUMMARY AND CONCLUSION

Previous sections of this dissertation have been devoted to the individual parts (hardware, software, and strategy) involved in a digital control system. The chapters have described, on a basic level, how the system was put together. At this point it is appropriate to address how the work fits in the overall field of applied control.

When this work was conceived in 1975, microprocessors were barely 5 years old and still a curiosity for control engineers. This work assumed the task of not only exploring the capabilities of microprocessors for use in a distributed digital control system, but also determining how process control techniques can be successfully applied to an energy management system.

This dissertation is a description or definition of a microprocessor based control system. While many articles have reported on applications, none describe the details in anything more than general terms. This dissertation offers complete documentation of all phases.

Additionally, this work has shown that process control techniques can be successfully used for an energy management system. In the chiller controller, the strategy added several features to conventional chilled water temperature controls. Because in many cases the most efficient operation will occur at a constraint, two controllers were added to allow operation as close as possible to the

constraints without risk of destroying the equipment. Additionally, the strategy attempts to continually move the unit toward more efficient operation for a given load and has links for overall load distribution for energy minimization.

In like manner, the boiler controller contains more than conventional firing and drum level controls. Controls to insure safe combustion during load changes, controls to minimize energy loss from excess air, and controls to minimize steam usage for air preheat were included. Additionally, a special controller was developed to allow automatic operation of several boilers without contention between control systems.

In terms of software, this work has developed into a statement of the essential parts required for a digital control system. The overall software system is organized along functional lines and each segment is examined for its exact purpose. This organization will serve the individual looking for a place to start in developing process control software or defining the essential parts of a digital control system.

LIST OF REFERENCES

LIST OF REFERENCES

1. Smith, C. L., Digital Computer Process Control, Intext Educational Publishers, Scranton, PA, 1972.
2. Dahlin, E. B., "Designing and Tuning Digital Controllers, Part I," Instruments and Control Systems, Vol. 41, June 1968, pp. 77-83.
3. Dahlin, E. B., Ziemer, R. L., Wickstrom, W. A., and Horner, M. G., "Designing and Tuning Digital Controllers, Part II," Instruments and Control Systems, Vol. 41, July 1968, pp. 87-91.
4. Berger, A. R., and Ragazzini, J. R., "Sampled Data Processing Techniques for Feedback Control Systems," Trans. AIEE (Appl. and Ind.), Vol. 73, Nov. 1954, pp. 236-247.
5. Williams, T. J., "An Historical Background and a Definition of the Field of Computer Process Control," Purdue University Short Course, July 1966.
6. Collins, D. C., "Microprocessor Design and Application," Computer, Oct. 1975, pp. 20-21.
7. Rony, P. R., Larsen, D. G., and Titus, J. A., "Microprocessors: Where Do They Fit?" Computer Design, Jan. 1976, pp. 108-109.
8. Kahne, S., Lefkowitz, I., and Rose, C., "Automatic Control by Distributed Intelligence," Scientific American, June 1979, pp. 78-90.
9. Anderson, L. H., "Distributed Intelligence Microprocessor Systems," ISA Transactions, Vol. 15, No. 2, pp. 127-130.
10. Mears, F. C., "Distributed Computer Control for Stratfjord," Transactions ISA Annual Conference 1978, pp. 109-118.
11. Thor, M. G., Skrokov, M. R., and Weiss, M. D., "Energy Conservation Control in Olefins Plants by Mini and Micro Computers," Transactions ISA Annual Conference 1978, pp. 153-160.
12. Thomson, J. R., "The Design of a Real Time Microprocessor Based Two Phase Flow Data Acquisition System," Proceedings of the ISA Conference, 1979, pp. 477-484.
13. Thomas, A. T., "Architecture and Application of a 12 Bit CMOS Microcomputer," Proceedings of the IEEE, Vol. 64, No. 6, June 1976, pp. 873-881.

14. Parker, S. W., Microprocessors—Hardware, Software, and Applications. Master's Thesis, The University of Tennessee, Knoxville, August 1975.
15. Mallicote, R. E., A Microcomputer Implementation for Supervisory Control and Data Acquisition in Electric Power Substations. Master's Thesis, The University of Tennessee, Knoxville, August 1979.
16. Hamilton, W. L., A Microcomputer Implementation for an Industrial Process Control Regulator, Master's Thesis, The University of Tennessee, Knoxville, June 1977.
17. Bryan, R. N., Discrete Time Regulator for a Digital Heat Control System, Master's Thesis, The University of Tennessee, Knoxville, March 1980.
18. Monks, M. H., A Microprocessor Data Acquisition System for Control of a Variable Air Volume Air Conditioning System, Master's Thesis, The University of Tennessee, Knoxville, December 1980.
19. Moore, C. F., and Green, W., Development and Implementation of Control Algorithms for HVAC Energy Optimization, Applied Control Labs Report, June 28, 1976.
20. Oglesby, D. A., A Digital Simulation of a Dual Duct Air Handling System, Master's Thesis, The University of Tennessee, Knoxville, August 1977.
21. Keane, J. M., A Specific Single Zone Heating, Ventilating and Air Conditioning System Steady State Simulation for Energy Assessment, Master's Thesis, The University of Tennessee, Knoxville, December 1978.
22. Perry, R. H., Chilton, C. H., and Kirkpatrick, S. D., Chemical Engineers' Handbook, Fourth Edition, McGraw-Hill Book Company, New York, 1963, pp. 22.52-22.55.
23. Harmala, A. D., "Benefits of Localized Control with Microprocessors," Computer Design, May 1975, pp. 59-72.
24. Weissberger, A. J., "Distributed Function Microprocessor Architecture," Computer Design, Nov. 1974, pp. 77-83.
25. Zvolner, S. M., "Optimizing Control Strategies for Energy Conservation Through Use of a Computerized Building Automated System," Transactions ISA Annual Conference 1978, pp. 133-140.

26. Gable, M. G., "A Local Network Architecture for Industrial Applications," Transactions ISA Annual Conference 1978, pp. 119-124.
27. LaBarge, D. L., and Levine, A., "Computer Networks in Process Control," Instruments and Control Systems, Dec. 1978, pp. 41-44.
28. Noyce, R. N., "Microelectronics," Scientific American, Sept. 1977.
29. Toong, H. D., "Microprocessors," Scientific American, Sept. 1977.
30. Oliver, B. M., "The Role of Microelectronics in Instrumentation and Control," Scientific American, Sept. 1977.
31. Willmott, T. L., "A Survey of Software for Direct Digital Control," ISA Transactions 1969, Vol. 8, No. 4, pp. 227-234.
32. Pike, H. E., "Process Control Software," Proceedings of IEEE, Jan. 1970, Vol. 58, pp. 87-97.
33. Diehl, W., and Mensch, M., "Programming Industrial Control Systems in Fortran," IFAC/IFIP Symposium, 1968, Toronto.
34. Hawkins, R. D., "The Impact of Software on Future Microprocessor Distributed Architecture," Transactions ISA Annual Conference 1978, pp. 167-176.
35. Dessy, R. E., and Starling, M. K., "Fourth Generation Languages," American Laboratory, Sept. 1976, pp. 34-41.
36. Townzen, D. A., "A Task Scheduling Executive Program for Micro-computer Systems," Computer Design, June 1977, pp. 194-202.
37. Kahn, K. C., "A Small Scale Operating System for Microprocessor Applications," Proceedings of the IEEE, Vol. 66, No. 2, Feb. 1978, pp. 209-216.
38. RMX/80 System Users Guide, Intel Corp., Santa Clara, CA, 1977.
39. Burgett, K., and O'Neil, E. E., "An Integral Real Time Executive for Microprocessors," Computer Design, Vol. 16, No. 7, pp. 77-82.
40. Chien, Y. P., "Multitasking Executive Simplifies Real Time Micro-processor System Design," Computer Design, Jan. 1980, pp. 109-117.
41. Murrill, P. W., Automatic Control of Processes, Intext Educational Publishers, Scranton, PA, 1967, p. 4.

42. Nyquist, H., "Regenerative Theory," Bell System Tech. J., Vol. 11, 1932, p. 1.
43. Shinskey, F. G., "Feedforward Control Applied," ISA Journal, 1963, p. 61.
44. LSI 11, PDP 11/03 Users Manual, Digital Equipment Corporation, Maynard, Massachusetts, First Edition, 1975.
45. Instruction Manual for Model 600 LSI 11 Data Acquisition and Control System, ADAC Corporation, 15 Cummings Park, Woburn, Massachusetts, 1978.
46. MRV11 Data Sheet, MDB Corporation, Orange, California, 1978.
47. Moore, C. F., Meals, M., Gaddis, R., and Green, W., "Micro-processor Control and Optimization of a 1000 Ton Steam Driven Chiller," Proceedings AIChE National Meeting, Miami, October 1978.
48. Meals, M., Total Development System for Direct Digital Control Using Microprocessors, Applied Control Labs Report, October 1980.
49. Meals, M., and Moore, C. F., "Microprocessor Based Distributed Control - Software Organization," Proceedings Fourth International Symposium on Computers, Electronics, and Control, Toronto, November 1978.
50. Moore, C. F., "The Basic Structure of Applied Process Control," Invited Lecture for Fourth International Symposium on Computers, Electronics, and Control, Toronto, November 1978.
51. Finger, A., "A New Architecture for Computer Based Measurement and Control Systems," Transactions ISA Annual Conference 1978, pp. 161-166.
52. Beoghter, K., and Ramler, R., "Advanced Minicomputer Process Control Software Package Put Into a Stand Alone Microprocessor Based Multi-Loop Controller," Transactions ISA Annual Conference 1978, pp. 91-98.
53. Maynard, J., Modular Programming, First Edition, Petrocelli Books, New York, 1972.
54. Horowitz, E., Practical Strategies for Developing Large Software Systems, Addison-Wesley Publishing Company, Reading, MA, 1975.

55. Madnic, S. E., and Donovan, J. J., Operating Systems, McGraw-Hill, New York, 1974.
56. MCS 80 Data Sheet, Intel Corporation, 1978.
57. Auslander, D. M., Takahashi, Y., and Tomizuka, M., "Direct Digital Process Control: Practice and Algorithms for Microprocessor Applications," Proceedings of the IEEE, Vol. 66, No. 2, February 1978.
58. Meals, M., Aspects of Microprocessor Direct Digital Control, Master's Thesis, University of Tennessee, Knoxville, March 1977.
59. Safabakhsh, R., On Digital PI and PID Controller Design, Master's Thesis, The University of Tennessee, Knoxville, March 1979.
60. Troutman, P., and Tasch, F., "A Unique Approach Distributed Digital Control at the Valve," Transactions ISA Annual Conference 1978, pp. 331-338.
61. Gupta, S. C., and Hasdorff, L., Fundamentals of Automatic Control, John Wiley, New York, 1970.
62. Kuo, B. C., Automatic Control Systems, Third Edition, Prentice-Hall, New Jersey, 1975.
63. Cadzow, J. A., Discrete Time Systems, Prentice-Hall, New Jersey, 1973.
64. Didomenico, P. N., "Practical Application of Feedwater Controls to a Utility Drum Boiler," Joint Automatic Control Conference, San Francisco, August 1980.
65. Moore, B. C., "A State Space Theory of Direct Digital Control," Proceedings of JACC, San Francisco, August 1980.
66. Silverman, L. M., and Bettayeb, M., "Optimal Approximations of Linear Systems," Proceedings of JACC, San Francisco, August 1980.
67. Laub, A. J., "On Computing 'Balancing' Transformations," Proceedings of JACC, San Francisco, August 1980.
68. Belew, P. W., Applications of an 8080 Microprocessor in an Infrared Spectroreflectometer Data Acquisition and Processing System, Master's Thesis, The University of Tennessee, Knoxville, March 1979.

69. Blach, R. A., A Computerized Data Acquisition System for a Power Laboratory, Master's Thesis, The University of Tennessee, Knoxville, December 1979.
70. Patel, J. G., A Microcomputer Design and Application, Master's Thesis, The University of Tennessee, Knoxville, June 1976.
71. Phillips, L. A., Development and Performance of a Microprocessor for Closed Loop Digital Control, Master's Thesis, The University of Tennessee, Knoxville, March 1976.
72. Eckhouse, R. H., Minicomputer Systems, Organization, Programming, and Applications, Second Edition, Prentice-Hall, New Jersey, 1979.
73. MacEwen, G. H., Introduction to Computing Systems, McGraw-Hill, New York, 1980.

APPENDICES

APPENDIX A

CHILLER SOFTWARE

This appendix is the complete software package for the steam driven chiller. It is written in MACRO 11 assembler language and makes extensive use of macro calls to a system library. In this software listing, the function of each macro with its arguments are described and exemplified. Appendices C through M are listings of the various macros, but need only be referred to for a detailed description of the actual software expansions. As a final note to aid reading the software, a semicolon (;) denotes a comment, and everything to the right of it is ignored by the assembler.

```
;
.MCALL  MAP$, RTMO$$, DATA$$, SCAN$$, VALV$$, PICO$$, LOGV$$
.MCALL  PARI$$, PARO$$, STKST$, TASK$
;
; THE ABOVE MCALL STATEMENTS DIRECT THE ASSEMBLER
; TO LOOK IN THE SYSTEM LIBRARY FOR MACRO DEFINITIONS
;
;
MAP$  RAM, CON, PAR, SER, VEC
;
; MAP$ is a definition macro WHICH DEFINES BOTH
; SYMBOLIC AND LOCATION PARAMETERS.
; THE ABOVE CALL DEFINES RAM MEMORY
; AND CONTROL TABLES
; AND LABELS PARALLEL I/O CARDS
; AND LABELS SERIAL LINE I/O CARDS
; AND AND INTERRUPT VECTORS
```

```
;
RTMO$$ 4,3,8.
;
; RTMO$$ IS THE REAL TIME MONITOR MACRO. IT IS A STRIPPED
; OPERATING SYSTEM WHOSE ONLY FUNCTION IS TO CALL
; IN TASKS IN ASYNCHRONOUS FASHION.
; ITS ARGUMENTS ARE THE NUMBER OF TASKS
; IN EACH TIME DIVISION. THE ABOVE CALL SPECIFIES:
; 4 VALVES
; 3 TASKS ON A TICK CYCLE
; 8 TASKS ON A SECOND CYCLE
```

```

;
; THE FOLLOWING IS THE MAIN BODY OF THE CONTROL
; STRATEGY. IT IS THE LIST OF TASKS WHICH
; CALL IN ALL CONTROL FUNCTIONS AND SPECIFY
; INFORMATION FLOW BETWEEN CONTROLLERS.
;
;
;
STKST$      ;CALL IN STANDARD MACROS FOR A/D PARALLEL I/O
;
;---CONDENSOR FLOW CONTROL-----
V$20:  TASK$  PICO$,P04$,P00$
;
; TASK$ IS A TASK DEFINITION MACRO. ITS ARGUMENTS
; SPECIFY THE CONTROLLER TO BE USED, IN THIS CASE, A PI
; AS WELL AS THE CONTROL TABLE TO BE WORKED ON, P04$
; AND THE CONTROL TABLE TO CASCADE INTO, P00$.
; IN THIS EXAMPLE, P04$ CONTAINS THE SETPOINT, PROCESS
; VALUE (LOADED BY THE SCAN ROUTINE), AND TUNING
; VALUES FOR A PI CONTROLLER. THE CONTROLLER
; OUTPUT IS CASCADED INTO CONTROL TABLE P00$, THE
; CONDENSOR VALVE CONTROL TABLE.
;
;---EVAPORATOR FLOW CONTROL-----
V$21:  TASK$  PICO$,P05$,P01$
;
;---SPEED CONT.-----
V$22:  TASK$  PICO$,P06$,P02$
;
;---CHILLED WATER TEMP. CONT.-----
V$23:  TASK$  PICO$,P07$,P06$
;
;---SURG1  CONT-----
V$24:  ;NOT IMPLEMENTED
      RTS    PC
;
;---REFRIGERANT FLOW CONTROL-----
V$25:  ;NOT IMPLEMENTED
      RTS    PC
;
;---REFIG TEMP -----
V$26:  TASK$  PICO$,P12$,P04$
;
;---LOAD CONTROL-----
V$27:  ;NOT IMPLEMENTED
      RTS    PC
;

```

```
;
DATA$$ ITGR$$,COLD$$
;
; DATA$$ IS THE SERIAL LINE COMMUNICATIONS MACRO.
; IT SERVICES THE TELETYPE AND ALLOWS INTERNAL VALUES
; TO BE OBSERVED AND MODIFIED WITH THE USE OF SIMPLE
; COMMANDS. THERE ARE TWO TYPES OF INTERNAL NUMBERS,
; FLOATING POINT, AND INTERGER.
;
; ITGR$$ IS A TABLE LISTING THOSE WHICH ARE INTERGERS
;
; DATA$$ HAS THE SECOND FUNCTION OF COLD START
; RAM INITIALIZATION.
;
; COLD$$ IS A TABLE CONTAINING INITIALIZATION TUNING
; PARAMETERS FOR THE CONTROL TABLES.
```

```

;
; INTERGER VARIABLE TABLE
;
ITGR$$: TH$ + P00$ ;CONDENSOR VALVE
U$$ + P00$
ST$ + P00$
;
TH$ + P01$ ;EVAPORATOR VALVE
U$$ + P01$
ST$ + P01$
;
TH$ + P02$ ;GOVERNOR VALVE
U$$ + P02$
ST$ + P02$
;
TH$ + P03$ ;BY-PASS VALVE
U$$ + P03$
ST$ + P03$
;
ST$ + P04$ ;CONDENSOR FLOW
;
ST$ + P05$ ;EVAPORATOR FLOW
;
ST$ + P06$ ;SPEED CONTROL
;
ST$ + P07$ ;CHILLED WATER TEMP
;
ST$ + P10$ ;SURGE
;
ST$ + P12$ ;REFIG TEMP
;
ST$ + P13$ ;CHILLER LOAD

```

```

;
; COLD START TABLE
;
COLD$$: COLD1
  POO$ ;CONDENSOR VALVE
.FLT2 50. ; S$$ = 0 ;SET POINT
.FLT2 50. ; X$$ = 4 ;PROCESS
.FLT2 60. ; HL$ = 10 ;HIGH LEVEL
.FLT2 5. ; LL$ = 14 ;LOW LEVEL
.FLT2 1. ; P1$ = 20 ;GAIN 1
.WORD 3,0 ; TH$ = 24 ;THRESHOLD
.WORD 0,0 ; U$$ = 30 ; WIND UP _CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ;SUM OF ERROR -
.WORD 6 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD1: COLD2
  P01$ ; EVAPORATOR VALVE
.FLT2 50. ; S$$ = 0 ;SET POINT
.FLT2 50. ; X$$ = 4 ;PROCESS
.FLT2 95. ; HL$ = 10 ;HIGH LEVEL
.FLT2 5. ; LL$ = 14 ;LOW LEVEL
.FLT2 1. ; P1$ = 20 ;GAIN 1
.WORD 3,0 ; TH$ = 24 ;THRESHOLD
.WORD 0,0 ; U$$ = 30 ; WIND UP _CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ;SUM OF ERROR -
.WORD 6 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD2: COLD3
  P02$ ;GOVERNOR VALVE
.FLT2 50. ; S$$ = 0 ;SET POINT
.FLT2 50. ; X$$ = 4 ;PROCESS
.FLT2 60. ; HL$ = 10 ;HIGH LEVEL
.FLT2 40. ; LL$ = 14 ;LOW LEVEL
.FLT2 3.0 ; P1$ = 20 ;GAIN 1
.WORD 3,0 ; TH$ = 24 ;THRESHOLD
.WORD 0,0 ; U$$ = 30 ; WIND UP _CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ;SUM OF ERROR -
.WORD 6 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD3: COLD4
  P03$ ;BY-PASS VALVE
.FLT2 10. ; S$$ = 0 ;SET POINT
.FLT2 10. ; X$$ = 4 ;PROCESS
.FLT2 50. ; HL$ = 10 ;HIGH LEVEL
.FLT2 0. ; LL$ = 14 ;LOW LEVEL
.FLT2 1. ; P1$ = 20 ;GAIN 1
.WORD 3,0 ; TH$ = 24 ;THRESHOLD
.WORD 0,0 ; U$$ = 30 ; WIND UP _CONTROLLER OUTPUT

```

```

.FLT2 0 ; SE$ = 34 ;SUM OF ERROR
.WORD 6 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD4: COLD5
  P04$ ; CONDENSOR FLOW
.FLT2 3380. ; S$$ = 0 ;SET POINT
.FLT2 3380. ; X$$ = 4 ;PROCESS
.FLT2 4000. ; HL$ = 10 ;HIGH LEVEL
.FLT2 1000. ; LL$ = 14 ;LOW LEVEL
.FLT2 0. ; P1$ = 20 ;GAIN 1
.FLT2 .004 ; P2$ = 24 ;GAIN 2
.FLT2 50. ; U$$ = 30 ; WIND UP CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ;SUM OF ERROR
.WORD 1 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
.FLT2 0 ; UNUSED
;
COLD5: COLD6
  P05$ ; EVAPORATOR FLOW
.FLT2 2210. ; S$$ = 0 ;SET POINT
.FLT2 2210. ; X$$ = 4 ;PROCESS
.FLT2 3000. ; HL$ = 10 ;HIGH LEVEL
.FLT2 750. ; LL$ = 14 ;LOW LEVEL
.FLT2 0. ; P1$ = 20 ;GAIN 1
.FLT2 .007 ; P2$ = 24 ;GAIN 2
.FLT2 50. ; U$$ = 30 ; WIND UP CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ;SUM OF ERROR
.WORD 1 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
.FLT2 0 ; UNUSED
;
COLD6: COLD7
  P06$ ; SPEED CONTROL TABLE
.FLT2 3150. ; S$$ = 0 ;SET POINT
.FLT2 3150. ; X$$ = 4 ;PROCESS
.FLT2 3700. ; HL$ = 10 ;HIGH LEVEL
.FLT2 2600. ; LL$ = 14 ;LOW LEVEL
.FLT2 0. ; P1$ = 20 ;GAIN 1
.FLT2 .002 ; P2$ = 24 ;GAIN 2
.FLT2 50. ; U$$ = 30 ; WIND UP CONTROLLER OUTPUT
.FLT2 0. ; SE$ = 34 ;SUM OF ERROR
.WORD 1 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
.FLT2 0 ; UNUSED
;
COLD7: COLD10
  P07$ ; CHILLED WATER TEMP
.FLT2 44. ; S$$ = 0 ;SET POINT
.FLT2 44. ; X$$ = 4 ;PROCESS
.FLT2 55. ; HL$ = 10 ;HIGH LEVEL

```

```

.FLT2 32. ; LL$ = 14 ;LOW LEVEL
.FLT2 1. ; P1$ = 20 ;GAIN 1
.FLT2 2. ; P2$ = 24 ;GAIN 2
.FLT2 50. ; U$$ = 30 ; WIND UP CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ;SUM OF ERROR
.WORD 6 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD NGA+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD10: COLD11
P10$ ; SURGE
.FLT2 33. ; S$$ = 0 ;SET POINT
.FLT2 33. ; X$$ = 4 ;PROCESS
.FLT2 50. ; HL$ = 10 ;HIGH LEVEL
.FLT2 30. ; LL$ = 14 ;LOW LEVEL
.FLT2 100. ; P1$ = 20 ;GAIN 1
.FLT2 0. ; P2$ = 24 ;GAIN 2
.FLT2 0. ; U$$ = 30 ; WIND UP CONTROLLER OUTPUT
.FLT2 0. ; SE$ = 34 ;SUM OF ERROR
.WORD 1 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD NGA+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD11: COLD12
P11$ ;NOT IMPLEMENTED
COLD12: COLD13
P12$ ; REFRIGERANT TEMPERATURE
.FLT2 35. ; S$$ = 0 ;SET POINT
.FLT2 35. ; X$$ = 4 ;PROCESS
.FLT2 45. ; HL$ = 10 ;HIGH LEVEL
.FLT2 30. ; LL$ = 14 ;LOW LEVEL
.FLT2 1000. ; P1$ = 20 ;GAIN 1
.FLT2 0. ; P2$ = 24 ;GAIN 2
.FLT2 2500. ; U$$ = 30 ; WIND UP CONTROLLER OUTPUT
.FLT2 0. ; SE$ = 34 ;SUM OF ERROR
.WORD 0 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD NGA+ALALL ; A$$ = 42 ; NGA/NEG ACTION
COLD13: COLD14
P13$ ;CHILLER LOAD
.FLT2 0. ; S$$ = 0 ;SET POINT
.FLT2 0. ; X$$ = 4 ;PROCESS
.FLT2 0. ; HL$ = 10 ;HIGH LEVEL
.FLT2 0. ; LL$ = 14 ;LOW LEVEL
.FLT2 0. ; P1$ = 20 ;GAIN 1
.FLT2 0. ; P2$ = 24 ;GAIN 2
.FLT2 0. ; U$$ = 30 ; WIND UP CONTROLLER OUTPUT
.FLT2 0. ; SE$ = 34 ;SUM OF ERROR
.WORD 0 ; ST$ = 40 ;AUTO/MAN SAMPLE TIME
.WORD POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD14: COLD14
COLD14

```

```
;
SCAN$$ 16.,SLOPE$,INTER$,FILCO$,ENTRY$
;
; SCAN$$ IS THE A/D HANDLER MACRO.
; IT HAS THREE BASIC FUNCTIONS:
; 1)OPERATE THE A/D CONVERTER HARDWARE
; 2)CONVERT THE BINARY RESULT TO ENGINEERING UNITS
; 3)STORE THE RESULT IN THE APPROPRIATE CONTROL TABLE
;
; ITS ARGUMENTS THEREFOR ARE:
; THE NUMBER OF ACTIVE CHAnels, IN THIS CASE 16
; SLOPE$ IS A TABLE WITH A/D CONVERSION CONST.
; INTER$ IS A TABLE WITH A/D CONVERSION CONST.
; FILCO$ IS A TABLE WITH A/D FILTERING CONST.
; ENTRY$ IS A TABLE LISTING LOCATIONS FOR VALUE STORAGE
```

```

;
; SLOPE FOR LINEAR EXTRAPOLATION
;

```

```

SLOPE$:

```

```

.FLT2 .02442 ;CONDENSOR VALVE
.FLT2 .02442 ;EVAPORATOR VALVE
.FLT2 .026 ;GOVERNOR VALVE
.FLT2 .02442 ;BY-PASS VALVE
.FLT2 1.25 ;TURBIN RPM
.FLT2 .00611 ;REFIG FLOW
.FLT2 .009158 ;DEL PRESS, COMPRESSOR
.FLT2 .00945 ;EVAP PRESS
.FLT2 .01954 ;INLET EVAP WATER TEMP
.FLT2 .4565 ;STEAM FLOW
.FLT2 75. ;EVAP FLOW
.FLT2 120. ;COND FLOW
.FLT2 .00611 ;REFIG TEMP
.FLT2 .01954 ;INLET COND. WATER TEMP
.FLT2 .01954 ;OUTLET COND WATER TEMP
.FLT2 .0275 ;OUTLET EVAP WATER TEMP

```

```

;
; INTERCEPT FOR LINEAR EXTRAPOLATION
;

```

```

INTER$:

```

```

.FLT2 .0 ;CONDENSOR VALVE
.FLT2 .0 ;EVAPORATOR VALVE
.FLT2 .0 ;GOVERNOR VALVE
.FLT2 .0 ;BY-PASS VALVE
.FLT2 0. ;TURBIN RPM
.FLT2 -5. ;REFIG FLOW
.FLT2 -7.5 ;DEL PRESS, COMPRESSOR
.FLT2 -19.064 ;EVAP PRESS
.FLT2 32 ;INLET EVAP WATER TEMP
.FLT2 -13.06 ;STEAM FLOW
.FLT2 -1305 ;EVAP FLOW
.FLT2 -2560 ;COND FLOW
.FLT2 25. ;REFIG TEMP
.FLT2 32. ;INLET COND. WATER TEMP
.FLT2 32. ;OUTLET COND WATER TEMP
.FLT2 32 ;OUTLET EVAP WATER TEMP

```

```

;
;   FILTER CONSTANTS _SQUART ROOT
;
FILCO$:
  0,0 ;CONDENSOR VALVE
  0,0 ;EVAPORATOR VALVE
  0,0 ;GOVERNOR VALVE
  0,0 ;BY-PASS VALVE
  0,0 ;TURBIN RPM
  0,0 ;REFIG FLOW
  0,0 ;DEL PRESS, COMPRESSOR
  0,0 ;EVAP PRESS
  0,0 ;INLET EVAP WATER TEMP
  0,1 ;STEAM FLOW
  0,1 ;EVAP FLOW
  0,1 ;COND FLOW
  0,0 ;REFIG TEMP
  0,0 ;INLET COND. WATER TEMP
  0,0 ;OUTLET COND WATER TEMP
  0,0 ;OUTLET EVAP WATER TEMP
;
;
;   TABLE ENTRY
;
ENTRY$:
  P00$+X$$ ;CONDENSOR VALVE
  P01$+X$$ ;EVAPORATOR VALVE
  P02$+X$$ ;GOVERNOR VALVE
  P03$+X$$ ;BY-PASS VALVE
  P06$+X$$ ;TURBIN RPM
  PB1$+X$$ ;REFIG FLOW
  PB1$+4 ;DEL PRESS, COMPRESSOR
  PB1$+10 ;EVAP PRESS
  P11$+X$$ ;INLET EVAP WATER TEMP
  PB1$+14 ;STEAM FLOW
  P05$+X$$ ;EVAP FLOW
  P04$+X$$ ;COND FLOW
  P12$+X$$ ;REFIG TEMP
  PB1$+20 ;INLET COND. WATER TEMP
  PB1$+24 ;OUTLET COND WATER TEMP
  P07$+X$$ ;OUTLET EVAP WATER TEMP

```

```
;
VALV$$
;
; VALV$$ IS A STANDARD VALVE CONTROL ROUTINE
```

```
;
PICO$$
;
;STANDARD PI CONTROLLER
```

```
;
PARI$$  ACTIV$,ACTIP$,BMPTB$,STON$,STOF$
;
;PARI$$ IS THE PARALLEL INPUT HANDLER
; ITS FUNCTIONS ARE TO :
; 1)MONITOR THE VALVE STATION PUSH BUTTONS
;   FOR OPERATOR REQUESTS TO TURN ON OR OFF
;   CONTROL LOOPS
; 2)DETERMINE IF THAT REQUEST CAN BE ALLOWED
; 3)TURN THE CONTROL LOOP ON WITH A BUMPLESS TRANSFER
;
; ITS ARGUMENTS THEREFOR ARE:
; ACTIV$ AND ACTIP$ ARE ACTIVE VALVES AND
;   CONTROL TABLES
; BMPTB$ IS A TABLE WITH BUMPLESS TRANSFER REQUIRMENTS
; STON$ IS A TABLE WITH STATUS ON REQUIRMENTS
; STOF$ IS A TABLE WITH STATUS OFF REQUIRMENTS
```

```

STON$:  XB,0      ;XB
        0,0      ;XG
        0,0      ;XE
        0,0      ;XC
        XE+XG,RPM+FE ;CW
        XG,0      ;RPM
        XE,0      ;FE
        XC,0      ;FC
;  XB,RPM      ;SR
;  XC,FC      ;RT
;  FE+RPM+CW,XE+XG ;LD
;
STOF$:  XB,SR      ;XB
        XG,LD+SR+RPM+CW ;XG
        XE,LD+FE      ;XE
        XC,RT+FC      ;XC
        0,LD+CW      ;CW
        0,CW+LD+RPM ;RPM
        0,LD+FE      ;FE
        0,RT+FC      ;FC
;  0,SR      ;SR
;  0,RT      ;RT
;  0,LD      ;LD
ACTIV$ = XB+XG+XE+XC
ACTIP$ = RPM,+FE+FC+CW+SR+LD

```

;
BMPTB\$:

ST\$+P03\$,XBON
ST\$+P02\$,XGON
ST\$+P01\$,XEON
ST\$+P00\$,XCON
ST\$+P07\$,CWON
ST\$+P06\$,RPMON
ST\$+P05\$,FEON
ST\$+P04\$,FCON
; ST\$+P10\$,SRON
ST\$+P12\$,RTON
; ST\$+P13\$,LDON
BMPED
BMPED

;
XBON: X\$\$+P03\$,S\$\$+P03\$
XGON: X\$\$+P02\$,S\$\$+P02\$
XEON: X\$\$+P01\$,S\$\$+P01\$
XCON: X\$\$+P00\$,S\$\$+P00\$
CWON: X\$\$+P06\$,SE\$+P07\$
RPMON: X\$\$+P02\$,SE\$+P06\$
X\$\$+P06\$,S\$\$+P06\$
FEON: X\$\$+P01\$,SE\$+P05\$
X\$\$+P05\$,S\$\$+P05\$
FCON: X\$\$+P00\$,SE\$+P04\$
X\$\$+P04\$,S\$\$+P04\$
;SRON: X\$\$+P03\$,SE\$+P11\$
RTON: X\$\$+P04\$,SE\$+P12\$
;LDON: X\$\$+P05\$,SE\$+P13\$
; X\$\$+P13\$,S\$\$+P13\$
BMPED: ;END OF FILE

```
;
;PARO$$
;
;PARO$$ IS THE PARALLEL OUTPUT HANDLER MACRO
;
;  ITS FUNCTION IS TO DRIVE THE
;  PARALLEL OUTPUT HARDWARE TO MOVE VALVES AND
;  DISPLAY STATUS VIA LEDS.
```

APPENDIX B

BOILER SOFTWARE

This appendix is the complete software package for the boiler. It is written in MACRO 11 assembler language and makes extensive use of macro calls to a system library. In this software listing, the function of each macro with its arguments are described and exemplified. Appendices C through M are listings of the various macros, but need only be referred to for a detailed description of the actual software expansions. As a final note to aid reading the software, a semicolon (;) denotes a command, and everything to the right of it is ignored by the assembler.

```

;
.MCALL  MAP$,RTM$$,DATA$$,SCAN$$,VALV$$,PIGCO$,PARI$$,PARO$$
.MCALL  LELA$$,STASK$,TASK$
;
; THE ABOVE MCALL STATEMENTS DIRECT THE ASSEMBLER
; TO LOOK IN THE SYSTEM LIBRARY FOR MACRO DEFINITIONS
;
.NLIST  BEX
; THIS WILL SHORTEN BINARY LISTINGS
.ASECT
; ABSOLUTE ADDRESSING
;
; MAP$      RAM,CON,PAR,SER,CVEC
;
; MAP$ is a definition macro WHICH DEFINES BOTH
; SYMBOLIC AND LOCATION PARAMETERS.
; THE ABOVE CALL DEFINES RAM MEMORY
; AND CONTROL TABLES
; AND LABELS PARALLEL I/O CARDS
; AND LABELS SERIAL LINE I/O CARDS
; AND AND INTERRUPT VECTORS

```

```
      ;
      RTM$$ 6,4,14.
;
;
; RTM$$ IS THE REAL TIME MONITOR MACRO. IT IS A STRIPPED
; OPERATING SYSTEM WHOSE ONLY FUNCTION IS TO CALL
; IN TASKS IN ASYNCHRONOUS FASHION.
; ITS ARGUMENTS ARE THE NUMBER OF TASKS
; IN EACH TIME DIVISION. THE ABOVE CALL SPECIFIES:
; 6 VALVES
; 4 TASKS ON A TICK CYCLE
; 14 TASKS ON A SECOND CYCLE
```

```

;
;
; THE FOLLOWING IS THE MAIN BODY OF THE CONTROL
; STRATEGY. IT IS THE LIST OF TASKS WHICH
; CALL IN ALL CONTROL FUNCTIONS AND SPECIFY
; INFORMATION FLOW BETWEEN CONTROLLERS.
;
;
;
; TASK$ IS A TASK DEFINITION MACRO. ITS ARGUMENTS
; SPECIFY THE CONTROLLER TO BE USED,
; AS WELL AS THE CONTROL TABLE TO BE WORKED ON,
; AND THE CONTROL TABLE TO CASCADE INTO,
;
;
; --- TASK VECTORS ---
;
STASK$
; STASK$ CALLS IN THE STANDARD I/O HANDLERS (A/D AND DI/DO)
V$13: JSR PC,V$130
      MOV 6,(R1)
      RTS PC
;
;
; AIR FLOW
V$20: JSR PC,AFLOW0 ;CALCULATE AIR FLOW
      TASK$ AIR,P7$,P1$
;
; OPACITY
V$21: TASK$ PIN,P8$
;
; DRAFT FEED FORWARD
V$22: MOV X$$+P7$,X$$+P9$
      MOV X$$+2+P7$,X$$+2+P9$
      TASK$ LL,P9$,P16$
;
; FUEL FLOW
V$23: TASK$ FUEL,P10$,P2$
;
; O2 RATIO CONTROL
V$24: TASK$ PIN,P11$
;
; AIR LEAD
V$25: TASK$ LAGCO$,P12$,P10$
;
; STEAM PRESSURE CONTROL (STD.)
V$26: TASK$ PI,P13$,P12$
;
; STEAM PRESSURE CONTROL (GAP)
V$27: MOV X$$+P13$,X$$+P14$
      MOV X$$+2+P13$,X$$+2+P14$
      TASK$ PIG,P14$,P12$
;
; STEAM PRESSURE FEED FORWARD
V$28: TASK$ LL,P15$,P13$
;
; FURNACE DRAFT
V$29: TASK$ PI,P16$,P3$
;
; DRUM LEVEL CONTROL
V$210: TASK$ PI,P17$,P4$
;
; DRUM LEVEL FEED FORWARD

```

```
V$211:  MOV      X$$+P15$,X$$+P18$
        MOV      X$$+2+P15$,X$$+2+P18$
        TASK$    LL,P18$,P17$
;        AIR PRE HEAT CONTROL
V$212:  TASK$    PI,P19$,P5$
;        BY PASS TEMP CONTROL
V$213:  TASK$    PI,P20$,P6$
```

```

; THIS ROUTINE SWITCHES UPON REQUEST FROM
; STANDARD TO GAP CONTROL FOR THE STEAM
; PRESSURE CONTROLLER
V$130:  CMP      STA2$,STA6$      ;CHECK FOR CHANGES IN STATUS
        BEQ      V$134A          ;EXIT IF NONE
        ;
        MOV      STA2$,-(SP)      ;LOOK FOR NEW BIT
        BIC      STA6$,(SP)      ;BY CLEARING OLD
        BEQ      V$134          ;EXIT IF NO NEW BITS
        ;
        BIT      SPG,(SP);WAS IT PRESSURE GAP?
        BEQ      V$131          ;BRANCH IF NO
        BIC      SPS,STA2$;KILL STD PRESS CONTROL
        ;
V$131:  BIT      SPS,(SP)        ;WAS IT STD PRESS CONTROL
        BEQ      V$133          ;BRANCH IF NO
        BIC      SPG,STA2$      ;KILL PRESS GAP IF YES
        ;

```

```

;   THIS ROUTINE TURNS ON THS O2 AND OPACITY
;   CONTROLLERS IN RESPONSE TO TURNING ON HTE
;   AIR/FUEL BOOST.
V$133:  BIT      AL,(SP)          ;WAS IT AIR LEAD
        BEQ      V$134          ;BRANCH IF NO
        JSR      PC,V$139
        ;
V$134:  MOV      STA2$,STA6$      ;STORE FOR NEXT PASS
        TST      (SP)+          ;POP CELL
        ;
V$134A: MOV      100000,RO
        BIT      AL,STA2$       ; IS AIR LEAD ON
        BEQ      V$135         ;BRANCH IF NO
        BIS      RO,ST$+P11$    ; TURN O2 ON TOO
        BIS      RO,ST$+P8$    ;  TURN ON OPACITY TOO
        BR       V$135A
        ;
V$135:  BIC      RO,ST$+P11$     ;TURN O2 OFF
        BIC      RO,ST$+P8$     ;TURN OPACITY OFF TOO
        ;

```

```
; THIS ROUTINE CALCULATES AO2
; SETPOINT BASED ON THE STEAM FLOW
; AND THE TWO GAINS IN TABLE 15
```

```
V$135A:
```

```
V$136:  MOV      X$$+2+P15$,-(SP);CALC O2 FROM STEAM FLOW
        MOV      X$$+P15$,-(SP)
        MOV      G1$+2+P21$,-(SP)
        MOV      G1$+P21$,-(SP)
        FMUL     SP                ; SF * G1
        ;
        MOV      G2$+2+P21$,-(SP)
        MOV      G2$+P21$,-(SP)
        FADD     SP                ; O2 = (SF * G1) + G2
        MOV      (SP)+,S$$+P11$
        MOV      (SP)+,S$$+2+P11$
        ;
        RTS      PC
        ;
```

```

;   CALCULAED BUMPLESS TRANSFER FOR O2 CONTROLLER
V$139:  MOV     S$$+2+P7$,-(SP) ; CALCULATE AIR FUEL RATIO
        MOV     S$$+P7$,-(SP)
        MOV     S$$+2+P10$,-(SP)           ;USE SET POINT
        MOV     S$$+P10$,-(SP)
        FDIV    SP;RATIO = AIR FLOW / FUEL FLOW
        ;
        MOV     (SP),SE$+P11$ ;STORE IN O2 CONTROL
        MOV     2(SP),SE$+2+P11$
        MOV     (SP)+,U$$+P11$ ;FOR THE FIRST PASS
        MOV     (SP)+,U$$+2+P11$
        RTS     PC
;

```

```

;   THIS ROUTINE IS A FUEL TIE BACK
;   WHICH KEEPS THE FUEL FLOW LESS
;   THAN THE AIR FLOW UNDER ALL CONDITIONS.
FUEL:  MOV     200,-(SP)          ;STOP INTERRUPTS
        MTPS   (SP)+
        MOV     X$$$+2+P7$,-(SP)      ;GET AIR FLOW
        MOV     X$$$+P7$,-(SP)
        MOV     S$$$+2+P10$,-(SP)     ;GET FUEL
        MOV     S$$$+P10$,-(SP)
        FSUB    SP; FUEL TIE = AIR FLOW - FUEL FLOW
        ;
        BPL     FUEL1              ; OK,OK,LIMIT
        MOV     X$$$+P7$,S$$$+P10$
        MOV     X$$$+2+P7$,S$$$+2+P10$
        ;
FUEL1:  CMP     (SP)+,(SP)+          ;FLUSH STACK
        CLR     -(SP)
        MTPS   (SP)+              ;ALLOW INTERRUPTS
        JMP     PI
        ;
;

```

```

;   THIS ROUTINE IS THE AIR TIE BACK.
;   IT KEEPS THE AIR FLOW ABOVE THE FUEL
;   FLOW FOR ALL CONDITIONS.
AIR:  MOV      200,-(SP)          ;STOP INTERRUPTS
      MTPS     (SP)+
      MOV      $$$+2+P7$,-(SP) ;GET AIR
      MOV      $$$+P7$,-(SP)
      MOV      $$$+2+P10$,-(SP)          ;GET FUEL
FLOW  MOV      $$$+P10$,-(SP)
      FSUB     SP          ; AIR TIE = AIR FLOW - FUEL
FLOW
      ;
      BPL      AIR1          ; OK,OK,LIMIT
      MOV      $$$+P10$, $$$+P7$
      MOV      $$$+2+P10$, $$$+2+P7$
      ;
AIR1:  CMP      (SP)+,(SP)+      ;FLUSH STACK
      CLR      -(SP)
      MTPS     (SP)+          ;ALLOW INTERRUPTS
      JMP      PI
      ;

```

```

; THIS ROUTINE IS A FUNCTION GENERATOR
; FOR THE AIR FLOW. THE TRANSDUCER READING
; IS STORED IN PB1$ ( 310)
; THE FUNCTION GENERATOR IS A ONE DIMENSIONAL TABLE
; WITH FLOATING BREAKPOINTS.
; TABLE 23 CONTAINS THE TRANSDUCER BREAKPOINTS
; TABLE 24 CONTAINS THE GENERATED SIGNAL
; BREAKPOINTS.
; BOTH TABLES MUST HAVE THE SAME NUMBER OF ENTRIES
; AND BOTH MUST START WITH 0 AND END WITH 100
AFLOW0: CLR      R3                      ;LOAD POINTER
        CMP      -(SP),-(SP)
        BR       AFLOWA
;
AFLOW1: CMP      (R3)+,(R3)+          ; POINTER TO NEXT BREAK POINT
AFLOWA: MOV      P22$+2(R3),2(SP)    ; GET BREAK POINT
        MOV      P22$(R3),(SP)
        MOV      PB1$+2,-(SP)       ; GET READING
        MOV      PB1$,-(SP)
        FSUB     SP                  ; COMPARE THEM
        BMI     AFLOW1;LOOP IF BREAK POINT < READING
;
        TST      R3                  ; FIRST ENTRY?
        BNE     AFLOW2              ; BRANCH IF NO
        MOV      P22$,(SP)
        MOV      P22$+2,2(SP)
        BR       AFLOW3
;
AFLOW2: MOV      P22$+2(R3),-(SP)    ; CALC P22$ SPAN
        MOV      P22$(R3),-(SP)
        MOV      P22$-2(R3),-(SP)
        MOV      P22$-4(R3),-(SP)
        FSUB     SP                  ; SPAN
        FDIV     SP                  ; % = DIFF/SPAN
;
        MOV      P23$+2(R3),-(SP)    ; CALC P23$ SPAN
        MOV      P23$(R3),-(SP)
        MOV      P23$-2(R3),-(SP)
        MOV      P23$-4(R3),-(SP)
        FSUB     SP                  ; SPAN
        FMUL     SP                  ; DIFF = SPAN * %
        BIS      100000,(SP)         ; MAKE IT NEGATIVE
        MOV      P23$+2(R3),-(SP)    ; CALC TRUE FLOW
        MOV      P23$(R3),-(SP)
        FADD     SP; FLOW = BREAK POINT - DIFF
;
AFLOW3: MOV      (SP)+,X$$+P7$        ; STORE RESULT
        MOV      (SP)+,X$$+2+P7$
        RTS      PC
;
;

```

DATA\$\$ ITGR\$\$,COLD\$\$

;
;
; DATA\$\$ IS THE SERIAL LINE COMMUNICATIONS MACRO.
; IT SERVICES THE TELETYPE AND ALLOWS INTERNAL VALUES
; TO BE OBSERVED AND MODIFIED WITH THE USE OF SIMPLE
; COMMANDS. THERE ARE TWO TYPES OF INTERNAL NUMBERS,
; FLOATING POINT, AND INTERGER.
;
; ITGR\$\$ IS A TABLE LISTING THOSE WHICH ARE INTERGERS
;
; DATA\$\$ HAS THE SECOND FUNCTION OF COLD START
; RAM INITIALIZATION.
;
; COLD\$\$ IS A TABLE CONTAINING INITIALIZATION TUNING
; PARAMETERS FOR THE CONTROL TABLES.

```
SCAN$$ 16.,SLOPE$,INTER$,FILCO$,ENTRY$
;
;
; SCAN$$ IS THE A/D HANDLER MACRO.
; IT HAS THREE BASIC FUNCTIONS:
; 1)OPERATE THE A/D CONVERTER HARDWARE
; 2)CONVERT THE BINARY RESULT TO ENGINEERING UNITS
; 3)STORE THE RESULT IN THE APPROPRIATE CONTROL TABLE
;
; ITS ARGUMENTS THEREFOR ARE:
; THE NUMBER OF ACTIVE CHANnELS, IN THIS CASE 16
; SLOPE$ IS A TABLE WITH A/D CONVERSION CONST.
; INTER$ IS A TABLE WITH A/D CONVERSION CONST.
; FILCO$ IS A TABLE WITH A/D FILTERING CONST.
; ENTRY$ IS A TABLE LISTING LOCATIONS FOR VALUE STORAGE
```

```
      VALV$$  
      PIGCO$  
; VALV$$ AND PIGCO$$ ARE THE TWO BASIC CONTROL  
; ALGORITHMS,  
;  
      LELA$$  
; LELA$ IS THE BASIC LEAD/LAG ALGORITHM
```

PARI\$\$ ACTIV\$,ACTIP\$,BMPTB\$,STON\$,STOF\$

```
;
;
;PARI$$ IS THE PARALLEL INPUT HANDLER
; ITS FUNCTIONS ARE TO :
; 1)MONITOR THE VALVE STATION PUSH BUTTONS
;   FOR OPERATOR REQUESTS TO TURN ON OR OFF
;   CONTROL LOOPS
; 2)DETERMINE IF THAT REQUEST CAN BE ALLOWED
; 3)TURN THE CONTROL LOOP ON WITH A BUMPLESS TRANSFER
;
; ITS ARGUMENTS THEREFOR ARE:
; ACTIV$ AND ACTIP$ ARE ACTIVE VALVES AND
;   CONTROL TABLES
; BMPTB$ IS A TABLE WITH BUMPLESS TRANSFER REQUIRMENTS
; STON$ IS A TABLE WITH STATUS ON REQUIRMENTS
; STOF$ IS A TABLE WITH STATUS OFF REQUIRMENTS
```

PARO\$\$

```
;
;
; PARO$$ IS THE PARALLEL OUTPUT HANDLER MACRO
;
;   ITS FUNCTION IS TO DRIVE THE
;   PARALLEL OUTPUT HARDWARE TO MOVE VALVES AND
;   DISPLAY STATUS VIA LEDS.
;
```

```

;          INTERGER VARIABLE TABLE
;
ITGR$$:  TH$      + P1$      ;FD DAMPER
          U$$      + P1$
          ST$      + P1$
          ;
          TH$      + P2$      ;OIL VALVE
          U$$      + P2$
          ST$      + P2$
          ;
          TH$      + P3$      ;ID DAMPER
          U$$      + P3$
          ST$      + P3$
          ;
          TH$      + P4$      ;FEED WATER VALVE
          U$$      + P4$
          ST$      + P4$
          ;
          TH$      + P5$      ;STEAM PRE HEAT VALVE
          U$$      + P5$
          ST$      + P5$
          ;
          TH$      + P6$      ;BYPASS DAMPER
          U$$      + P6$
          ST$      + P6$
          ;
          ST$      + P7$      ;AIR FLOW
          ;
          ST$      + P8$      ;OPACITY HIGH LIMIT
          ;
          ST$      + P9$      ;OPACITY LOW LIMIT
          ;
          ST$      + P10$     ;OIL FLOW
          ;
          ST$      + P11$     ;O2 RATIO CONTROL
          ;
          ST$      + P12$     ;AIR LEAD
          ;
          ST$      + P13$     ;STEAM PRESS. CONTROL STD.
          ;
          ST$      + P14$     ;STEAM PRESS CONTROL GAP
          ;
          ST$      + P15$     ;STEAM PRESS CONTROL, FEED FORWARD
          ;
          ST$      + P16$     ;FURNACE DRAFT
          ;
          ST$      + P17$     ;DRUM LEVEL
          ;
          ST$      + P18$     ;DRUM LEVEL, FEED FORWARD
          ;

```

ST\$ + P19\$;AIR PRE HEAT
;
ST\$ + P20\$;BYPASS TEMP
;

```

;          COLD START TABLE
;
COLD$$: COLD1
        P1$+HL$ ;FD DAMPER
.FLT2   60.      ;          HL$      = 10      ;HIGH LEVEL
.FLT2   5.       ;          LL$      = 14      ;LOW LEVEL
.FLT2   5.       ;          G1$      = 20      ;GAIN 1
        11,0     ;          TH$      = 24      ;THRESHOLD
        0,0      ;          U$$      = 30      ;CONTROLLER OUTPUT
.FLT2   0        ;          SE$      = 34      ;SUM OF ERROR
        6        ;          ST$      = 40      ;AUTO/MAN _SAMPLE
TIME    POS+ALALL ;          A$$      = 42 ; POS/NEG ACTION
;
COLD1:  COLD2
        P2$+HL$ ;OIL VALVE
.FLT2   95.      ;          HL$      = 10      ;HIGH LEVEL
.FLT2   5.       ;          LL$      = 14      ;LOW LEVEL
.FLT2   1.       ;          G1$      = 20      ;GAIN 1
        6,0      ;          TH$      = 24      ;THRESHOLD
        0,0      ;          U$$      = 30      ;CONTROLLER OUTPUT
.FLT2   0        ;          SE$      = 34      ;SUM OF ERROR
        6        ;          ST$      = 40      ;AUTO/MAN _SAMPLE
TIME    POS+ALALL ;          A$$      = 42 ; POS/NEG ACTION
;
COLD2:  COLD3
        P3$+HL$ ;ID DAMPER
.FLT2   95.      ;          HL$      = 10      ;HIGH LEVEL
.FLT2   0.       ;          LL$      = 14      ;LOW LEVEL
.FLT2   3.       ;          G1$      = 20      ;GAIN 1
        6,0      ;          TH$      = 24      ;THRESHOLD
        0,0      ;          U$$      = 30      ;CONTROLLER OUTPUT
.FLT2   0        ;          SE$      = 34      ;SUM OF ERROR
        6        ;          ST$      = 40      ;AUTO/MAN _SAMPLE
TIME    POS+ALALL ;          A$$      = 42 ; POS/NEG ACTION
;
COLD3:  COLD4
        P4$+HL$ ;WATER VALVE
.FLT2   95.      ;          HL$      = 10      ;HIGH LEVEL
.FLT2   0.       ;          LL$      = 14      ;LOW LEVEL
.FLT2   4.       ;          G1$      = 20      ;GAIN 1
        6,0      ;          TH$      = 24      ;THRESHOLD
        0,0      ;          U$$      = 30      ;CONTROLLER OUTPUT
.FLT2   0        ;          SE$      = 34      ;SUM OF ERROR
        6        ;          ST$      = 40      ;AUTO/MAN _SAMPLE
TIME    POS+ALALL ;          A$$      = 42 ; POS/NEG ACTION
;
COLD4:  COLD5

```

```

P5$+HL$ ; STEAM VALVE
.FLT2 95. ; HL$ = 10 ; HIGH LEVEL
.FLT2 8. ; LL$ = 14 ; LOW LEVEL
.FLT2 2. ; G1$ = 20 ; GAIN 1
      3,0 ; TH$ = 24 ; THRESHOLD
      0,0 ; U$$ = 30 ; CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ; SUM OF ERROR
      6 ; ST$ = 40 ; AUTO/MAN _SAMPLE
TIME
      POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
      ;
COLD5: COLD6
P6$+HL$ ; BYPASS DAMPER
.FLT2 80. ; HL$ = 10 ; HIGH LEVEL
.FLT2 0. ; LL$ = 14 ; LOW LEVEL
.FLT2 1. ; G1$ = 20 ; GAIN 1
      3,0 ; TH$ = 24 ; THRESHOLD
      0,0 ; U$$ = 30 ; CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ; SUM OF ERROR
      6 ; ST$ = 40 ; AUTO/MAN _SAMPLE
TIME
      POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
      ;
COLD6: COLD7
P7$+HL$ ; AIR FLOW
.FLT2 100. ; HL$ = 10 ; HIGH LEVEL
.FLT2 10. ; LL$ = 14 ; LOW LEVEL
.FLT2 .1 ; G1$ = 20 ; GAIN 1
.FLT2 .1 ; G2$ = 24 ; GAIN 2
.FLT2 0. ; U$$ = 30 ; CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ; SUM OF ERROR
      1 ; ST$ = 40 ; AUTO/MAN _SAMPLE
TIME
      POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
      ;
COLD7: COLD8
P8$ ; MAX OPACITY
.FLT2 5. ; S$$ = 0 ; SET POINT
.FLT2 0. ; X$$ = 4 ; PROCESS
.FLT2 10. ; HL$ = 10 ; HIGH LEVEL
.FLT2 0. ; LL$ = 14 ; LOW LEVEL
.FLT2 .5 ; G1$ = 20 ; GAIN 1
.FLT2 .01 ; G2$ = 24 ; GAIN 2
.FLT2 0. ; U$$ = 30 ; CONTROLLER OUTPUT
.FLT2 0 ; SE$ = 34 ; SUM OF ERROR
      1 ; ST$ = 40 ; AUTO/MAN _SAMPLE
TIME
      NGA+ALALL ; A$$ = 42 ; POS/NEG ACTION
.FLT2 1.3 ; NHL$ = 44 ; NO OUTPUT HIGH
LEVEL
.FLT2 1. ; NLL$ = 50 ; NO OUTPUT LOW LEVEL

```

```

;
COLD8:  COLD9
        P9$+HL$ ; FURNACE DRAFT FEED FORWARD
.FLT2   50.    ; HL$      = 10    ; HIGH LEVEL
.FLT2   50.    ; LL$      = 14    ; LOW LEVEL
.FLT2    .5    ; G1$
.FLT2    .0    ; P2
.FLT2    0.    ; U$$
.FLT2    0.    ; SE$
        2      ; SAMPLE TIME
        POS+ALALL
;
COLD9:  COLD10
        P10$+HL$ ; OIL FLOW
.FLT2   100.   ; HL$      = 100   ; HIGH LEVEL
.FLT2   10.    ; LL$      = 14    ; LOW LEVEL
.FLT2    1.    ; G1$      = 20    ; GAIN 1
.FLT2    .2    ; G2$      = 24    ; GAIN 2
.FLT2    0.    ; U$$      = 30    ; CONTROLER OUTPUT
.FLT2    0     ; SE$      = 34    ; SUM OF ERROR
        1     ; ST$      = 40    ; AUTO/MAN _SAMPLE
TIME
        POS+ALALL ; A$$      = 42 ; POS/NEG ACTION
;
COLD10: COLD11
        P11$+HL$ ; O2
.FLT2   10.    ; HL$      = 10    ; HIGH LEVEL
.FLT2    0.    ; LL$      = 14    ; LOW LEVEL
.FLT2    .5    ; G1$      = 20    ; GAIN 1
.FLT2    .1    ; G2$      = 24    ; GAIN 2
.FLT2    0.    ; U$$      = 30    ; CONTROLER OUTPUT
.FLT2    0.    ; SE$      = 34    ; SUM OF ERROR
        1     ; ST$      = 40    ; AUTO/MAN _SAMPLE
TIME
        POS+ALALL ; A$$      = 42 ; POS/NEG ACTION
.FLT2   1.    ; NHL$
.FLT2   1.    ; NLL$
;
COLD11: COLD12
        P12$+HL$ ; AIR LEAD
.FLT2   100.   ; HL$      = 10    ; HIGH LEVEL
.FLT2   20.    ; LL$      = 14    ; LOW LEVEL
.FLT2    1.    ; G1$      = 20    ; GAIN 1
.FLT2   10.    ; G2$      = 24    ; GAIN 2
.FLT2   10.    ; U$$      = 30    ; CONTROLER OUTPUT
.FLT2    0.    ; SE$      = 34    ; SUM OF ERROR
        1.    ; ST$      = 40    ; AUTO/MAN _SAMPLE
TIME
        POS+ALALL ; A$$      = 42 ; POS/NEG ACTION
.FLT2   .95    ; FUEL LAG
.FLT2   .95    ; AIR LAG

```

```

;
COLD12: COLD13
P13$ ;STEAM PRESSURE STD
.FLT2 150. ; S$$ = 0 ;SET POINT
.FLT2 0. ; X$$ = 4 ;PROCESS
.FLT2 160. ; HL$ = 10 ;HIGH LEVEL
.FLT2 100. ; LL$ = 14 ;LOW LEVEL
.FLT2 8. ; G1$ = 20 ;GAIN 1
.FLT2 .01 ; G2$ = 24 ;GAIN 2
.FLT2 0. ; U$$ = 30 ; CONTROLLER OUTPUT
.FLT2 0. ; SE$ = 34 ;SUM OF ERROR
TIME 6 ; ST$ = 40 ;AUTO/MAN _SAMPLE
POS+ALALL ; A$$ = 42 ; POS/NEG ACTION
;
COLD13: COLD14
P14$ ;STEAM PRESS GAP
.FLT2 150. ; S$$ = 0 ;SET POINT
.FLT2 0. ; X$$ = 4 ;PROCESS
.FLT2 160. ; HL$ = 10 ;HIGH LEVEL
.FLT2 100. ; LL$ = 14 ;LOW LEVEL
.FLT2 10. ; G1$
.FLT2 .02 ; P2
.FLT2 0. ; U$$
.FLT2 0. ; SE$
6 ; SAMPLE TIME
POS+ALALL
.FLT2 0. ;NHL
.FLT2 0. ;NLL
.FLT2 2. ;GAP
;
COLD14: COLD15
P15$+HL$ ;STEAM PRESS FEED FORWARD
.FLT2 200. ; HL$ = 10 ;HIGH LEVEL
.FLT2 140. ; LL$ = 14 ;LOW LEVEL
.FLT2 1. ; G1$
.FLT2 .0 ; P2
.FLT2 0. ; U$$
.FLT2 0. ; SE$
12 ; SAMPLE TIME
POS+ALALL
;
COLD15: COLD16
P16$ ;FURNACE DRAFT
.FLT2 -.25 ;SET POINT
.FLT2 0. ;PROCESS
.FLT2 0. ; HL$ = 10 ;HIGH LEVEL
.FLT2 -.5 ; LL$ = 14 ;LOW LEVEL
.FLT2 1. ; G1$
.FLT2 .5 ; P2
.FLT2 0. ; U$$

```

```

.FLT2  0.      ;      SE$
        1      ;      SAMPLE TIME
        NGA+ALALL
        ;
COLD16: COLD17
        P17$    ;DRUM LEVEL
.FLT2  0.      ;      S$$      = 0      ;SET POINT
.FLT2  0.      ;      X$$      = 4      ;PROCESS
.FLT2  5.      ;      HL$      = 10     ;HIGH LEVEL
.FLT2 -5.      ;      LL$      = 14     ;LOW LEVEL
.FLT2  8.      ;      G1$
.FLT2  .2      ;      P2
.FLT2  0.      ;      U$$
.FLT2  0.      ;      SE$
        10     ;      SAMPLE TIME
        POS+ALALL
        ;
COLD17: COLD18
        P18$+HL$ ;DRUM LEVEL FEED FORWARD
.FLT2  100.    ;      HL$      = 10     ;HIGH LEVEL
.FLT2  100.    ;      LL$      = 14     ;LOW LEVEL
.FLT2  1.      ;      G1$
.FLT2  .0      ;      P2
.FLT2  0.      ;      U$$
.FLT2  0.      ;      SE$
        10     ;      SAMPLE TIME
        POS+ALALL
        ;
COLD18: COLD19
        P19$    ;PRE HEAT TEMP
.FLT2  200.    ;      S$$      = 0      ;SET POINT
.FLT2  0.      ;      X$$      = 4      ;PROCESS
.FLT2  350.    ;      HL$      = 10     ;HIGH LEVEL
.FLT2  50.     ;      LL$      = 14     ;LOW LEVEL
.FLT2  2.      ;      G1$
.FLT2  .05     ;      P5
.FLT2  0.      ;      U$$
.FLT2  0.      ;      SE$
        1      ;      SAMPLE TIME
        POS+ALALL
        ;
COLD19: COLD20
        P20$    ;BYPASS TEMP
.FLT2  300.    ;      S$$      = 0      ;SET POINT
.FLT2  0.      ;      X$$      = 4      ;PROCESS
.FLT2  650.    ;      HL$      = 10     ;HIGH LEVEL
.FLT2  50.     ;      LL$      = 14     ;LOW LEVEL
.FLT2  10.     ;      G1$
.FLT2  .02     ;      P2
.FLT2  0.      ;      U$$
.FLT2  0.      ;      SE$

```

```

1          ;          SAMPLE TIME
POS+ALALL ;
COLD20: COLD21
P21$
.FLT2 0.          ;
.FLT2 0.          ;
.FLT2 0.          ;
.FLT2 0.          ;
.FLT2 -.05        ;02 = -.05*Sf + 4
.FLT2 4.          ;02 = -.05*Sf + 4
COLD21: COLD22
P22$
.FLT2 0.          ;TABLE OF TRANSDUCER READINGS
.FLT2 4.5
.FLT2 10.
.FLT2 13.
.FLT2 18.
.FLT2 26.
.FLT2 33.
.FLT2 43.
.FLT2 100.
;
COLD22: COLD23
P23$
.FLT2 0.          ;TABLE OF TRUE FLOWS
.FLT2 34.
.FLT2 49.
.FLT2 57.
.FLT2 63.
.FLT2 80.
.FLT2 84.
.FLT2 91.
.FLT2 100.
COLD23: COLD23
COLD23
;

```

; THE FOLLOWING IS A CONSOLE MESSAGE
 ; DESCRIBING SIGNALS AND CHANnELS
 ASC\$: ; ANALOG SIGNAL SCALING

```

;
.ASCIII /VARB          CHAN    MIN    MAX    STORED/
.BYTE      CR,LF
.ASCIII /              OVOLTS    10VOLTS/
.BYTE      CR,LF
.ASCIII /FD DAMPER      1        0    100    10/
.BYTE      CR,LF
.ASCIII /OIL FLOW       2        0    100    100/
.BYTE      CR,LF
.ASCIII /FURNACE DRAFT  3        0    -0.5    160/
.BYTE      CR,LF
.ASCIII /WATER VALVE    4        0    100    40/
.BYTE      CR,LF
.ASCIII /DRUM LEVEL     5       -10    10    170/
.BYTE      CR,LF
.ASCIII /O2             6        0.1    10    110/
.BYTE      CR,LF
.ASCIII /OPACITY        7        0     50    80/
.BYTE      CR,LF
.ASCIII /STACK TEMP     8        50    650    200/
.BYTE      CR,LF
.ASCIII /STEAM PRESS     9        0    200    130S _140G/
.BYTE      CR,LF
.ASCIII /STEAM FLOW    10        0     40    150S _180L/
.BYTE      CR,LF
.ASCIII /AIR FLOW      11        0    100    70/
.BYTE      CR,LF
.ASCIII /REG AIR TEMP  12       50    350    190/
.BYTE      CR,LF
.ASCIII /ID DAMPER     13        0    100    30/
.BYTE      CR,LF
.ASCIII /OIL VALVE     14        0    100    20/
.BYTE      CR,LF
.ASCIII /BYPASS DAMPER 15        0    100    60/
.BYTE      CR,LF
.ASCIII /STEAM VALVE   16        0    100    50/
.BYTE      CR,LF
ASCE$:

```

```

;
SLOPE$: ;SLOPE FOR LINEAR EXTRAPOLATION
.FLT2 2.44200E-2 ; FD DAMPER
.FLT2 2.442E-2 ; OIL FLOW
.FLT2 -1.22100E-4 ; FURNACE DRAFT
.FLT2 2.44200E-2 ; WATER VALVE
.FLT2 4.88400E-3 ; DRUM LEVEL
.FLT2 2.41758E-3 ; O2
.FLT2 1.22100E-2 ; OPACITY
.FLT2 0.14652 ; STACK TEMP
.FLT2 4.88400E-2 ; STEAM PRESS
.FLT2 0.625 ; STEAM FLOW
.FLT2 2.442E-2 ; AIR FLOW
.FLT2 7.32601E-2 ; REG AIR TEMP
.FLT2 2.44200E-2 ; ID DAMPER
.FLT2 2.44200E-2 ; OIL VALVE
.FLT2 2.44200E-2 ; BYPASS DAMPER
.FLT2 2.44200E-2 ; STEAM VALVE

```

```

;-----
INTER$: ; INTERCEPT FOR LINEAR EXTRAPOLATION
.FLT2 0 ; FD DAMPER
.FLT2 0 ; OIL FLOW
.FLT2 0 ; FURNACE DRAFT
.FLT2 0 ; WATER VALVE
.FLT2 -10 ; DRUM LEVEL
.FLT2 0.1 ; O2
.FLT2 0 ; OPACITY
.FLT2 50 ; STACK TEMP
.FLT2 0 ; STEAM PRESS
.FLT2 0 ; STEAM FLOW
.FLT2 0 ; AIR FLOW
.FLT2 50 ; REG AIR TEMP
.FLT2 0 ; ID DAMPER
.FLT2 0 ; OIL VALVE
.FLT2 0 ; BYPASS DAMPER
.FLT2 0 ; STEAM VALVE

```

```

;-----
FILCO$: ; FILTER CONST AND SQUARE ROOT
0,0 ; FD DAMPER
10,0
0,0 ; FURNACE DRAFT
0,0 ; WATER VALVE
0,0 ; DRUM LEVEL
0,0 ; O2
0,0 ; OPACITY
0,0 ; STACK TEMP
0,0 ; STEAM PRESS
0,1 ; STEAM FLOW
0,0 ; AIR FLOW
0,0 ; REG AIR TEMP
0,0 ; ID DAMPER

```

0,0	; OIL VALVE
0,0	; BYPASS DAMPER
0,0	; STEAM VALVE

;-----
ENTRY\$: ; ENTRY POINTS FOR ENGINEERING UNITS

X\$\$\$+P1\$	;FD DAMPER
X\$\$\$+P10\$	;OIL FLOW
X\$\$\$+P16\$	;FURNACE DRAFT
X\$\$\$+P4\$	;WATER VALVE
X\$\$\$+P17\$	;DRUM LEVEL
X\$\$\$+P11\$	;O2
X\$\$\$+P8\$	;OPACITY
X\$\$\$+P20\$	;STACK TEMP
X\$\$\$+P13\$	;STEAM PRESS
X\$\$\$+P15\$	;STEAM FLOW
PB1\$	;AIR FLOW
X\$\$\$+P19\$	;REG AIR TEMP
X\$\$\$+P3\$	;ID DAMPER
X\$\$\$+P2\$	;OIL VALVE
X\$\$\$+P6\$	;BYPASS DAMPER
X\$\$\$+P5\$	;STEAM VALVE

;-----
;

```

; ENTRIES FOR THE PARALLEL I/O HANDLER
XB      = V6$           ;BYPASS DAMPER
XS      = V5$           ;STEAM PRE HEAT
XW      = V4$           ;FEED WATER VALVE
XI      = V3$           ;ID DAMPER
XO      = V2$           ;OIL VALVE
XF      = V1$           ;FD DAMPER
;
BT      = PBL12$        ;BYPASS TEMP CONTROL
PT      = PBL11$        ;PRE HEAT TEMP CONTROL
DLF     = PBL10$        ;DRUM LEVEL, FEED FORWARD
DL      = PBL9$         ;DRUM LEVEL
FD      = PBL8$         ;FURNACE DRAFT
SPF     = PBL7$         ;STEAM PRESSURE, FEED FORWARD
SPG     = PBL6$         ;STEAM PRESSURE, GAP
SPS     = PBL5$         ;STEAM PRESSURE, STD
AL      = PBL4$         ;AIR LEAD
OF      = PBL3$         ;OIL FLOW
OL      = PBL2$         ;OPACITY LIMITS           NOTE, ITS
NOW DRAFT FEED FORWARD !!!!!
AF      = PBL1$         ;AIR FLOW
;
      ACTIV$ = XF+XO+XI+XW+XS+XB
      ACTIP$ = AF+OL+OF+AL+SPS+SPG+SPF+FD+DL+DLF+PT+
BT
;

```

; STATUS ON QUALIFICATION
STON\$:

0,0 ;XB = V6\$;BYPASS DAMPER
0,0 ;XS = V5\$;STEAM PRE HEAT
0,0 ;XW = V4\$;FEED WATER VALVE
0,0 ;XI = V3\$;ID DAMPER
0,0 ;XO = V2\$;OIL VALVE
0,0 ;XF = V1\$;FD DAMPER

; ;
XB,0 ;BT = PBL12\$;BYPASS TEMP CONTROL
XS,0 ;PT = PBL11\$;PRE HEAT TEMP CONTROL
XW,DL ;DLF = PBL10\$;DRUM LEVEL, FEED FORWARD
XW,0 ;DL = PBL9\$;DRUM LEVEL
XI,0 ;FD = PBL8\$;FURNACE DRAFT
XO+XF,SPS+AL+OF+AF ;SPF = PBL7\$;STEAM PRESSURE,
FEED FORWARD

XO+XF,AL+OF+AF ;SPG = PBL6\$;STEAM PRESSURE, GAP
XO+XF,AL+OF+AF ;SPS = PBL5\$;STEAM PRESSURE, STD
XO+XF,OF+AF ;AL = PBL4\$;AIR LEAD
XO,0 ;OF = PBL3\$;OIL FLOW
XF+XI,FD+AF ;OL = PBL2\$;OPACITY LIMITS

NOTE!!!!!!

XF,0 ;AF = PBL1\$;AIR FLOW
;

; STATUS OFF REQUIRMENTS
STOF\$:

```

XB,BT    ;XB    = V6$      ;BYPASS DAMPER
XS,PT    ;XS    = V5$      ;STEAM PRE HEAT
XW,DL+DLF ;XW    = V4$      ;FEED WATER VALVE
XI,FD+OL  ;XI    = V3$      ;ID DAMPER
XO,OF+AL+SPS+SPG+SPF ;XO    = V2$      ;OIL VALVE
XF,AF+AL+SPS+SPG+SPF+OL ;XF    = V1$      ;FD DAMPER
;
O,BT      ;BT    = PBL12$    ;BYPASS TEMP CONTROL
O,PT      ;PT    = PBL11$    ;PRE HEAT TEMP CONTROL
O,DLF     ;DLF    = PBL10$    ;DRUM LEVEL, FEED FORWARD
O,DL+DLF  ;DL     = PBL9$     ;DRUM LEVEL
O,FD+OL   ;FD     = PBL8$     ;FURNACE DRAFT
O,SPF     ;SPF    = PBL7$     ;STEAM PRESSURE, FEED FORWARD
O,SPG+SPF ;SPG    = PBL6$     ;STEAM PRESSURE, GAP
O,SPS+SPF ;SPS    = PBL5$     ;STEAM PRESSURE, STD
O,AL+SPS+SPG+SPF ;AL    = PBL4$     ;AIR LEAD
O,OF+AL+SPS+SPG+SPF ;OF    = PBL3$     ;OIL FLOW
O,OL      ;OL     = PBL2$     ;OPACITY LIMITS NOTE!!!
O,AF+AL+SPS+SPG+SPF+OL ;AF    = PBL1$     ;AIR FLOW
;

```

; BUMPLESS TRANSFER DATA TABLE
BMPTB\$:

DAMPER	ST\$+P6\$,XBON	;XB	= V6\$	;BYPASS
HEAT	ST\$+P5\$,XSON	;XS	= V5\$	;STEAM PRE
VALVE	ST\$+P4\$,XWON	;XW	= V4\$	;FEED WATER
	ST\$+P3\$,XION	;XI	= V3\$	;ID DAMPER
	ST\$+P2\$,XOON	;XO	= V2\$	;OIL VALVE
	ST\$+P1\$,XFON	;XF	= V1\$	;FD DAMPER
	;ST\$+P20\$,BTON	;BT	= PBL12\$	;BYPASS TEMP
CONTROL	ST\$+P19\$,PTON	;PT	= PBL11\$	;PRE HEAT
TEMP CONTROL	ST\$+P18\$,DLFON	;DLF	= PBL10\$	;DRUM LEVEL,
FEED FORWARD	ST\$+P17\$,DLON	;DL	= PBL9\$	;DRUM LEVEL
	ST\$+P16\$,FDON	;FD	= PBL8\$	;FURNACE
DRAFT	ST\$+P15\$,SPFON	;SPF	= PBL7\$	;STEAM
PRESSURE, FEED FORWARD	ST\$+P14\$,SPGON	;SPG	= PBL6\$	;STEAM
PRESSURE, GAP	ST\$+P13\$,SPSON	;SPS	= PBL5\$	;STEAM
PRESSURE, STD	ST\$+P12\$,ALON	;AL	= PBL4\$	;AIR LEAD
	ST\$+P10\$,OFON	;OF	= PBL3\$	;OIL FLOW
	ST\$+P9\$,OLON	;OL	= PBL2\$	;OPACITY
LIMITS	ST\$+P7\$,AFON	;AF	= PBL1\$	;AIR FLOW
	BMPED			
	BMPED			
	;X\$\$\$+P6\$,S\$\$\$+P6\$	;XB	= V6\$	;BYPASS
XBON: DAMPER	X\$\$\$+P5\$,S\$\$\$+P5\$	;XS	= V5\$	;STEAM PRE
XSON: HEAT	X\$\$\$+P4\$,S\$\$\$+P4\$	;XW	= V4\$	;FEED WATER
XWON: VALVE	X\$\$\$+P3\$,S\$\$\$+P3\$	;XI	= V3\$	;ID DAMPER
XION:	X\$\$\$+P2\$,S\$\$\$+P2\$	;XO	= V2\$	;OIL VALVE
XOON:	X\$\$\$+P1\$,S\$\$\$+P1\$	;XF	= V1\$	;FD DAMPER
XFON:	;BT			
BTON:	= PBL12\$			;BYPASS TEMP CONTROL
	S\$\$\$+P6\$,SE\$+P20\$			
PTON:	;PT			
	= PBL11\$			;PRE HEAT TEMP CONTROL
	S\$\$\$+P5\$,SE\$+P19\$			
DLFON:	;DLF			
	= PBL10\$			;DRUM LEVEL, FEED FORWARD
DLON:	;DL			
	= PBL9\$			;DRUM LEVEL

S\$\$+P4\$, SE\$+P17\$
 FDON:;FD = PBL8\$;FURNACE DRAFT
 S\$\$+P3\$, SE\$+P16\$
 SPFON: ;SPF = PBL7\$;STEAM PRESSURE, FEED
 FORWARD
 SPGON: S\$\$+P12\$, SE\$+P14\$;SPG =
 PBL6\$;STEAM PRESSURE, GAP
 SPSON: S\$\$+P12\$, SE\$+P13\$;SPS =
 PBL5\$;STEAM PRESSURE, STD
 ALON: S\$\$+P10\$, S\$\$+P12\$;AL = PBL4\$;AIR
 LEAD
 S\$\$+P10\$, X\$\$+P12\$
 FLT0, DA\$+P12\$
 OFON: X\$\$+P10\$, S\$\$+P10\$;OF = PBL3\$;OIL
 FLOW
 S\$\$+P2\$, SE\$+P10\$
 OLON: ;OL = PBL2\$;OPACITY LIMITS NOTE!!!!
 AFON: X\$\$+P7\$, S\$\$+P7\$;AF = PBL1\$;AIR FLOW
 S\$\$+P1\$, SE\$+P7\$
 FLT1, SE\$+P8\$;OPACITY
 BMPED: ;END OF LIST
 FLT0: .FLT2 0.
 ;

```

;      AIR FUEL BOOST ROUTINE
;
      LGD$      = NHL$
      LGU$      = NLL$
      DF$       = 60
      DA$       = 64
      SA$       = SE$
      GD$       = TH$
      GU$       = U$$

;
;      BEGINING OF PROGRAM
LAGCO$: TST      ST$(R4)          ;TEST IF LOOP IS IN AUTO
      BMI      LAGO              ;BRANCH IF YES
      RTS      PC                ;OTHERWISE, EXIT
;
;      CHECK SET POINT FOR UPPER LIMITS
LAGO:  MOV      R0,-(SP)
      MOV      R1,-(SP)          ;SAVE A REGISTER
      MOV      R2,-(SP)
      MOV      R3,-(SP)
      MOV      HL$+2(R5),-(SP)
      MOV      HL$(R5),-(SP)
      MOV      S$$+2(R4),-(SP)
      MOV      S$$ (R4),-(SP)
      FSUB     SP
      BGE      LAGA              ;IF (HL-S) RESET,OK,OK
      MOV      HL$(R5),S$$ (R4)
      MOV      HL$+2(R5),S$$+2(R4)
;
;      CHECK SET POINT FOR LOWER LIMITS
LAGA:  CMP      (SP)+,(SP)+      ;POP CELLS
      MOV      LL$+2(R5),-(SP)
      MOV      LL$(R5),-(SP)
      MOV      S$$+2(R4),-(SP)
      MOV      S$$ (R4),-(SP)
      FSUB     SP
      BLE      LAGB              ;IF (LL-S) OK,OK, RESET
      MOV      LL$(R5),S$$ (R4)
      MOV      LL$+2(R5),S$$+2(R4)
LAGB:  CMP      (SP)+,(SP)+      ;POP CELLS
;      SET UP ARGUYMENTS AND COMPUTE CONTROL ERROR
      MOV      S$$+2(R4),-(SP)
      MOV      S$$ (R4),-(SP)
      MOV      X$$+2(R4),-(SP)
      MOV      X$$ (R4),-(SP)
      FSUB     SP                ; E = S - X
      BIC      GAPSG$,ST$(R4)    ;CLEAR SIGN BIT
      TST      (SP)              ;TEST ERROR SIGN
      BPL      LAGC              ;BRANCH IF POSITIVE
      BIS      GAPSG$,ST$(R4)    ;SET NEGATIVE BIT
LAGC:

```

```

        BIC      100000,(SP)      ; ABSOLUTE VALUE FOR ERROR
        MOV      G1$(R4),R2      ; GET SLEW RATE
        MOV      G1$+2(R4),R3    ; FOR THRESHOLD CHECK
        ;
; CHECK ERROR FOR THRESHOLD
        ;
        MOV      R3,-(SP)        ; PUT SLEW ON STACK
        MOV      R2,-(SP)
        ;
LAG1A:   FSUB     SP              ; SUBTRACT OFF THRESHOLD
        TST      (SP)           ; TEST NEW ERROR
        BPL      LAG1C          ; BRANCH IF POSITIVE
LAG1B:   CLR      R2             ; ZERO ERROR
        CLR      R3
LAG1C:   MOV      R2,(SP)        ; LOAD SLEW RATE
        BIT      GAPSG$,ST$(R4) ; RAMP DOWN?
        BEQ      LAG1D          ; BRANCH IF NO

        BIS      100000,(SP)    ; MAKE RAMP NEGATIVE
LAG1D:   MOV      R3,2(SP)
        ;
        ;
        ;
LAG2:    MOV      X$$+2(R4),-(SP) ; GET FUEL SUM
        MOV      X$$$(R4),-(SP)
        FADD     SP              ; SUM = ERR + DF
        MOV      (SP),R0
        MOV      2(SP),R1
        ;
; CHECK OUTPUT LIMITS
        ;
        MOV      HL$+2(R5),-(SP) ; HIGH LEVEL FIRST
        MOV      HL$(R5),-(SP)
        FSUB     SP              ; IF (U-HL) OK,OK,RESET
        BLE      LAG2A          ; BRANCH IF OK
        MOV      HL$+2(R5),R1
        MOV      HL$(R5),R0
        ;
LAG2A:   CMP      (SP)+,(SP)+    ; CLEAR CELL
        MOV      R1,-(SP)       ; GET OUTPUT
        MOV      R0,-(SP)
        MOV      LL$+2(R5),-(SP) ; GET LOW LEVEL
        MOV      LL$(R5),-(SP)
        FSUB     SP              ; IF (U-LL) RESET,OK,OK
        BGE      LAG2B          ; BRANCH IF OK
        MOV      LL$+2(R5),R1
        MOV      LL$(R5),R0
LAG2B:   MOV      R0,X$$$(R4)    ; STORE NEW SUM
        MOV      R1,X$$+2(R4)
        MOV      R0,S$$$(R5)    ; CASCADE OUTPUT
        MOV      R1,S$$+2(R5)

```

```

CMP      (SP)+,(SP)+      ; POP CELL
;
MOV      DA$+2(R4),-(SP)      ; GET DELTA
MOV      DA$(R4),-(SP)
BIT      GAPSG$,ST$(R4) ; RAMP UP?
BNE      LAG30A      ; BRANCH IF NO
MOV      LGU$+2(R4),-(SP) ; UP UP LAG
MOV      LGU$(R4),-(SP)
FMUL     SP      ; LA* DELT
MOV      GU$+2(R4),-(SP) ; GAIN
MOV      GU$(R4),-(SP)
MOV      LGU$+2(R4),-(SP) ; UP LAG
MOV      LGU$(R4),-(SP)
BR       LAG30B
;
LAG30A:  MOV      LGD$+2(R4),-(SP) ; DOWN LAG
MOV      LGD$(R4),-(SP)
FMUL     SP      ; LAG * DELT
MOV      GD$+2(R4),-(SP)
MOV      GD$(R4),-(SP)
MOV      LGD$+2(R4),-(SP) ; DOWN LAG
MOV      LGD$(R4),-(SP)
;
LAG30B:  BIS      100000,(SP)      ; -LAG
MOV      FLT1+2,-(SP)      ; GET A 1
MOV      FLT1,-(SP)
FADD     SP      ; 1 - LAG
FMUL     SP      ; (1 - LAG) * GAIN
MOV      R3,-(SP)      ; GET ERROR
MOV      R2,-(SP)
FMUL     SP      ; (1-LAG)GAIN * ERR
;
FADD     SP      ; LAG*DELT + (1-LAG)GAIN*ERR
;
MOV      (SP),DA$(R4)      ; SAVE DELT AIR
MOV      2(SP),DA$+2(R4)
;
;
MOV      X$$+2(R4),-(SP) ; GET FUEL SUM
MOV      X$$$(R4),-(SP)
FADD     SP      ; SUM = FUEL + DELT
;
MOV      (SP),SA$(R4)      ; STORE AIR SUM
MOV      2(SP),SA$+2(R4)
;
MOV      U$$+P11$+2,-(SP)      ; GET AIR RATIO
MOV      U$$+P11$,-(SP)
FMUL     SP      ; AIR = AIR SUM *
RATIO
MOV      U$$+P8$+2,-(SP) ; GET AIR RATIO
MOV      U$$+P8$,-(SP)

```

```
RATIO * FMUL      SP                ;AIR = AIR SUM *
        RATIO
        MOV      (SP)+,S$$+P7$      ;CASCADE OUTPUT
        MOV      (SP)+,S$$+2+P7$
        ;
        ;
        MOV      (SP)+,R3
        MOV      (SP)+,R2
        MOV      (SP)+,R1
        MOV      (SP)+,R0
        RTS      PC
;
;
```

APPENDIX C

MACRO MAP\$

The previous two appendices contain listings of complete software packages. Both make heavy use of system library software. Appendices C through M are listings of those standard macros which both systems call. They are not necessary for understanding the control system software on a function-by-function basis, but are included to document the software in a complete, detailed manner. All macros used in this work are written in MACRO 11 assembly language with liberal use of MACRO assembler directives for listing, page control, and conditional assembly. As such, it should be kept in mind that an interpretation and understanding of their operation should be attempted only by those with a firm grasp on assembly language fundamentals as well as MACRO 11 assembler directives.

MAP\$ is a macro whose function is to define a standard format use by the control software. It contains only address and symbol definitions. It is the only definition routine in the system and contains sections for:

1. Random Access Memory (RAM) address layout for a standard control system. It defines the locations in an 8k system for twenty control tables, serial line buffers, interrupt clocks, status words, and stack starting point.
2. Control table layout for standard controllers has specified positions for set points, process values, controller tuning parameters, as well as others for controller operation.
3. Parallel I/O hardware addresses
4. Serial I/O hardware addresses
5. Standard interrupt addresses

It is accessed by:

MAP\$ DUM1,DUM2, . . .

where DUMX refers to the functions to be defined as listed at the start of the macro.

```

;
.MACRO MAP$ DUM1,DUM2,DUM3,DUM4,DUM5,DUM6,DUM7
.NLIST
; DUMX = LIST MEMORY MAP SEGMENT
;
; RAM = RANDOM ACCESS MEMORY
; ROM = READ ONLY MEMORY
; CON = CONTROL TABLE FORMAT
; PAR = DRV11 STANDARD ADDRESSES
; SER = DLV11 STANDARD ADDRESSES
; VEC = INTERRUPT VECTORS
;
.IRP DUM,<DUM1,DUM2,DUM3,DUM4,DUM5,DUM6,DUM7>
.IF IDN,DUM,CON
.LIST
;
;----- CONTROL TABLE FORMAT
;
S$$ = 0 ;SET POINT
X$$ = 4 ;PROCESS
HL$ = 10 ;HIGH LEVEL
LL$ = 14 ;LOW LEVEL
P1$ = 20 ;GAIN 1
P2$ = 24 ;GAIN 2
TH$ = 24 ;THRESHOLD
U$$ = 30 ; WIND UP & CONTROLLER OUTPUT
SE$ = 34 ;SUM OF ERROR
ST$ = 40 ;AUTO/MAN & SAMPLE TIME
A$$ = 42 ; POS/NEG ACTION & ALLOW/DISALLOW PARAM UPDATE
NHL$ = 44 ;HIGH LIMIT FOR NO OUTPUT CONTROLLER
NLL$ = 50 ;LOW LIMIT FOR NO OUTPUT CONTROLLER
GP$ = 54 ;GAP FOR GAP CONTROL
;
GAP$ = 1000 ;GAP CONTROL
GAPON$ = 2000 ;GAP CONTROL ON
GAPSG$ = 4000 ;GAP SIGN
NOUT$ = 400 ;NO OUTPUT CONTROLLER LOOP
ALALL = 0 ; ALLOW ALL UPDATES
POS = 0 ;POSITIVE ACTION
NGA = 100000 ;NEGATIVE ACTION
;----- PUSH BUTTON FORMAT
RB$ = 1 ;RUN BIT FOR PARALLEL INPUT HANDLER
V1$ = 2
V2$ = 4
V3$ = 10
V4$ = 20
V5$ = 40
V6$ = 100
;
PBL1$ = 2
PBL2$ = 4

```

```

PBL3$ = 10
PBL4$ = 20
PBL5$ = 40
PBL6$ = 100
PBL7$ = 200
PBL8$ = 400
PBL9$ = 1000
PBL10$ = 2000
PBL11$ = 4000
PBL12$ = 10000
.NLIST
.ENDC
.IF IDN,DUM, RAM
.LIST
;
;----- RAM MEMORY -----
;
MEMTO$ = 37400 ;8K SYSTEM, LEAVE ROOM FOR ABS LOADER
;
; CONTROL TABLES
PTTOP$ = MEMTO$
PLG$ = PTTOP$ - 100 ;VARIABLE LOG TABLE 23
PB2$ = PLG$ - 100 ; BUFFER 2 TABLE 22
PB1$ = PB2$ - 100 ; BUFFER 2 TABLE 21
SYLD$ = PB1$+44 ; COMPUTER LOADING POSITION 219
;
P20$ = PB1$ - 100 ; CONTROL TABLE 20
P17$ = P20$ - 100 ; CONTROL TABLE 17
P16$ = P17$ - 100 ; CONTROL TABLE 16
P15$ = P16$ - 100 ; CONTROL TABLE 15
P14$ = P15$ - 100 ; CONTROL TABLE 14
P13$ = P14$ - 100 ; CONTROL TAGLE 13
P12$ = P13$ - 100 ; CONTROL TABLE 12
P11$ = P12$ - 100 ; CONTROL TABLE 11
P10$ = P11$ - 100 ; CONTROL TABLE 10
P07$ = P10$ - 100 ; CONTROL TABLE 07
P06$ = P07$ - 100 ; CONTROL TABLE 06
P05$ = P06$ - 100 ; CONTROL TABLE 05
P04$ = P05$ - 100 ; CONTROL TABLE 04
P03$ = P04$ - 100 ; CONTROL TABLE 03
P02$ = P03$ - 100 ; CONTROL TABLE 02
P01$ = P02$ - 100 ; CONTROL TABLE 01
P00$ = P01$ - 100 ; CONTROL TABLE 00
PTBOT$ = P00$ ;CONTROL TABLE BOTTOM
;
;SERIAL LINE BUFFERS
;
CTTYI$ = PTBOT$ - 200 ;CONSOLE LINE IN
CTTYO$ = CTTYI$ - 200 ; CONSOLE LINE OUT
NTTYI$ = CTTYO$ -200 ; NETWORK LINE IN
NTTYO$ = NTTYI$ - 200 ;NETWORK LINE OUT

```

```

;
;
CK00$ = NTTYO$ - 300 ; INTERRUPT CLOCKS, SPACE FOR 48
TASKS
;
;
TIME$ = CK00$ - 10 ; ELAPSE CLOCK
;
;
; STATUS WORDS
;
STA6A$ = TIME$ - 36 ; STATUS WORD 6A
STA5A$ = STA6A$ - 2
STA4A$ = STA5A$ - 2
STA3A$ = STA4A$ - 2
STA2A$ = STA3A$ - 2
STA1A$ = STA2A$ - 2
STA6$ = STA1A$ - 2 ; 6
STA5$ = STA6$ - 2 ; 5
STA4$ = STA5$ - 2 ; 4
STA3$ = STA4$ - 2 ; 3
STA2$ = STA3$ - 2 ; 2
STA1$ = STA2$ - 2 ; 1
STAOA$ = STA1$ - 2 ; OA RESERVED FOR
STAO$ = STAOA$ - 2 ; O SYSTEM USE
VLBUF$ = STAO$ - 2 ; VALVE DRIVE BUFFER, RESERVED FOR
SYSTEM
DRB1$ = VLBUF$ - 2 ; INPUT PARALLEL LATCH, RESERVED FOR
SYSTEM
DRB2$ = DRB1$ - 2 ; RESERVED FOR SYSTEM
DRB3$ = DRB2$ - 2 ; RESERVED FOR SYSTEM
;
;
SKST$ = 30000 ; STACK START ADDRESS
.NLIST
.ENDC
.IF IDN, DUM, ROM
.LIST
;
; ROM MEMORY GOES HERE
;
.NLIST
.ENDC
.IF IDN, DUM, PAR
.LIST
;
; ----- DRV11 ADDRESSES -----
;
DRCSB1 = 167770 ; DRV11 CONTROL STATUS REGISTER
DROUB1 = 167772 ; DRV11 OUT PUT BUFFER
DRINB1 = 167774 ; DRV11 IN PUT BUFFER

```

```

;
DRCSB2 = 167760      ;DRV11 CONTROL STATUS REGISTER
DROUB2 = 167762      ;DRV11 OUT PUT BUFFER
DRINB2 = 167764      ;DRV11 IN PUT BUFFER
;
DRCSB3 = 167750      ;DRV11 CONTROL STATUS REGISTER
DROUB3 = 167752      ;DRV11 OUT PUT BUFFER
DRINB3 = 167754      ;DRV11 IN PUT BUFFER
.NLIST
.ENDC
.IF IDN,DUM,SER
.LIST
;
;----- DLV11 ADDRESSES -----
;
SRD1 = 177562      ;CONSOLE READER DATA BUFFER
SRS1 = 177560      ;CONSOLE READER STATUS BUFFER
STD1 = 177566      ;CONSOLE TRANSMITTER DATA BUFFER
STS1 = 177564      ;CONSOLE TRANSMITTER STATUS BUFFER
;
SRD2 = 176502      ;NETWORK READER DATA BUFFER
SRS2 = 176500      ;NETWORK READER STATUS BUFFER
STD2 = 176506      ;NETWORK TRANSMITTER DATA BUFFER
STS2 = 176504      ;NETWORK TRANSMITTER STATUS BUFFER
;
SRD3 = 176512      ;DOE NETWORK READER DATA BUFFER
SRS3 = 176510      ;DOE NETWORK READER STATUS BUFFER
STD3 = 176516      ;DOE NETWORK TRANSMITTER DATA BUFFER
STS3 = 176514      ;DOE NETWORK TRANSMITTER STATUS BUFFER
;
SRD4 = 176522      ;SERIAL 4 READER DATA BUFFER
SRS4 = 176520      ;SERIAL 4 READER STATUS BUFFER
STD4 = 176526      ;SERIAL 4 TRANSMITTER DATA BUFFER
STS4 = 176524      ;SERIAL 4 TRANSMITTER STATUS BUFFER
.NLIST
.ENDC
.IF IDN,DUM,VEC
.LIST
;----- INTERRUPT VECTORS -----
; THIS SECTION LOADS THE TRAP VECTORS
; WITH THE CORRECT VECTOR ADDRESSES
;
. = 4      ;TRAP VECTOR 4
.WORD ERTRA$,200
;
. = 10     ;TRAP VECTOR 10
.WORD ERTRA$,200
;
. = 14     ;TRAP VECTOR 14
.WORD ERTRA$,200
;

```

```

. = 24      ;TRAP VECTOR 24
.WORD  CSTART,200
;
. = 60      ;TRAP VECTOR 60
.WORD  S1I$,0 ;CONSOLE SERIAL LINE
.WORD  S1O$,200
S1I$:  JMP  VE60
S1O$:  JMP  CTIN
;
. = 100     ;TRAP VECTOR 100
.WORD  RTM$,0
;
. = 244     ;TRAP VECTOR 244
.WORD  NUTR$,200
;
;
. = 300     ;SERIAL LINE 2 INTERRUPT VECTOR
.WORD  S2I$,0
.WORD  S2O$,200
.WORD  S3I$,0 ;SERIAL LINE 3 INTERRUPT VECTOR
.WORD  S3O$,200
.WORD  S4I$,0 ;SERIAL LINE 4 INTERRUPT VECTOR
.WORD  S4O$,200
. = 400
;
.NLIST
.ENDC
.ENDM
.LIST
.ENDM  MAP$

```

APPENDIX D

MACRO RTMO\$\$

See note regarding appendices C through M at start of Appendix C.

RTMOSS\$\$ is the real time monitor macro for the control software. It was developed to provide several features important for microprocessor digital control. These features include:

1. Minimum overhead both in memory and CPU time requirements.
This allows a low cost system capable of supporting several cascaded control loops.
2. Support of multiple, asynchronous task execution in keeping with a cascade structure.
3. Effective use of reentrant, table driven routines stored in ROM type memory.

The macro makes extensive use of conditional assembly and builds a real time monitor into the software based on the number of tasks required. A task is allowed its own sample rate and can be associated with clocks of various frequencies with rates varying from 1/60 of a second (tick clock) to once per week. Its calling sequence is:

```
RTMO$$      DUMV,DUMT,DUMM, . . .
```

where DUMV, DUMT, etc., are the respective numbers of tasks associated with each clock as described at the top of the listing.

```

;
.MACRO RTMO$$ DUMV,DUMT,DUMS,DUMM,DUMH,DUMD,DUMW
.NLIST
; DUMX = NUMBER OF TASKS ON CLOCK
; V = VALVES
; T = TICKS
; S = SECONDS
; M = MINUTES
; H = HOURS
; D = DAYS
; W = WEEKS
;
; DE-FAULT RAM ADDRESSES
;
; CK00$ = START OF INTERRUPT CLOCKS
; TIME$ = START OF WALL CLOCK
;
; ALL REGISTERS (0-6) SAVED
;
;
.LIST
;-----RTMOO.MAC-----
;----REAL----TIME----MONITOR----
; THIS PROGRAM CONTAINS THE TIME CLOCKS AND THE INTERRUPT
CLOCKS
; CLOCK TABLES:
; TIME$=BASE ADDRESS
; TIME$+00 TICK CLOCK
; TIME$+01 SEC CLOCK
; TIME$+02 MIN CLOCK
; TIME$+03 HR CLOCK
; TIME$+04 DAY CLOCK
; TIME$+05 WEEK CLOCK
; INTERRUPT CLOCK TIMERS
TT$=0
SS$=1
MM$=2
HH$=3
DD$=4
WW$=5
;
;
.NLIST
;
;
;

```

```

;
; UTILITY MACROS
; USED FOR BUILDING THE REAL TIME MONITOR
;
.MACRO CODE1$
    DUM3 = 0
    .IF NB,DUM1
    .REPT DUM1
        CLOCK1 DUM2,DUM3
    .ENDR
    .ENDC
    .ENDM CODE1$
;
.MACRO CODE2$
    DUM3 = 0
    .REPT DUM1
        CLOCK3 DUM2,DUM3
    .ENDR
    .ENDM CODE2$
;
.MACRO CODE3$ DUM2,DUM3,DUM4
    .LIST
;
V$0'DUM3: MOV PO'DUM3'$,R4
    JSR PC,VALVE$
    TSTB ST$(R4) ;TEST FOR VALID SAMPLE TIME
    BGT Q$0'DUM3
    MOVB 10,ST$(R4)
Q$0'DUM3: MOVB ST$(R4),(R1) ;LOAD CLOCK COUNT
    TST ST$(R4) ;TEST FOR ACTIVE LOOP
    BPL V$0'DUM3'A ;STOP VALVE IF NOT
    MOV DUM2,R2 ;SET VALVE POINTER
    MOVB U$$(R4),R0 ;GET VALVE COUNT
    JSR PC,VOUT$ ;SET VALVE BUFFER
    RTS PC ;RETURN TO OPERATING SYSTEM
;
V$0'DUM3'A: BIC DUM4,VLBUF$ ;STOP VALVE
    RTS PC ;RETURN TO OPERATING SYSTEM
    .NLIST
    .ENDM CODE3$
;
.MACRO CLOCK1 DUM2,DUM3
    CLOCK2 dUM2,dUM3
    DUM3 = DUM3+1
    .ENDM
;
.MACRO CLOCK2 DUM2,DUM3
    .LIST
    MOV CK$'DUM2'DUM3,R1
    JSR PC,DTIME$
    .NLIST

```

```

.ENDM  CLOCK2$
;
.MACRO  CLOCK3  DUM2,DUM3
  CLOCK2  dUM2,dUM3
  CLOCK4  dUM2,dUM3
  DUM3    = DUM3 + 1
.ENDM  CLOCK3
;
.MACRO  CLOCK4  DUM2,DUM3
.LIST
  MOV     CK$'DUM2'DUM3'A,R1
  JSR     PC,DTIME$
.NLIST
.ENDM  CLOCK4
;
.MACRO  MADR1$  DUM1,DUM2
  DUM3    = 0
.REPT    DUM1
  MADR2$  dUM2,dUM3,cK00$$
  CK00$$  = CK00$$ + 4
  DUM3    = DUM3 + 1
.ENDR
.ENDM
;
.MACRO  MADR2$  DUM2,DUM3,CK00$$
.LIST
  CK$'DUM2'DUM3    = CK00$$
.NLIST
.ENDM
;
.MACRO  MADA2$  DUM2,DUM3,CK00$$
.LIST
  CK$'DUM2'DUM3'A  = CK00$$
.NLIST
.ENDM  MADA2$
;
.MACRO  MADR3$  DUM1,DUM2
  DUM3    = 0
.REPT    DUM1
  MADR4$  dUM2,dUM3,<DUM3+1*4>
  DUM3    = DUM3 + 1
.ENDR
.ENDM  MADR3$
;
.MACRO  MADR4$  DUM2,DUM3,DUM4
.LIST
;  CK$'DUM2'DUM3
  DUM4
  V$'DUM2'DUM3
.NLIST
.ENDM  MADR4$

```

```
;
.MACRO MADA4$ DUM2,DUM3
.LIST
  0
  V$'DUM2'DUM3'A
.NLIST
.ENDM MADA4$
```

```

; SET UP RAM TABLES
; VALVES FIRST
.IIF NB,DUMV, DUM1 = DUMV
.IIF B,DUMV, DUM1 = 0
  DUM2 = 0
  DUM3 = 0
  CK00$$ = CK00$
.REPT DUM1
  MADR2$ dUM2,dUM3,cK00$$
  CK00$$ = CK00$$ + 4
  MADA2$ dUM2,dUM3,cK00$$
  CK00$$ = CK00$$ + 4
  DUM3 = DUM3 + 1
.ENDR
.LIST
;
.NLIST
  DUM2 = 1
.IRP DUM1,<DUMT,DUMS,DUMM,DUMH,DUMD,DUMW>
.IF NB,DUM1
  MADR1$ DUM1,DUM2
.ENDC
  DUM2 = DUM2+1
.LIST
;
.NLIST
.ENDM

```

```

.LIST
.PAGE          ;PAGE CONTROL
RTM$: MOV R0,-(SP) ;SAVE  REGISTERS ON STACK
      MOV R1,-(SP)
      MOV R2,-(SP)
      MOV R3,-(SP)
      MOV R4,-(SP)
      MOV R5,-(SP)
TICK$: INCB TT$+TIME$
      CMPB TT$+TIME$, 60.
      BMI OUTT$
      CLRB TT$+TIME$
SEC$: INCB SS$+TIME$
      CMPB SS$+TIME$, 60.
      BMI OUTS$
      CLRB SS$+TIME$
MIN$: INCB MM$+TIME$
      CMPB MM$+TIME$, 60.
      BMI OUTM$
      CLRB MM$+TIME$
HR$: INCB HH$+TIME$
      CMPB HH$+TIME$, 24.
      BMI OUTH$
      CLRB HH$+TIME$
DAY$: INCB DD$+TIME$
      CMPB DD$+TIME$, 7.
      BMI OUTD$
      CLRB DD$+TIME$
WK$: INCB WW$+TIME$
      CMPB WW$+TIME$, 57.
      BMI OUTW$
      CLRB WW$+TIME$
;
.PAGE          ;PAGE CONTROL
OUTW$: ; BEGINNING OF WEEK CLOCK CHAIN
.NLIST
.IF NB,DUMW
      DUM1 = DUMW
      DUM2 = 6
      CODE1$
.ENDC
.LIST
;
OUTD$: ; BEGINNING OF DAY CLOCK CHAIN
.NLIST
.IF NB,DUMD
      DUM1 = DUMD
      DUM2 = 5
      CODE1$
.ENDC
.LIST

```

```

;
OUTH$: ; BEGINNING OF HOUR CLOCK CHAIN
.NLIST
.IF NB,DUMH
    DUM1 = DUMH
    DUM2 = 4
    CODE1$
.ENDC
.LIST
;
OUTM$: ; BEGINNING OF MIN CLOCK CHAIN
.NLIST
.IF NB,DUMM
    DUM1 = DUMM
    DUM2 = 3
    CODE1$
.ENDC
.LIST
;
OUTS$: ; BEGINNING OF SECONDS CLOCK CHAIN
.NLIST
.IF NB,DUMS
    DUM1 = DUMS
    DUM2 = 2
    CODE1$
.ENDC
.LIST
;
OUTT$: ; BEGINNING OF TICK CLOCK CHAIN
.NLIST
.IF NB,DUMT
    DUM1 = DUMT
    DUM2 = 1
    CODE1$
.ENDC
.LIST
; BEGINNING OF VALVE CHAIN
.NLIST
.IF NB,DUMV
    DUM1 = DUMV
    DUM2 = 0
    CODE2$
.ENDC
.LIST
;
; RESTORE REGISTERS
MOV (SP)+,R5
MOV (SP)+,R4
MOV (SP)+,R3
MOV (SP)+,R2
MOV (SP)+,R1

```

```
MOV    (SP)+, R0  
RTI
```

```
;  
;  
;
```

```
.PAGE          ;PAGE CONTROL
;-----DTIME: CLOCK TIMER ROUTINE, ISSUES INTERRUPT
;
DTIME$:  TSTB  (R1)  ;TEST IF CLOCK IS ACTIVE
        BLE  OUT$  ;IF NOT SKIP TO OUT
        DECB  (R1)
        BGT  OUT$
        JSR  PC,@2(R1)  ;IF YES CALL ROUTINE
OUT$:    RTS   PC
```

```

.NLIST
; SET UP COLD START TABLE
.LIST
CKTB$: CKTBE$ ; COLD START TABLE TO REFRESH RAM
CK00$
.NLIST
. IF NB,DUMV
    DUM1 = DUMV
    DUM2 = 0
    DUM3 = 0
.REPT DUM1
    MADR4$ dUM2,dUM3,<DUM3+1*4>
    MADA4$ dUM2,dUM3
    DUM3 = DUM3 + 1
.ENDR
.ENDC
.LIST
;
.NLIST
;
DUM2 = 1
.IRP DUM1,<DUMT,DUMS,DUMM,DUMH,DUMD,DUMW>
. IF NB,DUM1
    MADR3$ DUM1,DUM2
.ENDC
DUM2 = DUM2 + 1
.LIST
;
.NLIST
.ENDM
.LIST
CKTBE$: CKTBE$ ;END OF BLOCK
CKTBE$ ;END OF LOAD
.NLIST

```

```

;
.IF NB,DUMV ;GOT ANY VALVES ?
.LIST
.PAGE
;
; VALVE VECTORS
;
.NLIST
DUM1 = DUMV
DUM2 = 2
DUM3 = 0
DUM4 = 3
.REPT DUM1
CODE3$ dUM2,dUM3,dUM4
DUM2 = DUM2 * 4 ;BIT SHIFT
DUM3 = DUM3 + 1
DUM4 = DUM4 * 4
.ENDR
.LIST
;
;
VOUT$: BEQ VOUT1$ ;STOP VALVE MOVEMENT
BMI VOUT2$ ;CLOSE VALVE
BIC 177600,R0 ;OPEN VALVE
MOV R0,4(R1) ;LOAD CLOCK COUNT
BIC R2,VLBUF$ ;STOP VALVE
ASR R2
BIS R2,VLBUF$ ;OPEN VALVE
RTS PC ;RETURN
;
VOUT1$: BIC R2,VLBUF$ ;STOP VALVE
ASR R2
BIC R2,VLBUF$
RTS PC
;
VOUT2$: BIC 177600,R0 ;STRIP TO 7 BITS
MOV R0,4(R1) ;SET TIME OUT CLOCK
MOV R2,R0
ASR R2
BIC R2,VLBUF$ ;STOP VALVE
BIS R0,VLBUF$ ;CLOSE VALVE
RTS PC ;RETURN
V$EX: TSTB ST$(R4) ;IS SAMPLE TIME VALID?
BNE V$EX1 ;BRANCH IF YES
MOVB 1,ST$(R4)
V$EX1: MOVB ST$(R4),(R1) ;RELOAD INTERRUPT CLOCK
RTS PC
.NLIST
.ENDC
.LIST
.ENDM RTMO$$

```

APPENDIX E

MACRO DATA\$\$

See note regarding appendices C through M at the beginning of Appendix C.

DATA\$\$ is the serial line handler and cold start routine macro. The serial line allows communication with any external device such as a teletype. The handler not only provides the basic support of the I/O hardware, but also has several commands for observing and modifying internal control parameters such as setpoints or tuning values.

A table listing all integers is therefore required for DATA\$\$ to distinguish between them and floating point numbers. That table must appear as the first argument in the macro call.

The second function of the DATA\$\$ macro is to provide power RAM memory initialization. Since RAM memory is primarily control tables, DATA\$\$ must have as its second argument a table containing tuning parameters.

```

.MACRO DATA$$ ITGR$$,COLD$$,STAN1$
.NLIST
; ITGR$$ = ADDRESS OF INTEGER VARIABLES TABLE
; COLD$$ = START ADDRESS OF COLD START TABLE
; STAN1$ = NUMBER OF VALID STATUS WORDS, DEFAULT = 2
.IIF B,STAN1$,STANU$ = 2
.IIF NB,STAN1$,STANU$ = STAN1$
.LIST
;
;////////// DATA 11 IN //////////
;
;THIS ROUTINE MONITORS THE SERIAL LINE INTERFACE
; ON AN INTERRUPT BASIS AND GENERATES A TABLE
; OF ASCII CHARACTERS. UPON RECEIT OF AN END
; OF FIELD <CR> CHARACTER, IT TRANSFERS CONTROL TO
; DATA11. DATA 11 IN IS CAPABALE OF HANDLING TWO
; SIMEULTANEOUS INPUTS.
;
; DATA 11 IN
;
; M.MEALS SEPT.'76
;
;
; ALL REGISTERS SAVED
;
; AFTER ENTRY, R0 CONTAINS TABLE ADDRESS
; R1 CONTAINS DEVICE ADDRESS
; R2 CONTAINS THE CHARACTER
;
; ASSIGNMENT SECTION
;
;
;
CR = 15 ;ASCII CARRIAGE RETURN (OCTAL)
LF = 12 ;ASCII LINE FEED (OCTAL)
ESC = 33 ;ASCII ESCAPE
CNTRLR = 22 ;ASCII CONTROL R (OCTAL), NO ECHO REQUEST
CNTRLU = 25 ;ASCII CONTROL U, SCRATCH BUFFER
SPACE = 40 ;ASCII SPACE
COMMA = ' , ;ASCII COMMA
BCDCON = 177760 ;ASCII TO BCD CONVERSION
FLAGWD = 2.
TPOINT = 4 ;INPUT TABLE POINTER
INPUT = 10 ;INPUT BIT
ECHO = 2
INTVAR = 1 ;INTEGE VARIABLE FLAG BIT
ASCII = 60 ; ASCII CONVERSION
ZERO = 0 ; NO COMMENT
SLASH = 057 ;ASCII SLASH (OCTAL)

```

```
RUB   = 177      ;ASCII RUB OUT (DELETE)
BUSY  = 004      ;BUSY BIT
INPUT = 010      ;INPUT BIT
NECHO = 002      ;ECHO BIT
DEVICE = 001     ;DEVICE BIT
DEVMSK = 177776  ;DEVICE BIT MASK
PARMSK = 177600  ;STRIP IT TO 7 BIT ASCII
```

```
;
;
;
```

```
.PAGE
```

```

;//////// SERIAL LINE HANDLER //////////
;
;
VE60:  MOV  R0,-(SP)  ;SAVE  REG
      MOV  R1,-(SP)
      MOV  R2,-(SP)
      MOV  SRD1,R2  ;CONSOLE VECTOR, GET CHAR
      MOV  CTTYI$,R0  ;SET ADDRESS
      MOV  SRS1,R1  ;SET DEVICE TOO
      BR   INPT      ;DECODE IT
;
VE370: MOV  R0,-(SP)  ;SAVE REG
      MOV  R1,-(SP)
      MOV  R2,-(SP)
      MOV  SRD2,R2  ;NETWORK VECTOR,GET CHAR
      MOV  NTTYI$,R0  ;SET ADDRESS
      MOV  SRS2,R1
;
.PAGE

```

```

;
;//////// INPUT SECTION //////////
;
; THIS SECTION TAKES THE CHARACTER, STORES IT IN THE
BUFFER,
; AND IF A <CR> HAS BEEN RECIEVED, TRANSFERS CONTROL TO
THE
; COMMAND DE-CODER
;
;
; REG 0-2 ARE RESTORED ON EXIT
;
;
INPT:  BITB  (R0), INPUT ;IS THE TABLE IN INPUT MODE ?
      BNE  INPTA      ;BRANCH IF YES
;
;
INPTA1: MOVB  INPUT,(R0) ;SET UP INPUT MODE
      MOV  R0,2(R0) ;SET UP MAX ADDRESS
      ADD  64.,2(R0) ;MAX 64. CHAR
      MOV  R0,4(R0) ;SET UP BYTE ADDRESS POINTER
      ADD  6,4(R0) ;MOVE PAST STATUS WORDS
;
INPTA:  BIC   PARMSK,R2 ;STRIP PARITY
      CMPB  R2, LF      ;IGNORE LINE FEEDS
      BEQ  DAIEN      ;EXIT IF FOUND
      CMPB  R2, ESC     ;IGNORE ESCAPE CHAR
      BEQ  DAIEN      ;EXIT IF FOUND
;
      CMPB  R2, RUB     ;LOOK FOR RUBOUTS
      BNE  INPTB      ;BRANCH IF NOT FOUND
      MOVB  SLASH,R2   ;REPLACE RUBOUT WITH A SLASH
;
INPTB:  CMPB  R2, CNTRLR ;CHECK FOR NO ECHO SIGNAL
      BNE  INPTC      ;BRANCH IF NOT FOUND
      BIS  NECHO,(R0)  ;SET NO ECHO BIT
      BR   DAIEN      ;NO NEED TO STORE IT
;
INPTC:  CMPB  R2, CNTRLU ;WAS IT A CONTROL U?
      BNE  INPT1      ;BRANCH IF NO
      CLR  (R0)      ;OTHERWISE, SCRATCH BUFFER
      BR   DAIEN      ;AND EXIT
;
INPT1:  MOVB  R2,@4(R0) ;PUSH NEW CHAR
      BIT   NECHO,(R0) ;IS THE ECHO OFF ?
      BNE  INPT3      ;BRANCH IF IT IS
;
      JSR  R5,FTYP     ;SEND IT
      CMPB  R2, CR     ;WAS IT A CARRIAGE RETURN ?
      BNE  INPT2      ;BRANCH IF NO
      MOVB  LF,R2      ;LOAD A LINE FEED

```

```

    JSR  R5,FTYP      ;SEND IT
;
INPT2:  CMPB  @4(R0), SLASH  ;CHECK FOR RUBOUT REQUEST
        BEQ  RUBOUT      ;BRANCH IF TRUE
INPT3:  CMPB  @4(R0), CR    ;CHECK FOR FIELD TERMINATOR
        BEQ  INPT4      ;BRANCH IF FOUND
        CMP  4(R0),2(R0)  ;CHECK FOR MAX COUNNT
        BGE  DAIER      ;BRANCH TO ERROR IF FULL
        INC  4(R0)      ;INCREMENT POINTER
        BR   DAIEN      ;RETURN TO OPERATING SYSTEM
;
INPT4:  BIS   BUSY,(R0)  ;SET BUSY BIT
        JSR  PC,DATA$    ;DECODE THE MACRO
        BR   DAIEN      ;RETURN TO REAL TIME MONITOR
;
RUBOUT:  MOV   R0,R2      ;CHECK FOR BUFFER EMPTY
        ADD   6,R2      ;POINTER ADDRESS
        CMP   4(R0),R2   ;ADD - MIN ADD) ERR,ERR,OK
        BLE   DAIER      ;SCRATCH BUFFER
        DEC   4(R0)      ;RUBOUT ONE CHAR
        MOVB  @4(R0),R2  ;LOAD OUTPUT CHAR
        JSR   R5,FTYP    ;SEND IT
        BR    DAIEN      ;RETURN TO REAL TIME MONITOR
;
DAIER:   CLR   (R0)      ;CLEAR BUFFER
;
DAIEN:   MOV   (SP)+,R2  ;RESTORE REG
        MOV   (SP)+,R1
        MOV   (SP)+,R0
        RTI      ;EXIT FROM INTERRUPT
.PAGE

```



```

;
;////          INITILIZATION SECTION          //////////
;
;  THIS ROUTINE DETERMINES WHICH TABLE IS READY FOR
DECODING
;  AND LOADS THAT TABLE ADDRESS INTO RO.
;
;  AFTER ENTRY, RO CONTAINS ACTIVE TABLE ADDRESS
;  ALL REG (0-5) ARE SAVED
;
;
;
DATA$:  MOV  RO,-(SP)      ;PUSH RO ONTO THE STACK
;
BIT    BUSY,CTTYI$      ;IS TABLE 0 READY ?
BEQ    DATA1           ; BRANCH IF NOT
;
BIT    INPUT,CTTYI$     ;IS TABLE 0 BEING DECODED ?
BEQ    DATA1           ; YES, TRY TABLE 1
;
MOV    CTTYI$,RO        ;LOAD TABLE POINTER
;
BR     DATA2           ;DECODE TABLE 0
;
DATA1:  BIT    BUSY,NTTYI$ ;IS TABLE 1 READY ?
BEQ    DTBUSY          ;YES, RETURN TO O.S.
;
BIT    INPUT,NTTYI$     ;IS TABLE 1 READY FOR DECODE ?
BEQ    DTBUSY          ;NO, THEN RETURN TO O.S.
;
MOV    NTTYI$,RO        ;LOAD TABLE POINTER
;
;
DTBUSY:
DTBUS1:
MOV    (SP)+,RO         ;POP RO OFF STACK
RTS    PC               ;RETURN TO O.S.
.PAGE
DATA2:  BIC    INPUT,(RO) ;CLEAR INPUT BIT
MOV    R5,-(SP)         ;GET A
MOV    R4,-(SP)         ;    LITTLE
MOV    R3,-(SP)         ;    WORKING
MOV    R2,-(SP)         ;    SPACE
MOV    R1,-(SP)         ;
MOV    STAOA$,-(SP)
MOVB   TPOINT(RO),2(RO) ;LOAD FINAL BYTE ADDRESS
MOV    RO,TPOINT(RO);    RELOAD THE TABLE POINTER
ADD    TPOINT+2,TPOINT(RO) ; AND SET IT STRAIGHT
MOV    SP,STAOA$        ;LOAD STACK FLUSHER
.PAGE

```

```

;
;  /////          DECODE          //////////
;
; THIS SECTION RECIVES AN INSTRUCTION LETTER WITH
; ARGUMENTS AND DECODES IT ACCORDINGLY
;
; REG 1 SCRATCHED
;
DECODE: JSR R5,GET ;GET A CHARACTER, RETURN IT IN CHAR
;
DECO2:  CMPB 'C,R1 ;CHECK IF REQUEST FOR
      BEQ CURREN ; CURRENT VALUE
;
      CMPB 'F,R1 ;CHECK IF REQUEST FOR
      BEQ FPCONN ;FLOATING POINT CONVERSION
;
      CMPB 'S,R1 ;CHECK IF RQUEST FOR
      BEQ STATUU ;STATUS OF LOOP SWITCHES
;
      CMPB 'U,R1 ;CHECK IF REQUEST FOR
      BEQ UPDATT ;UPDATE OF PARAMETERS
;
      CMPB 'L,R1 ;CHECK IF REQUEST FOR
      BEQ LOGOUU ;LOGOUT OF PARAMETERS
;
ERR: JSR R5,CRLQ ;SEND ? ID AND PROMPT CHAR
      MOV STAOA$,SP ;FLUSH STACK
LEAV2: CLR (R0) ;CLEAR TABLE STATUS
      JMP LEAVE1
LEAV1: JMP LEAVE ;LEAVE DATA ROUTINE
;
UPDATT: JMP UPDATE ;GO TO DECODE
STATUU: JMP STATUS$ ; " STATUS
LOGOUU: JMP LOGOUT ; " LOGOUT
FPCONN: JMP FPCON ; " FLOATING POINT CONVERSION
.PAGE

```

```

;
;  ///          CURRENT VALUES          ///
;
; THIS SUBROUTINE SENDS THE CURRENT VALUES OF THE
; PROCESS AND PARAMETERS VARIABLES TO THE OUTPUT
; DEVICE
;
; REG 4 CONTAINS FIRST ADDRESS
; REG 3 CONTAINS SECOND ADDRESS
; REG 1 SCRATCHED
;
;
CURREN:
MOV  'V,R1      ;SEND IDENTIFER CHARACTER
JSR  R5,BTYP    ;SEND IT
JSR  R5,ARG     ;GET FIRST OCTAL ARGUMENT
MOV  R3,R4      ;MOVE FIRST ADDRESS TO R4
JSR  R5,ARG     ;GET SECOND OCTAL ARGUMENT
BR   VALUE     ;BRANCH TO VALUE ROUTINE
;
.PAGE

```

```

;
;//// FLOATING POINT CONV //////////
;
;THIS ROUTINE CONVERTS A DECIMAL NUMBER TO INTERNAL
;FLOATING POINT (PDP 11) BINARY AND DISPLAYS THE RESULT
;
; ON ENTRY, REG 4 MUST CONTAIN ADDRESS OF PARAMETER
; REG 1,2,4 SCRATCHED
;
;
FPCON: JSR PC,RELNO ;CONVERT IT
MOV R4,R1 ;LOAD HIGH ORDER PART
JSR PC,INTOU2 ;PRINT IT
JSR R5,SPACE$ ;SPACE IT OUT
MOV R2,R1 ;LOAD LOW ORDER
JSR PC,INTOU2 ;TYPE IT
JMP LEAVE ;EXIT
;
CPACE$: MOV ',,R1 ;GET A COMMA
JSR R5,BTYP ;LOAD IT INTO THE BUFFER
SPACE$: MOV SPACE,R1 ;GET A SPACE CHARACTER
JSR R5,BTYP
RTS R5
.PAGE

```

```

; ////          VALUE OF VARIABLES          //////////
;
; THIS ROUTINE SENDS THE CURRENT VALUE OF THE PROCESS
; AND PARAMETER TO THE OUTPUT DEVICE
;
; R4 CONTAINS THE ADDRESS OF THE PROCESS
; R3 CONTAINS THE ADDRESS OF THE PARAMETER
; REG 4 SCRATCHED
;
VALUE:  MOV  R3,-(SP)  ;SAVE PARAMETER SELECT NUMBER
        MOV  R4,-(SP)
        JSR  R5,SPACE$ ;SPACE IT OUT WITH A COMMA
        ;
        MOV  (SP)+,R4 ;LOAD FIRST POINTER
        JSR  PC,VALUE1 ; OUTPUT THE PROCESS VALUE
        JSR  R5,CSPACE$
        ;
        MOV  (SP)+,R4 ;RELOAD SECOND ARG
        JSR  PC,VALUE1 ;OUTPUT PARAMETER VALUE
        BR   LEAV1 ;RETURN
.PAGE

```

```

;
;//////          INTEGER UPDATE          //////////
;
;THIS ROUTINE UPDATES AN INTEGER VALUE IN THE P&P TABLES
;
;  ON ENTRY, REG 3 MUST POINT AT PARAMETER
;  REG 3,4 SCRATCHED
;
;
INTIN: JSR  PC,INTER ;GET THE SECOND INTEGER NUMBER
      DEC  R3      ;RESET INTEGER FLAG BIT
      MOV  R4,(R3) ;STORE INTEGER
      JMP  LEAVE   ;RETURN TO OPERATING SYSTEM
;
INTER: CLR  R4      ;R4 WILL CONTAIN INTEGER
INTE1: JSR  R5,GET   ;GET NEXT CHARACTER
      CMPB CR,R1    ;LOOK FOR FIELD TERMINATOR
      BEQ  INTE2    ;BRANCH IF FINISHED
      CMPB '0,R1    ;IS CHAR BELOW ZERO
      BGT  INTEE    ;BRANCH IF YES
      CMPB '9,R1    ;IS R1 ABOVE 9 ?
      BMI  INTEE    ;BRANCH IF YES
      BIC  BCDCON,R1 ;CONVER IT FROM ASCII
      MOV  3,-(SP)   ;CORRECTION FOR EPROM
      ASH  (SP)+,R4  ;OCTAL INPUT, RIGHT ?
      ADD  R1,R4     ;PACK NEW BCD INTO R4
      BR   INTE1     ;KEEP ON A'LOOPING
INTE2: RTS  PC      ;RETURN
INTEE: JMP  ERR     ;JUMP TO ERROR ROUTINE
.PAGE

```



```

;
;  ////          STATUS          //////////
;
; THIS ROUTINE SENDS SEND OUT THE STATUS OF THE
; MANUAL/AUTO SWITCHES COMBINED WITH THE
; REQUESTED SWITCH SETTINGS
;
;  REG 1,2,5 SCRATCHED
;
STATU$: JSR  R5,GET    ;GET THE NEXT CHARACTER
CMPB    '1,R1    ;STATUS 1 UPDATE ?
BGT  STATO    ;BRANCH IF OUT OF BOUNDS
CMPB    '6,R1
BLT  STATO    ;BRANCH IF OUT OF BOUNDS
BIC    177770,R1    ;STRIP ASCII
ASL  R1    ;SHIFT IT
ADD    STA1A$-2,R1    ;PACK IT IN
MOV  R1,R5
BR  STATC

STATO:
MOV  'S,R1    ;LOAD IDENTIFER CHARACTER
JSR  R5,BTYP    ;AND SEND IT
;
MOV  STA1$,R2    ;LOAD POINTER
MOV  STANU$,R5    ;AND LOOP COUNT
;
STATO1: MOV  (R2)+,R1    ;LOAD FIRST STATUS WORD
JSR  PC,INTOU2    ;TYPE IT OUT
;
JSR  R5,SPACE$
;
SOB  R5,STATO1    ;LOOP
;
STATE: JMP  LEAVE    ;RETURN TO REAL TIME MONITOR
;
;
;
STATC: JSR  R5,GET    ;GET THE NEXT CHAR
CMPB    ',,R1    ;FIELD SEPARATOR CHECK
BNE  STATO
JSR  PC,INTER    ;GET THE INTEGER
MOV  R4,(R5)    ;STORE IT
BR  STATE    ;EXIT
.PAGE

```

```

;
;  ///          UPDATE          ///
;
; THIS ROUTINE UPDATES THE CURRENT VALUE OF THE SELECTED
; PARAMETER
;
; ON ENTRY, REG 3 MUST CONTAIN ADDRESS OF VARIABLE
; REG 1,2,3,4 SCRATCHED
;
;
UPDATE:  JSR  R5,ARG  ;GET FIRST OCTAL ARGUMENT
        MOV  R3,R1   ;SET UP FOR INVALID CHECK TO FOLLOW
UPDAT2:
        BIC  77,R1    ;STRIP TO TABLE
        ADD  A$$,R1   ;POINT TO INVALID WORD
        NEG  R2       ;SET R2 UP AS A RIGHT SHIFT COUNTER
        MOV  (R1),R1  ;MOVE VALID FLAG WORD TO R1
        ASH  R2,R1    ;SHIFT BIT INTO CARRY
        ASR  R1       ;ONE MORE FOR GOOD MEASURE
        BCS  INVALID  ;BRANCH IF UPDATE IS INVALID
        BIT  INTVAR,R3 ;UPDATING AN INTEGER ?
        BNE  INTIN    ;BRANCH IF YES
        JSR  PC,RELNO  ;GET SECOND FLOATING POINT NUMBER
        MOV  R4,(R3)+ ;LOAD HIGH ORDER FP WORD
        MOV  R2,(R3)  ;LOAD LOW ORDER FP WORD
        BR   LEAVE    ;LEAVE DATA ROUTINE
INVALID:
        MOV  'I,R1    ;LOAD IDENTIFER MESSAGE
        JSR  R5,BTYP
;
.PAGE

```

```

;
;  ////          LEAVE          //////////
;
; THIS ROUTINE RESTORES THE REGISTERS 0-5
; AND RETURNS CONTROL TO THE OPERATING SYSTEM
;
LEAVE:  CLR  (R0)      ;CLEAR TABLE POINTER
        JSR  R5,CRLF   ;SEND OUT <CR LF>
LEAVE1: JSR  PC,DUMP   ;UN-LOAD I/O BUFFER
        MOV  (SP)+,STAOA$$
        MOV  (SP)+,R1  ;LOAD REG 1
        MOV  (SP)+,R2  ;  AND REG 2
        MOV  (SP)+,R3  ;    ETC.
        MOV  (SP)+,R4  ;    ETC.
        MOV  (SP)+,R5  ;    ETC.
        MOV  (SP)+,R0  ;RESTORE LAST REGISTER
        RTS  PC  ;GOOD BYE !
.PAGE

```

```

;
;  ///          GENERAL PURPOSE ROUTINES          ///
;
;
;  ///          GET          ///
;
; THIS ROUTINE IS A GENERAL PURPOSE CHARACTER
; INPUT ROUTINE. CHARACTER GOES TO REG 1
; REG 0 MUST CONTAIN ADDRESS OF BUFFER
;
GET:  MOVB  @TPOINT(R0),R1  ;PUT NEW R1 INTO R1
      CMPB  2(R0),TPOINT(R0) ; ARE YOU AT THE END OF YOUR ROPE ?
      BMI  ERR1  ; WELL, BRANCH IF YOU'VE GONE TOO FAR
      INC  TPOINT(R0)  ; INCREMENT TABLE POINTER
GET1:  TSTB  R1  ; IGNORE
      BEQ  GET  ; NULLS
      CMPB  SPACE,R1  ; IGNORE
      BEQ  GET  ; SPACES
      RTS  R5  ; RETURN
;
;
.PAGE

```

```

;
;//////////          DUMP          //////////
;
;THIS ROUTINE IS A INTERRUPT DRIVEN SERIAL I/O OUTPUT
;BUFFER HANDLER
;  ALL REG SAVED
;  ON ENTRY, RO MUST POINT TO INPUT BUFFER
;
DUMP:  CMP  RO, CTTYI$  ;CHEKS FOR WHICH TABLE
      BNE  DUMP1
      BIS  100,STS1  ;SET CONSOLE TRANSMITE ENABLE
      RTS  PC        ;RETURN
;
DUMP1:  CMP  RO, NTTYI$  ;NETWORK REQUEST?
      BNE  DUMP2  ;EXIT IF NO
      BIS  100,STS2  ;SET NETWORK TRANSMITE INTERRUPT ENABLE
DUMP2:  RTS  PC        ;RETURN
;
;
;
CTIN:  TST  CTTYO$      ;CONSOLE INTERRUPT SERVICE ROUTINE
      BEQ  CTCL      ;TEST DATA BUFFER
      CMP  CTTYO$+2,CTTYO$  ;CHECK FOR COMPLEATION
      BLO  CTCL      ;EXIT WHEN THRU
      MOVB @CTTYO$,STD1  ;SEND THE NEXT CHAR
      INC  CTTYO$      ;POINTER TO THE NEXT BYTE
      RTI          ;RETURN FROM INTERRUPT
;
CTCL:  BIC  100,STS1  ;DIS-ENABLE CONSOLE TRANSMITTER
INTERRUPTS
      CLR  CTTYO$      ;CLEAR ADDRESS POINTER
      RTI          ;RETURN FROM INTERRUPT
;
NTIN:  TST  NTTYO$      ;NETWORK INTERRUPT SERVICE ROUTINE
      BEQ  NTCL      ;TEST DATA BUFFER
      CMP  NTTYO$+2,NTTYO$  ;CHECK FOR COMPLEATION
      BLO  NTCL      ;EXIT WHEN THRU
      MOVB @NTTYO$,STD2  ;SEND THE NEXT CHAR
      INC  NTTYO$      ;POINTER TO THE NEXT BYTE
      RTI          ;RETURN FROM INTERRUPT
;
NTCL:  BIC  100,STS2  ;DIS-ENABLE CONSOLE TRANSMITTER
INTERRUPTS
      CLR  NTTYO$      ;CLEAR ADDRESS POINTER
      RTI          ;RETURN FROM INTERRUPT
;
.PAGE

```

```

;
;//////////          ARGUMENT          //////////
;
;THIS ROUTINE  GETS PARAMETER ADDRESSES
; FROM THE INPUT DEVICE
;
; ON EXIT, REG 3 CONTAINS ADDRESS OF PARAMETER
; LSB OF REG3 WILL BE SET IF IT POINTS TO AN INTERGER
; REG 1,2 SCRATCHED
;
;
ARG:  CLR  R2      ;CLEAR PREVIOUS INPUT DIGIT
      CLR  R3      ;R3 WILL CONTAIN ADDRESS ARGUMENT
ASCAN: JSR  R5,GET  ;GET A CHARACTER, RETURN IT IN CHAR
      CMPB CR,R1   ;LOOK FOR FIELD TERMINATOR
      BEQ  ARG1    ;BRANCH IF DONE
      CMPB COMMA,R1 ;LOOK FOR FIELD SEPARATOR
      BEQ  ARG1    ;BRANCH IF SEPARATED
      CMPB '0,R1   ;COMPARE IT WITH ASCII ZERO
      BGT  ERR1    ;ERROR IF BELOW ZERO
      CMPB ' ',R1  ;COMPARE IT WITH HEX B
      BMI  ERR1    ;ERROR IF ABOVE B
      BIC  BCDCON,R1 ;CONVER IT FROM ASCII
      MOV  R1,R2   ;SAVE THE CHAR FOR AWHILE
      MOV  3,-(SP)
      ASH  (SP)+,R3 ;OCTAL ADDRESS RIGHT ?
      ADD  R1,R3   ;PACK NEW BCD INTO R3
      BR   ASCAN   ;LOOP UNTIL FINISHED
ARG1: SUB  R2,R3   ;GET P&P TABLE NUMBER
      MOV  3,-(SP)
      ASH  (SP)+,R3 ; AND SPACE IT OUT
      MOV  R2,R1   ;GET TABLE ENTRY
      MOV  2,-(SP)
      ASH  (SP)+,R1 ; AND SPACE IT OUT TOO !
      ADD  R1,R3   ;PACK 'UM UP
      ADD  PTBOT$,R3 ;ADD IN THE TABLE STARTING ADDRESS
      TST  R3      ;CHECK FOR MAX ADDRESS
      BMI  ERR1    ;EXIT IF INVALID
      CMP  R3, PB1$ ;CHECK IF BUFFER TABLE
      BPL  ARG4    ;BRANCH IF YES
;
      MOV  R2,-(SP)
      MOV  ITGR$$,R2 ;POINTER TO INTEGER TABLE
ARG2: CMP  R3,(R2)+ ;IS ADDRESS AN INTEGER ?
      BEQ  ARG2B    ;EXIT IF YES
      BPL  ARG2     ;LOOP IF NOT AN INTEGER
;
      MOV  (SP)+,R2
ARG2A: RTS  R5      ;RETURN
;
ARG2B: MOV  (SP)+,R2 ;RESTORE REG

```

```
INC R3      ;SET INTEGER BIT
RTS R5      ;RETURN
;
;
;
ARG4: CMP R3, PLG$ ;WAS IT A LOG TABLE REQUEST?
      BPL ARG5     ;ALL REALS IF NO
      RTS R5
ARG5: CMP PTTOP$,R3 ;ADDRESS OUT OF RANGE ?
      BMI ERR1     ;JUMP IF YES
;
      INC R3      ;OTHERWISE, SET FOR ALL INTEGERS
      RTS R5      ; AND RETURN
ERR1: JMP ERR     ;JUMP TO ERROR SUBROUTINE
.PAGE
```

```

;
;//////////          FTYP          //////////
;
;THIS ROUTINE PRINTS TO THE OUTPUT DEVICE ON A POLLED
BASIS
; THE CHARACTER CONTAINED IN R2
;
; R1 POINTS TO OUTPUT DEVICE
; R2 CONTAINS CHARACTER
;
FTYP:  TSTB  4(R1)  ;CHECK FOR EMPTY OUTPUT BUFFER
        BPL  FTYP  ;LOOP UNTIL  CLEAR
        MOVB R2,6(R1) ;LOAD R1
FTY3:  RTS   R5     ;RETURN
.PAGE

```

```

;
;//////          CARRIAGE RETURN LINE FEED      //////////
;//////          WITH A PROMPT                    //////////
;
;THIS ROUTINE RESPONDS TO THE TTY WITH A
; CARRIAGE RETURN AND A LINE FEED FOLLOWED
; BY AN OPTIONAL PROMPT CHARACTER '*'
;
; REG 3,4 SCRATCHED
;
;
CRLF:  MOV    O.CR+1,R3  ;SEND A <CR,LF>
      BR    CRS  ;GET THE REST
CRLS:  MOV    O.CR+2,R3  ;SEND A <CR,LF>*
CRS:   MOV    O.CR,R4    ;SET FIRST BYTE LOCATION
CRSE:  JSR    R5,TYPE    ;TYPE IT
      RTS    R5    ;RETURN
CRLQ:  MOV    O.CR+2,R3
      MOV    O.CRQ,R4
      BR    CRSE
;
O.CRQ:  .BYTE  '?,'
O.CR:   .BYTE  012  ;ASCII LF
      .BYTE  015  ;ASCII CR
      .BYTE  '*'
      .EVEN
.PAGE

```

```

;
;//////////          TYPE          //////////
;
;THIS ROUTINE TYPES A MESSAGE ON THE OUTPUT DEVICE
;LOAD STARTING ADDRESS OF MESSAGE IN R4
;LOAD ENDING ADDRESS IN R3  ( R4 < R3 )
;  RO CONTAINS ADDRESS OF OUTPUT DEVICE
;
;
;
TYPE:  MOV  RO,-(SP)  ;SAVE REG
      CMP  RO,CTTYI$  ;REQUEST FROM CONSOLE?
      BNE  TYPE1      ;BRANCH IF NO
      MOV  CTTYO$,RO  ;LOAD POINTER
      BR   TYPE2
;
TYPE1:  CMP  RO,NTTYI$ ;REQUEST FROM THE NETWORK?
      BNE  TYPE4      ;EXIT IF NO
      MOV  NTTYO$,RO  ;LOAD POINTER
;
TYPE2:  TST  (RO)      ;CHECK FOR EMPTY TABLE
      BNE  TYPE3A      ;BRANCH IF NOT EMPTY
      MOV  RO,(RO)      ;LOAD POINTERS
      ADD  4,(RO)      ; STARTING ADDRESS POINTER
      MOV  (RO),2(RO)  ; ENDING ADDRESS POINTER
      BR   TYPE3B
;
TYPE3A:  INC  2(RO)      ;SET POINTER
;
TYPE3B:  MOV  RO,-(SP)  ;CALC END OF TABLE
      ADD  72.,(SP)
TYPE3:  CMP  R3,R4      ;CHECK FOR COMPLETION
      BLO  TYPE4A      ;EXIT WHEN DONE
      MOVB (R4)+,@2(RO) ; MOVE CHAR TO STACK
      CMP  (SP),2(RO)  ;IS THE TABLE FULL
      BLE  TYPE4A      ;EXIT IF YES
      INC  2(RO)      ;INCREMENT POINTER
      BR   TYPE3      ;LOOP UNTIL DONE
;
TYPE4A:  TST  (SP)+      ;POP CELL
      DEC  2(RO)      ;SET POINTER
TYPE4:  MOV  (SP)+,RO    ;RESTORE REG
      RTS  R5          ;RETURN
;
BTYP:  MOV  R3,-(SP)    ;SAVE REGISTERS
      MOV  R4,-(SP)
      MOV  R1,-(SP)
      MOV  SP,R3
      MOV  SP,R4
      JSR  R5,TYPE
      MOV  (SP)+,R1    ;RESTORE REGISTER

```

```
MOV (SP)+, R4  
MOV (SP)+, R3  
RTS R5 ;AND RETURN  
.PAGE
```

```

;
; ////////////// INTEGER OUTPUT //////////////
;
; THIS ROUTINE TYPES OUT AN INTEGER (OCTAL)
;
; R4 CONTAINS ADDRESS OF INTERGER
; REG 1,4 SCRATCHED
;
;
INTOUT: DEC R4 ;RESET INTEGER FLAG BIT
MOV (R4),R1 ;LOAD INTEGER INTO R1
INTOU2:
MOV R1,-(SP)
MOV R2,-(SP)
MOV R3,-(SP)
MOV R4,-(SP)
CMP -(SP),-(SP) ;OPEN 8 BYTE CELLS
CMP -(SP),-(SP)
MOV SP,R3
MOV SP,R4 ;LOAD CELL POINTERS
MOVB 'I,(R3)+ ;INTEGER IDENTIFER
MOV R0,-(SP) ;SAVE DEVICE POINTER
CLR R0 ;CLEAR OUTPUT BUFFER
MOV 1,-(SP)
ASHC (SP)+,R0 ;ROTATE MOST SIGNIFICANT BIT
ADD ASCII,R0 ;CONVERT IT TO ASCII
MOVB R0,(R3)+ ;LOAD BYTE
MOV 5,R2 ;SET UP COUNTER REGISTER
INTOU1: CLR R0 ;CLEAR OUTPUT BUFFER
MOV 3,-(SP)
ASHC (SP)+,R0 ;ROTATE NEXT (SP)+ BITS INTO BUFFER
ADD ASCII,R0 ;CONVERT IT TO ASCII
MOVB R0,(R3)+ ;LOAD BYTE
SOB R2,INTOU1 ;LOOP UNTIL FINISHED
DEC R3 ;SET POINTER
MOV (SP)+,R0 ;RELOAD POINTER
JSR R5,TYPE ;SEND THE STRING
CMP (SP)+,(SP)+ ;POP BYTE CELLS
CMP (SP)+,(SP)+
MOV (SP)+,R4 ;RESTORE THE REGISTERS
MOV (SP)+,R3
MOV (SP)+,R2
MOV (SP)+,R1
RTS PC ;RETURN
.PAGE

```

```

;
; /////////////// VALUE1 ///////////////
;
; THIS ROUTINE TYPES OUT TO THE DEVICE POINTED TO BY R0
; THE FLOATING POINT NUMBER POINTED TO BY R4
;
; REG 1,2,3,4 ARE SCRATCHED
;
;
VALUE1: MOV R0,-(SP) ;SAVE TABLE POINTER
        BIT INTVAR,R4 ;IS ADDRESS AN INTEGER ?
        BEQ VALAE2 ;BRANCH IF NO
;
        JSR PC,INTOUT ;TYPE OUT INTERGER
        MOV (SP)+,R0 ;RESTORE TABLE POINTER
        RTS PC ; RETURN
;
VALAE2: MOV R0,-(SP) ;ONCE FOR POINTER TO MESSAGE
        ADD TPOINT,(SP) ;MOVE PAST FLAG WORD
        MOV 12.,-(SP) ;PUSH 12. ONTO THE STACK
        MOV 4,-(SP) ;PUSH 4 ONTO THE STACK
        MOV 0,-(SP) ;PUSH 0 ONTO THE STACK
        MOV 2(R4),-(SP) ;PUSH 2(R4) ONTO THE STACK
        MOV (R4),-(SP) ;PUSH (R4) ONTO THE STACK
;
;THIS ROUTINE TAKES AN INTERNAL NUMBER AND
;CONVERTS IT TO AN ASCII CHARACTER STRING FOR
;OUTPUT
;
;
; ASCOUT
;
;
; $ECO THE E CONVERSION OUTPUT ROUTINE FOR REALS
; $FCO THE F CONVERSION OUTPUT ROUTINE FOR REALS
; $GCO THE G CONVERSION OUTPUT ROUTINE FOR REALS
; $DCO THE D CONVERSION ROUTINE FOR DOUBLES
; CALLING SEQUENCE :
; PUSH FIELD START
; PUSH FIELD LENGTH
; PUSH D PART OD W.D SPECIFICATION
; PUSH P SCALE
; PUSH VALUE TO BE OUTPUT
; JSR PC,$ECO (OR $FCO) (OR $GCO) (OR $DCO)
; R0, R1, R2, R3 ARE DESTROYED
POINT=2
BEXP=4
EEXP=6
TYPE$=12.
P=16.
D=18.
L=20.

```

```

S=22.
.PAGE
JSR PC,$FCO
JMP ENRCO$
$GCO: MOV 42403,R0 ;FLAG G FORMAT
BR XCO$9
$FCO: CLR R0 ;FLAG F FORMAT
BR XCO$9
$DCO: MOV (SP)+,R0 ;POP RETURN
MOV (SP)+,R1 ;GET HIGHEST ORDER ARG
MOV (SP)+,R2 ;GET NEXT
MOV @SP,R3 ;THIRD ARG WORD
MOV 42002,@SP ;FLAG D FORMAT
MOV R4,-(SP) ;SAVE R4
MOV 4(SP),R4 ;GET LOWEST ORDER ARG
MOV R0,4(SP) ;SAVE RETURN
BR XC1$9
$ECO: MOV 42402,R0 ;FLAG E FORMAT
XCO$9: MOV (SP)+,R3 ;SAVE RETURN
MOV (SP)+,R1 ;GET HIGH ORDER ARG
MOV (SP)+,R2 ;GET LOW ORDER ARG
MOV R3,-(SP) ;PUSH RETURN
MOV R0,-(SP) ;PUSH TYPE
CLR R3 ;CLEAR LOW ORDER REGISTERS
MOV R4,-(SP) ;SAVE R4
CLR R4
XC1$9: MOV R5,-(SP) ;SAVE R5 AND CONTINUE ALL TYPES
CLR -(SP) ;CLEAR EXP
CLR -(SP) ;CLEAR BEXP
CMP -(SP),-(SP) ;ROOM FOR POINT AND SIGN
ADD S(SP),L(SP) ;POINT ONE BEYOND END OF FAELD
MOV S(SP),R0
CLE$9: MOVB ' ',(R0)+ ;BLANK OUT END OF FIELD
CMP R0,L(SP)
BLO CLE$9
ROL R1 ;GET ARG SIGN
ROL @SP ;SAVE IT
SWAB R1
MOVB R1,BEXP(SP) ;GET BINARY EXP
BNE NNZ$9 ;JUMP IF ARG NOT 0
CLR R0 ;CLEAR OVERFLO ACCUMULATOR
BR NOD$9 ;PRINT THE 0 IN FORMAT
NNZ$9: SEC ;INSERT NORMAL BIT
ROR R1
CLRB R1 ;LEFT JUSTIFY FRACTION
SWAB R2
BISB R2,R1
CLRB R2
SWAB R3
BISB R3,R2
CLRB R3

```

```

SWAB   R4
BISB   R4,R3
CLRB   R4
SUB     200,BEXP(SP) ;REMOVE EXCESS 128 FROM BINARY
BLT     DIV$9        ;JMP IF BINARY EXPONENT NEG
BEQ     NOM$9        ;JUMP IF NO SCALING TO DO
MUL$9:  TST   R1      ;BINARY EXPONENT IS POSITIVE
BLT     ML1$9        ;JUMP IF FRACTION OVERFLO IMPENDING
ASL     R4           ;DOUBLE FRACTION
ROL     R3
ROL     R2
ROL     R1
DEC     BEXP(SP)     ;COMPENSATE EXPONENT
BGT     MUL$9        ;JUMP IF MORE BINARY SCALING TO DO
BR      NOM$9
ML1$9:  JSR   PC,M45$9 ;GET 4/5 FRACTION
INC     EEXP(SP)     ;MULTIPLY BY 10
SUB     3,BEXP(SP)   ;AND DIVIDE BY 8
BGT     MUL$9        ;JUMP IF BINARY EXPONENT IS STILL POS
BEQ     NOM$9        ;JUMP IF EXPONENT GONE NOW
DIV$9:  CMP   R1, 146314 ;BINARY EXP IS NEGATIVE
BHIS    DV1$9        ;JUMP IF NO ROOM FOR 4/5 FRACTION
CMP     BEXP(SP), -3
BGT     DV1$9        ;JUMP IF NOT ENOUGH BINARY EXP LEFT
JSR     PC,M54$9     ;MULTIPLY FRACTION BY 5/4
DEC     EEXP(SP)     ;DIVIDE BY 10
ADD     2,BEXP(SP)   ;MULTIPLY BY 4
BR      DV2$9
DV1$9:  JSR   PC,RIT$9 ;DIVIDE BY 2
DV2$9:  INC   BEXP(SP) ;MULTIPLY BY 2
BNE     DIV$9        ;HIT IT AGAIN IF BIN EXP NOT GONE
; AT THIS POINT THE BINARY EXPONENT IS 0
; AND THE FRACTION IS IN R1,R2,R3, AND R4
NOM$9:  CLR   R0      ;CLEAR OVERFLO ACCUMULATOR
NO1$9:  JSR   PC,M54$9 ;MULTIPLY FRACTION BY 5/4
JSR     PC,ML8$9     ;AND NOW BY 8
TST     R0
BNE     NOD$9        ;JUMP IF AN INTEGER PART RESULTS
DEC     EEXP(SP)     ;DECREMENT EXPONENT
BR      NO1$9        ;GO AGAIN TO GET AN INTEGER PART
; AT THIS POINT THE MOST SIGNIFICANT NON ZERO DIGIT IS IN
NOD$9:  TSTB  TYPE$(SP) ;TEST CONVERSION TYPE$
BEQ     FFT$9        ;JUMP IF F FORMAT
RORB    TYPE$(SP)
BCC     EFT$9        ;JUMP IF E OR D FORMAT
TST     EEXP(SP)     ;GFORMAT
BLT     EFT$9        ;JUMP IF RESULT LESS THAN .1
CMP     EEXP(SP),D(SP)
BGT     EFT$9        ;JUMP IF RESULT > 10**D
CLRB    TYPE$(SP)    ;MAKE IT TYPE$ F INSTEAD OF G
SUB     4,L(SP)      ;LEAVE ROOM FOR BLANKS ON RIGHT

```

```

SUB  EEXP(SP),D(SP) ;DECREASE D BY NUMBER OF DIGITS LEFT
CLR  P(SP)          ;SUSPEND P SCALE
FFT$9: MOV  EEXP(SP),R5 ;F FORMAT
FFE$9: ADD  D(SP),R5
      ADD  P(SP),R5
      JSR  PC,RUD$9 ;ROUND BY ADDING 5*10**-P-D-E
      MOV  L(SP),R5
      SUB  D(SP),R5
      TSTB TYPE$(SP)
      BNE  FF5$9 ;JUMP IF NOT P CONVERSION
      ADD  EEXP(SP),P(SP) ;COMBINE P AND EXPONENT
      BLE  FF5$9 ;JUMP IF THERE IS NO INTEGER PART IN RES
      SUB  P(SP),R5
      SUB  2,R5 ;SIGN SLOT IS S+L-D-E-P-2
      JSR  PC,ISN$9 ;INSERT SIGN AND CHECK WIDTH
      BR   FF3$9 ;JUMP TO INSERT IDIGITS
FF5$9: SUB  3,R5 ;SIGN SLOT IS S+L-D-3
      JSR  PC,ISN$9 ;GO INSERT SIGN AND CHECK WIDTH
      MOVB '0,(R5)+ ;INSERT LEADING 0
      MOVB '.,(R5)+ ;INSERT .
FF4$9: CMP  R5,L(SP) ;CHECK FIELD END
      BHIS FF3$9 ;JUMP IF FIELD IS FULL
      MOVB '0,(R5)+ ;PUT IN ANOTHER LEADING ZERO
      BR   FF4$9
FF3$9: MOV  L(SP),R5
      SUB  D(SP),R5
      DEC  R5 ;LOCATION FOR .
      MOV  R5,POINT(SP) ;REMEMBER ITS LOCATION
      TST  P(SP)
      BGT  FF6$9
      INC  R5 ;POINT TO SLOT FOR FIRST NON-ZERO DIGIT
FF6$9: SUB  P(SP),R5
      JSR  PC,DGS$9 ;GO NSERT ALL DIGITS
      TSTB TYPE$(SP)
      BEQ  DNE$9 ;ALL THROUGH IF F FORMAT
      BR   EFE$9 ;GO FINISH E FORMAT
EFT$9: SUB  4,L(SP) ;MAKE ROOM FOR E FIELD
      CLR  R5
      TST  P(SP)
      BLE  FFE$9 ;PROCESS AS F FMT & RETURN EFMTE
      MOV  D(SP),R5 ;GET ROUNDING FACTOR
      ADD  P(SP),R5 ;ALLOW FOR P SCALE
      JSR  PC,RUD$9 ;GO USE IT
      MOV  L(SP),R5 ;POINT TO SIGN SLOT
      SUB  D(SP),R5
      DEC  R5 ;POIN TO SLOT = L-D-1
      MOV  R5,POINT(SP) ;SAVE LOCATION FOR .
      SUB  P(SP),R5
      DEC  R5 ;SIGN SLOT = L-D-P-2
      JSR  PC,ISN$9 ;GO CHECK WIDTH AND INSERT SIGN
      JSR  PC,DGS$9 ;GO PROCESS ALL DIGITS

```

```

EFE$9:  SUB  P(SP),EEXP(SP)  ;CORRECT EXPONENT FOR P
        MOV  L(SP),R3
        MOVB TYPE$+1(SP),(R3)+ ;MOVE OUT E OR D
        MOV  EEXP(SP),R4
        BGE  EXP$9           ;JUP IF EXPGNENT POS
        NEG  R4              ;GET ABS VALUE
        MOVB '-,(R3)+       ;INSERT -
        BR   EX1$9
EXP$9:  MOVB  ' ,(R3)+       ;INSERT BLANK FOR +
EX1$9:  MOVB  '0,@R3         ;CLEAR TENS DIGIT
EX3$9:  SUB   10.,R4         ;TEST FOR TENS
        BLT  EX2$9
        INCB @R3             ;ACCUMULATE TENS
        BR   EX3$9
EX2$9:  ADD   72,R4          ;GET POSITIVE UNITS
        MOVB R4,1(R3)       ;MOVE UNITS OUT
DNE$9:  ADD   8.,SP
        MOV  (SP)+,R5
        MOV  (SP)+,R4
        MOV  (SP)+,6(SP)    ;MOVE FLAG AND RETURN UP
        MOV  (SP)+,6(SP)
        CMP  (SP)+,(SP)+    ;FLUSH JUNK
        ROL  (SP)+         ;SET C BIT IF ERROR
        RTS  PC             ;RETURN TO CALLER
; MULTIPLY CONTENTS OF R1 ... R4 BY 5/4
; ANY OVERFLO GOES TO R0
M54$9:  MOV  R1,-(SP)       ;5/4X=X+X/4
        MOV  R2,-(SP)
        MOV  R3,-(SP)
        MOV  R4,-(SP)
        JSR  PC,RIT$9       ;X/2
        JSR  PC,RIT$9       ;X/4
        ADC  R4             ;ROUND
        ADC  R3
        ADC  R2
        ADC  R1
        ADD  (SP)+,R4
        ADC  R3
        ADC  R2
        ADC  R1
        ADC  R0
        ADD  (SP)+,R3
        ADC  R2
        ADC  R1
        ADC  R0
        ADD  (SP)+,R2
        ADC  R1
        ADC  R0
        ADD  (SP)+,R1
        ADC  R0
        RTS  PC             ;RETURN TO CALLER

```

```

M45$9:  MOV    16.,R5      ;MULT R1 ... R4 BY 4/5
        JSR    PC,RIT$9
        MOV    R4,-(SP)
        MOV    R3,-(SP)
        MOV    R2,-(SP)
        MOV    R1,-(SP)
M51$9:  JSR    PC,RIT$9
        JSR    PC,RIT$9
        MOV    2,R0
M52$9:  JSR    PC,RIT$9
        ADD    6(SP),R4
        ADC    R3
        ADC    R2
        ADC    R1
        ADD    4(SP),R3
        ADC    R2
        ADC    R1
        ADD    2(SP),R2
        ADC    R1
        ADD    @SP,R1
        DEC    R0
        BGT    M52$9
        DEC    R5
        BGT    M51$9
        ADD    8.,SP      ;FLUSH MULTIPLIER
        RTS    PC
;
; MULTIPLY CONTENTS OF R0 ... R4 BY 8.
; NO OVERFLO IS ANTICIPATED
ML8$9:  MOV    R5,-(SP)
        MOV    3,R5
M81$9:  ASL    R4
        ROL    R3
        ROL    R2
        ROL    R1
        ROL    R0
        DEC    R5
        BGT    M81$9
        MOV    (SP)+,R5
        RTS    PC
ERR$9:  TST    (SP)+      ;POP RETURN
        MOV    S(SP),R3   ;POINT TO FIELD BEGIN
        MOV    L(SP),R4   ;GET FIELD END + 1
        TSTB   TYPE$(SP)  ;CHECK IF FIELD END MODIFIED
        BEQ    STS$9      ;NO, THIS IS P FORMAT
        ADD    4,R4       ;PUT BACK EXPONENT SPACE
STS$9:  MOVB    '*,(R3)+  ;FIL FIELD WITH *****
        CMP    R3,R4
        BLO    STS$9      ;JUMP IF MORE TO GO
        COM    TYPE$(SP)  ;FLAG IF ERROR
        BR     DNE$9

```

```

;
; ROUND THE CONTENTS OF R0 ... R4 TO THE PRECISION
; SPECIFIED BY R5.
; THIS ROUTINE IS SHORTER THAN THE TABLE THAT
; OTHERWISE WOULD BE NEEDED
RUD$9:  CMP  R5, 20.
        BGT  RU1$9      ;JUMP IF NOT WORTH ROUNDING
        MOV  R5,BEXP+0+2(SP)      ;SAVE ROUNDING PRECISION IN TEMP
        BEQ  RU3$9      ;JUMP IF ROUND IS TO LEADING DIGIT
        BLT  RU1$9      ;JUMP IF NO ROUNDING TO DO
        MOV  R0,-(SP)
        MOV  R1,-(SP)
        MOV  R2,-(SP)
        MOV  R3,-(SP)
        MOV  R4,-(SP)
        MOV  100000,R1  ;INSERT .5
        CLR  R2
        CLR  R3
        CLR  R4
RDF$9:  DEC  BEXP+0+2+10.(SP) ;COUNT PRECISION
        BEQ  RDD$9      ;JUMP IF DONE
        JSR  PC,M45$9    ;MULTIPLY BY 4/5
        JSR  PC,RIT$9
        JSR  PC,RIT$9
        JSR  PC,RIT$9    ;DIVIDE BY 8
        BR   RDF$9      ;GO CHECK IF DONE WITH FACTOR
RDD$9:  CLR  R0
        ADD  (SP)+,R4    ;ADD FACTOR TO RND FACTOR
        ADC  R3
        ADC  R2
        ADC  R1
        ADD  (SP)+,R3
        ADC  R2
        ADC  R1
        ADD  (SP)+,R2
        ADC  R1
        ADD  (SP)+,R1
        ADC  R0
        ADD  (SP)+,R0
RU2$9:  CMP  10.,R0
        BGT  RU1$9      ;JUMP IF NO OVERFLO
        INC  EEXP+2(SP) ;BUMP DECIMAL EXP
RU1$9:  RTS  PC          ;RETURN TO CALLER
RU3$9:  ADD  5,R0        ;ROUND TO MOST SIGNIFICANT DIGIT
        BR   RU2$9
;
; INSERT A - IF NECESSARY AND CHECK THAT THE FIELD
; IS WIDE ENOUGH TO CONTAIN THE RESULT.
ISN$9:  CMP  R5,S-0+2(SP) ;COMPARE SIGN SLOT WITH FIELD BE
        BLO  SPC$9      ;JUMP IF IT MAY NOT FIT
        ROR  0+2(SP)    ;TEST SIGN

```

```

    BCC ISR$9      ; JUMP IF +
    MOVB '-,@R5    ; INSRT -
ISR$9: INC R5      ; POINT TO LEADING DIGIT SLOT
    RTS PC        ; RETURN
SPC$9: ROR 0+2(SP) ; TEST SIGN
    BCS ERR$9     ; JUMP IF IT7S - 7CAUSE THER AINT ROOM
    INC R5        ; POINT TO LEADING DIGIT SLOT
    CMP R5,S+2(SP)
    BLO ERR$9     ; JUMP IF NO ROOM FOR IT EITHER
    RTS PC

;
; EXTRACT LEADING DIGITS FROM R0 ... R4 AND FILL IN
; THE AREA STARTING AT THE ADDRESS IN R5 AND
; BOUNDED BY THE MODIFIED FIELD END.
DGS$9: CMP 10.,R0 ; CHECK IF OVERFLO IN R0
    BGT DG1$9     ; JUMP IF ONLY ONE DIGITS WORTH
    MOVB '1,(R5)+ ; OUTPUT OVERFLO
    SUB 10.,R0    ; CORRECT R0 FOR NEXT DIGIT
DG1$9: CMP POINT+2(SP),R5 ; CHECK FOR . SLOT
    BNE DG2$9
    MOVB '.,(R5)+ ; INSERT THE .
DG2$9: CMP L+2(SP),R5 ; CHECK END OF FIELD
    BLOS DIG$9    ; JUMP IF DONE
DG3$9: ADD 60,R0   ; CONVERT TO ASCII
    MOVB R0,(R5)+ ; PUT IT IN FIELD
    CLR R0
    JSR PC,M54$9   ; MULTIPLY FRACTION BY 5/4
    JSR PC,ML8$9   ; AND BY 8
    BR DG1$9      ; GO CONVERT TO ASCII
DIG$9: RTS PC     ; RETURN TO CALLER
;
; SHIFT THE CONTENTS OF R1 ... R4 RIGHT 1.
RIT$9: CLC
    ROR R1
    ROR R2
    ROR R3
    ROR R4
    RTS PC
ENRCO$: NOP ; NO OPERATION
    MOV (SP)+,R0 ; RESTORE POINTER
    MOV R0,R3    ; SET POINTER
    ADD TPOINT+11.,R3 ; POINT TO END OF STRING
    MOV R0,R4    ; LOAD ASCII ADDRESS
    ADD TPOINT,R4 ; POINT TO START OF STRING
    JSR R5,TYPE  ; TYPE IT OUT
    RTS PC      ; RETURN
.PAGE

```

```

;
;  ///////////      REAL  NUMBER      ///////////
;
;  THIS ROUTINE TAKES A STRING OF ASCII CHARACTERS FROM
;  THE INPUT BUFFER AND CONVERTS THEM TO INTERNAL FLOATING
;  POINT REPRESENTATION.
;
;  ON ENTRY, REG 0 MUST POINT TO BUFFER
;  ON EXIT,  REG 2,4 CONTAIN FLOATING POINT
;  REG 1 SCRATCHED
;
ERR2:  JMP  ERR      ;NON-NUMERIC ENTRY
RELNO:
  MOV  TPOINT(R0),-(SP)      ;PUSH TPOINT(R0) ONTO THE STACK
  CLR  -(SP)      ;INITIALIZE BYTE COUNT
  MOV  0,-(SP)      ;PUSH 0 ONTO THE STACK
  MOV  0,-(SP)      ;PUSH 0 ONTO THE STACK
RELNO1: JSR  R5,GET      ;GET A CHAR
  CMPB CR,R1      ;IS IT A FIELD TERMINATOR ?
  BEQ  RELNO2      ;BRANCH IF IT IS.
  CMPB '9',R1      ;IS THE R1 ABOVE 9 ?
  BMI  ERR2      ; BRANCH IF IT IS
  CMPB ',',R1      ;IS THE R1 BELOW 0 ?
  BPL  ERR2      ; BRANCH IF IT IS
RELNO3: INC  4(SP)      ;INCREMENT COUNT IF IT ISN'T
  BR  RELNO1      ; AND LOOP UP AGAIN .
RELNO2:
;
;  /////      ASCII TO FP  ///////////
;
;  $DCI --- ASCII TO DOUBLE CONVERSION.
;  $RCI --- ASCII TO REAL CONVERSION
;  CALLING SEQUENCE:
;  PUSH ADDRESS OF START OF FIELD
;  PUSH LENGTH OF FIELD
;  PUSH FORMAT SCALE D FROM W.D
;  PUSH P FORMAT SCALE
;  JSR  PC,$DCI (OR $RCI)
NUMEND=0
POINTL=2
DIGITS=4
BEXP=6
ESIGN=8.
SIGN=10.
EEXP=12.
P=30.
D=32.
ERF=26.
LENGTH=34.
TEMP=LENGTH
RESULT=P

```

```

START=36.
END=START
.PAGE
    JSR    PC,$RCI        ;SET UP FORMAT
    JMP    ENRCI$         ;RETURN TO MAIN
$RCI:    CLR    -(SP)      ;CLEAR ERROR FLAG
    INC     @SP
    BR     CNV$8
$DCI:    CLR    -(SP)
CNV$8:    MOV    R0,-(SP)   ;SAVE NUMBER
    MOV    R1,-(SP)
    MOV    R2,-(SP)
    MOV    R3,-(SP)
    MOV    R4,-(SP)
    MOV    R5,-(SP)
    CLR    -(SP)          ;CLEAR EXPONENT
    CLR    -(SP)          ;CLEAR SIGN
    CLR    -(SP)          ;CLEAR EXP SIGN
    MOV    65.,-(SP)      ;INITIALIZE BEXP
    MOV    18.,-(SP)      ;INITIALIZE MAX DIGITS
    CLR    -(SP)          ;CLEAR POINT L
    CLR    -(SP)          ;CLEAR NUM END
    MOV    START(SP),R5    ;GET START FIELD ADDRESS
    ADD    LENGTH(SP),END(SP) ;POINT TO END PLUS ONE
    CLR    R0              ;CLEAR
    CLR    R1              ; NUMERIC
    CLR    R2              ; WORK
    CLR    R3              ; SPACE
SCN$8:    MOVB   (R5)+,R4   ;GET NEXT INPUT CHARACTER
    BIC    177600,R4       ;STRIP PARITY
    CMPB   R4,' '          ;TEST FOR BLANK
    BNE    SGS$8           ;IF NOT BLANK, LOOK FOR + OR -
    CMP    R5,START(SP)    ;CHECK FOR END OF FIELD
    BLT    SCN$8           ;IF NOT DONE, GO GET 'UM ANOTHER ONE
    JMP    ZER$8           ;ENTIRE FIELD IS BLANK !
SGS$8:    CMPB   R4,'+'     ;CHECK FOR + SIGN
    BEQ    FLD$8           ;IF FOUND, IGNORE IT
    CMPB   R4,'-'         ;CHECK FOR - SIGN
    BNE    NCK$8           ;IF NOT FOUND, CHECK NUMERICS
    INC    SIGN(SP)        ;SET - SIGN FLAG
    BR     FLD$8           ;
NXT$8:    MOVB   (R5)+,R4   ;GET NEXT INPUT CHARACTER
    BIC    177600,R4       ;
    CMPB   R4,' '          ;CHECK FOR BLANKS
    BNE    NCK$8
    MOV    '0,R4           ;TREAT BLANKS AS ZEROS
NCK$8:    CMPB   R4,'0      ;CHECK FOR LEGAL CHARACTERS
    BLT    PCK$8           ;CHECK FOR DECIMAL POINT
    BNE    NNZ$8           ;JMP IF NOT ZERO
    TST    R0              ;CHECK TO SEE IF ANY NON-ZERO DIGITS FOUND
    BNE    NNZ$8

```

```

TST R1
BNE NNZ$8
TST R2
BNE NNZ$8
TST R3
BEQ FLD$8
NNZ$8:  CMPB R4, '9
      BGT EXC$8 ;CHECK FOR EXPONENT
      DEC DIGITS(SP) ;COUNT IT AS A SIGNIFICANT DIGIT
      BGE A2I$8 ;JUMP IF WE CAN USE THIS DIGIT
      INC EEXP(SP) ;COMPENSATE FOR SKIPPED DIGIT
      BR FLD$8
A2I$8:  SUB 60,R4 ;CONVERT ASCII TO INTEGER
      JSR PC,ML5$8 ;MULTIPLY BY 5
      JSR PC,LFT$8 ;DOUBLE RESULT FOR TEN
      ADD R4,R3 ;ADD IN CURRENT DIGIT
      ADC R2
      ADC R1
      ADC R0 ;END CONVERT FOR THIS DIGIT
FLD$8:  CMP R5,END(SP) ;CHECK FOR END OF FIELD
      BLT NXT$8 ;POINTER TO LAST NUMERIC TO NUMEREND
      MOV R5,@SP
SCL$8:  TST R0
      BNE SC1$8 ;JUMP IF NUMBER IS 0
      TST R1
      BNE SC1$8
      TST R2
      BNE SC1$8
      TST R3
      BEQ ZER$8 ;INPUT NUMBER IS ZERO
SC1$8:  CMP @SP,R5 ;CHECK NUMEND
      BNE NOP$8 ;JUMP IF THER WAS AN EXPONENT FIELD
      SUB P(SP),EEXP(SP) ;USE THE FORMAT P SCALE
NOP$8:  TST POINTL(SP)
      BNE PNT$8 ;JUMP IF THERE WAS A DECIMAL POINT
      MOV D(SP),@SP ;USE THE D SCALE
PNT$8:  SUB POINTL(SP),@SP
      SUB @SP,EEXP(SP) ;FORM THE COMPLETE DECIMAL EXPONENT
      BGT MUL$8 ;MULTIPLY BY 10**EXPONENT
      BLT DIV$8 ;JUMP IF DECIMAL EXPONENT IS NEGATIVE
      JMP FLT$8 ;JUMP IF EXPONENT IS ZERO
MUL$8:  CMP R0, 31462
      BHI MDV$8 ;JUMP IF FRACTION IS TOO LARGE TO MULT BY 5
      JSR PC,ML5$8 ;FRACT = 5 * FRACT
      INC BEXP(SP) ;TIMES TWO
D10$8:  DEC EEXP(SP) ;OVER TEN
      BGT MUL$8 ;JUMP IF MORE DECIMAL EXPONENT
      JMP FLT$8 ;DECIMAL EXPONENT GONE
MDV$8:  JSR PC,M54$8 ;MULTIPLY BY 5/4
      ADD 3,BEXP(SP) ;TIMES EIGHT
      BR D10$8 ;GO DIVIDE IT BY 10

```

```

PCK$8:  CMPB  R4, '.'
        BNE  ERR$8      ;JUMP IF NOT A DECIMAL POINT
PTF$8:  TST  POINTL(SP)
        BNE  ERR$8      ;JUMP IF A "." ALREADY ENCOUNTERED
        MOV  R5,POINTL(SP) ;SAVE A POINTER TO THE . +1
        BR   FLD$8      ;GO FOR NEXT CHARACTER
ERR$8:  COMB  ERF+1(SP)  ;FLAG ERROR
ZER$8:  CLR  R0          ;RESULT IS ZERO
        CLR  R1
        CLR  R2
        CLR  R3
        JMP  STR$8      ;GO PUSH RESULT AND RETURN
EXC$8:  CMPB  R4, 'E'
        BEQ  EXT$8      ;JUMP IF E
        CMPB  R4, 'D'
        BNE  ERR$8      ;IF NOT E OR D THEN ERROR
EXT$8:  MOV  R5,@SP      ;SAVE POINTER TO END OF NUM +1
        DEC  @SP        ;DECREMENT NUMEND
        MOV  R3,TEMP(SP)
        CLR  R3
        CMP  R5,END(SP)
        BGE  ERR$8      ;JUMP IF NO ROOM FOR EXPONENT
        MOVB (R5)+,R4
        BIC  177600,R4
        CMPB  R4, '+'    ;CHECK FOR + EXPONENT
        BEQ  EF1$8
        CMPB  R4, '-'    ;CHEK FOR - EXPONENT
        BNE  ENM$8      ;GO CHECK FOR NUMERIC
        INC  ESIGN(SP)   ;FLAG EXPONENT NEGATIVE
EF1$8:  CMP  R5,END(SP)
        BGE  ERR$8
EF2$8:  MOVB (R5)+,R4    ;GET NEXT CHARACTER
        BIC  177600,R4
ENM$8:  CMPB  R4, ' '    ;CHECK FOR BLANK
        BNE  EN1$8
        MOV  '0,R4      ;TREAT BLANK AS ZERO
EN1$8:  CMPB  R4, '0'
        BLT  ERR$8
        CMPB  R4, '9'
        BGT  ERR$8      ;NOT A VALID CHARACTER
        SUB  60,R4      ;CONVER ASCII TO INTEGER
        ASL  R3          ;X=10*X+D
        ADD  R3,R4
        ASL  R3
        ASL  R3
        ADD  R4,R3      ;END OF ABOVE COMMENT
        CMP  R5,END(SP)
        BLT  EF2$8      ;JUMP IF MORE FIELD TO GO
        TST  ESIGN(SP)   ;CHECK EXPONENT SIGN
        BEQ  EN2$8      ;JUMP IF IT IS +
        NEG  R3          ;MAKE USER EXPONENT -

```

```

EN2$8:  ADD  R3,EEXP(SP)  ;GET COMPLETE EXPONENT
        MOV  TEMP(SP),R3
        JMP  SCL$8      ;GO SCALE THE NUMERIC PART
DIV$8:  TST  R0
        BLT  DV1$8      ;JUMP IF FRACTION LEFT JUSTIFIED
DV2$8:  DEC  BEXP(SP)    ;LEFT JUSTIFIED NUMERIC BITS
        JSR  PC,LFT$8
        BPL  DV2$8
DV1$8:  MOV  16.,R4      ;SET FOR SIXTEEN ITERATIONS
        JSR  PC,RIT$8
        MOV  R3,-(SP)
        MOV  R2,-(SP)
        MOV  R1,-(SP)   ;INITIALIZE QUOTIENT
        MOV  R0,-(SP)
DV3$8:  JSR  PC,RIT$8
        CLC
        JSR  PC,RIT$8
        MOV  2,R5
        CLC
DV4$8:  JSR  PC,RIT$8
        ADD  6(SP),R3
        ADC  R2
        ADC  R1
        ADC  R0
        ADD  4(SP),R2
        ADC  R1
        ADC  R0
        ADD  2(SP),R1
        ADC  R0
        ADD  @SP,R0
        DEC  R5          ;COUNT TWICE
        BGT  DV4$8
        DEC  R4
        BGT  DV3$8
        ADD  8.,SP      ;POP DIVIDEND
        SUB  3,BEXP(SP)
        INC  EEXP(SP)   ;BUMP DECIMAL EXPONENT
        BLT  DIV$8      ;JUMP IF MORE TO DO
FLT$8:  DEC  BEXP(SP)   ;POST NORMALIZE THE RESULT
        JSR  PC,LFT$8
        BCC  FLT$8
        ADD  200,BEXP(SP) ;SET EXCESS 128
        BLE  UND$8      ;NUMBER TOO SMALL TO REPRESENT
        CMP  BEXP(SP), 377
        BGT  OVR$8      ;JUMP IF NUMBER TOO BIG
        CLRB R3
        BISB R2,R3
        SWAB R3
        CLRB R2
        BISB R1,R2
        SWAB R2

```

```

CLRB R1
BISB RO,R1 ;MOVE OUT LOWEST ORDER BITS
SWAB R1
CLRB RO
BISB BEXP(SP),RO ;INSRT THE BINARY EXPONENT
SWAB RO ;PUT IN RIGHT ORDER
ROR SIGN(SP) ;TEST THE ARITHMETIC SIGN
JSR PC,RIT$8 ;INSERT THE RESULTS
ADC R3
ADC R2
ADC R1 ;FINAL ROUND
ADC RO
BVS OVR$8 ;JUMP IF OVERFLO
BCS OVR$8
STR$8: TSTB ERF(SP) ;TEST REAL/DOUBLE FLAG
BEQ DPR$8 ;JUMP IF DOUBLE
ROL R2 ;ROUND TO REAL PRECISION
ADC R1
ADC RO
BVS OVR$8
BCS OVR$8 ;JUMP IF OVERFLO ON ROUND
MOV RO,R2 ;MOVE HIGH ORDER RESULT UP
MOV R1,R3
DPR$8: MOV RO,RESULT(SP) ;STORE RESULT ON STACK
MOV R1,RESULT+2(SP)
MOV R2,RESULT+4(SP)
MOV R3,RESULT+6(SP)
ADD 14.,SP ;CLEAR STACK OF JUNK
MOV (SP)+,R5
MOV (SP)+,R4
MOV (SP)+,R3
MOV (SP)+,R2
MOV (SP)+,R1
MOV (SP)+,RO
TSTB @SP ;TEST REAL/DOUBLE FLAG
BEQ RRN$8 ;JUMP IF DOUBLE
MOV (SP)+,2(SP) ;PUSH FLAG UP
MOV (SP)+,2(SP) ;PUSH RETURN FLAG UP
RRN$8: ROL (SP)+ ;FLUSH FLAG AND SET C BIT IF ERROR
RTS PC
OVR$8:
UND$8: JMP ERR$8
M54$8: CMP RO,146314
BLO M55$8 ;JUMP IF ROOM FOR 5/4 * FRACTION
CLC
JSR PC,RIT$8 ;DIVIDE BY TWO
INC BEXP+0+2(SP) ;MULTIPLY BY 2
M55$8: MOV RO,-(SP)
MOV R1,-(SP)
MOV R2,-(SP)
MOV R3,-(SP)

```



```
;
;
;//////////      COLD START      //////////
;
;  THIS IS A COLD START ROUTINE TO BRING THE
;  DIGITAL CONTROLLER ON-LINE FROM A COMPLETE DEAD
;  (ALL RAM MEMORY LOST) STATE.
;
;  M.MEALS  NOV.'76
;
```

```

;
;
;
CSTART:  MOV    200,R0
MTPS    R0      ;DISALLOW ANY EXTERNAL INTERRUPTS
RESET    ;CLEAR ALL PERIPHERAL DEVICES
;
MOV      MEMTO$,R0 ;LOAD POINTER
MOV      STAO$,R1  ;CLEAR MEMORY INCLUSIVE
CSTA1$:  CLR    (R1)+ ;CLEAR IT
CMP      R0,R1
BGT      CSTA1$
MOV      SKST$,SP  ;RELOAD STACK POINTER
;
JSR      PC,RLCOTB ;RELOAD CONTROL TABLE
;
JSR      PC,RLINCK ;RELOAD INTERRUPT CLOCKS
;
CLR      R0
MTPS    R0      ;ALLOW EXTERNAL INTERRUPTS
.PAGE

```

```

;
; THIS SECTION IS A LOOP THAT THE CPU IDLES IN
; WHEN NOT USED BY THE RTM OR OTHER INTERRUPT.
; IT CALCULATES THE CPU TIME LOADING
;
; REG 0 IS SCRATCHED
;
CSTA2$: MOV SYLD$+10,R0 ;LOAD POINTER
MOV FLT1+2,-(R0)
MOV FLT1,-(R0) ; INCREMENT COUNT
;
FADD R0
;
BIS 100,SRS1 ;ALLOW CONSOLE INTERRUPTS
BR CSTA2$
;
FLT1: .FLT2 1.00 ;FLOATING POINT WORD
;
;
SYLD$$: MOV CTMX+2,-(SP) ; LOAD MAXIMUM VALUE
MOV CTMX,-(SP)
MOV SYLD$+12,-(SP) ;SUBTRACT OFF COUNT
MOV SYLD$+10,-(SP)
FSUB SP
;
MOV PERMX+2,-(SP) ;SET UP FOR PER CENT NORMALIZED LOAD
MOV PERMX,-(SP)
FDIV SP
;
MOV SYLD$,-(SP) ;LOAD OUTPUT ADDRESS
MOV 10,-(SP) ;TAU = 10
JSR PC,FILT$$ ;FILTER IT
;
CLR SYLD$+12 ;CLEAR LOAD COUNT
CLR SYLD$+10
;
RTS PC
CTMX: .FLT2 9750. ; MAXIMUM COUNT FOR 60. TICKS
PERMX: .FLT2 97.5 ; ( MAX COUNT ) / 100 FOR %
.PAGE

```

```

;
; THIS ROUTINE LOADS RAM MEMORY WITH TABLE VALUES
; STORED IN COLD$$
;
; REG 3,4,5 SCRATCHED
;
RLCOTB:  MOV    COLD$$,R4  ;SET R4 TO TOP OF COLD START TABLE
;
RLCO1$:  MOV    (R4)+,R3   ;LOAD BLOCK ENDING ADDRESS
MOV      (R4)+,R5         ;LOAD STORAGE ADDRESS
CMP      R3,R5           ; AT END OF THE RUN ?
BEQ      RLCO3$          ;EXIT IF YES
;
RLCO2$:  CMP     R4,R3      ;AT THE END OF A BLOCK ?
BEQ      RLCO1$           ;BRANCH IF YES
MOV      (R4)+,(R5)+      ; OTHERWISE, MOVE ANOTHER WORD
BR       RLCO2$           ;AND LOOP
;
;
RLCO3$:  RTS     PC        ;RETURN
;
;
RLINCK:  MOV     CKTB$,R4  ;SET R4 TO TOP OF INTERRUPT TABLE
BR       RLCO1$           ; AND LOAD 'ER UP !
;
;
;////////// TRAPS //////////
.PAGE

```

```

;
;-----FILTR.MAC-----
; THIS PROGRAM FILTERS A SIGNAL WITH A FILTER CONSTANT
;   X=C1*X+C2*U
;   C1=EXP(-T/TAU)
;   C2=1-C1
;
; ALL REG SAVED
;
; TO ACCESS FILTER
; PUSH FILTER INPUT FLT2 LOW
;   "           "           "   FLT2 HI
;   "           "   OUTPUT ADDRESS
;   "           "   TAU
; JSR PC,FILT$$
;
;
;
FILT$$: MOV RO,-(SP)
        MOV 4(SP),RO ;GET THE ARGUEMENT
;
        CMP RO, 20 ;TIME CONSTANT OF 20?
        BEQ FIL20A ;BRANCH IF YES
;
        CMP RO, 10 ;TIME CONSTANT OF 10 ?
        BEQ FIL10A ;BRANCH IF YES
;
        CMP RO, 2 ;TIME CONSTANT OF 2 ?
        BEQ FILO2A ;BRANCH IF YES
;
FILRE$: MOV (SP)+,RO
        MOV (SP)+,6(SP) ;SHUFFEL STACK
        TST (SP)+
        CMP (SP)+,(SP)+
        RTS PC
;
FIL20A: JSR R5,FLTR
        .FLT2 .9512294245 ; C1 = EXP(-T/TAU)
        .FLT2 .0487705775 ; C2 = 1. - C1
        BR FILRE$
;
FIL10A: JSR R5,FLTR
        .FLT2 .904837418
        .FLT2 .0951625819
        BR FILRE$
;
FILO2A: JSR R5,FLTR
        .FLT2 .6065306597
        .FLT2 .3934693403
        BR FILRE$
;
.PAGE

```

```

; COMPUTE C1*X
FLTR: MOV 10(SP),R0 ;GET OUTPUT POINTER
      MOV 2(R0),-(SP)
      MOV (R0),-(SP)
      MOV 2(R5),-(SP)
      MOV (R5),-(SP)
      CMP (R5)+,(R5)+ ;POP CELL
      FMUL SP
; COMPUTE C2*U
      MOV 20(SP),-(SP) ;LOAD FILTER INPUT
      MOV 20(SP),-(SP)
      MOV 2(R5),-(SP)
      MOV (R5),-(SP)
      CMP (R5)+,(R5)+ ;POP CELL
      FMUL SP
; COMPUTE C1*X+C2*U
      FADD SP
      MOV (SP)+,(R0)+ ;STORE ANSWER
      MOV (SP)+,(R0)
      RTS R5 ;RETURN
      ;
      ;
.PAGE

```

```

;
;//////////      ERROR TRAP HANDLER      //////////
;
;  THIS ROUTINE IS A TRAP HANDLER FOR TRAPS RESULTING
;  FROM SYSTEM ERRORS
;
;  ALL REG SAVED
;  M.MEALS  NOV'76
;
;
ERTRA$:  RESET  ; CLEAR OUTPUT DEVICES
NOP      ;LEAVE ROOM FOR HALT FOR DE-BUGGING
JMP      CSTART
;
;
NUTR$:   CMP     12,2(SP)  ;F.P. UNDERFLO
BNE      ERTRA$      ;STD ERROR IF NOT
;
;
CLR      10(SP)      ;ZERO ERROR
CLR      12(SP)
BIC      12,2(SP)    ;CLEAR UNDERFLO ERROR
BIS      4,2(SP)     ;SET ZERO BIT
MOV      (SP)+,2(SP) ;FIX UP RETURN ADDRESS
MOV      (SP)+,2(SP)
RTI      ;RETURN FROM TRAP
.PAGE
.ENDM DATA$$

```

APPENDIX F

MACRO VALV\$\$

See note regarding Appendix C through M at start of Appendix C.

VALV\$\$ is a control routine used to position valves. VALV\$\$ operates on a control table containing the valve position, setpoint, and tuning parameters as well as other dynamic values associated with the controller.

The output of this controller is a signed integer indicating the number of ticks (1/60 sec) the increase or decrease relay should be held closed to move the valve to the setpoint.

The controller has two tuning parameters. The first tuning parameter is a proportional gain and produces a controller output directly proportional to the error. The second tuning parameter is a threshold. The controller output will be zeroed if it is less than the threshold to eliminate relay chatter.

The controller also has high and low limits parameters for constricting valve position and for higher level cascade controllers.

```

;
.MACRO VALV$$
;----- VALVE.MAC -----
; THIS PROGRAM IS THE VALVE CONTROL ALGORITHM
;REG 4 POINTS TO CONTROL TABLE
; ALL REG SAVED
;
;
;
; BEGINING OF PROGRAM
.PAGE
VALVE$: TST ST$(R4) ;TEST IF LOOP IS IN AUTO
BMI VALVA ;CONTINUE IF YES
RTS PC ;OTHERWISE, EXIT
;
; CHECK SET POINT FOR UPPER LIMITS
VALVA:
MOV R1,-(SP) ;SAVE A REGISTER
MOV HL$+2(R4),-(SP)
MOV HL$(R4),-(SP)
MOV $$$+2(R4),-(SP)
MOV $$$ (R4),-(SP)
FSUB SP
BGE VALV1A ;IF (HL-S) RESET,OK,OK
MOV HL$(R4),$$$ (R4)
MOV HL$+2(R4),$$$+2(R4)
;
; CHECK SET POINT FOR LOWER LIMITS
VALV1A: CMP (SP)+,(SP)+ ;POP CELLS
MOV LL$+2(R4),-(SP)
MOV LL$(R4),-(SP)
MOV $$$+2(R4),-(SP)
MOV $$$ (R4),-(SP)
FSUB SP
BLE VALV1B ;IF (LL-S) OK,OK, RESET
MOV LL$(R4),$$$ (R4)
MOV LL$+2(R4),$$$+2(R4)
; CHECK VALVE FOR UPPER LIMITS
VALV1B: CMP (SP)+,(SP)+ ;POP CELLS
MOV HL$+2(R4),-(SP) ;SET UP FOR FLOATING POINT
COMPARISON
MOV HL$(R4),-(SP)
MOV X$$$+2(R4),-(SP)
MOV X$$$ (R4),-(SP)
FSUB SP ;IF (HL-X)DOWN,DOWN,OK
BGT VALV1C ;OK
TST A$$$ (R4) ;POSITIVE ACTION?
BPL DOWN ;DOWN IF YES
BR UP ;OTHERWISE, UP
; CHECK VALVE FOR LOWER LIMIT

```

```

VALV1C:  CMP  (SP)+,(SP)+
        MOV  LL$+2(R4),-(SP) ;SET UP FOR FP1$1 COMPARISON
        MOV  LL$(R4),-(SP)
        MOV  X$$+2(R4),-(SP)
        MOV  X$$$(R4),-(SP)
        FSUB SP ;IF (X-LL)OK,UP,UP
        BMI  VALV1D ;BRANCH IF OK
        TST  A$$$(R4) ;POSITIVE ACTION
        BPL  UP ;UP IF YES
        BR   DOWN ;OTHERWISE, DOWN
.PAGE
;      SET UP ARGUYMENTS AND COMPUTE CONTROL ERROR
VALV1D:  CMP  (SP)+,(SP)+
        MOV  S$$+2(R4),-(SP)
        MOV  S$$$(R4),-(SP)
        MOV  X$$+2(R4),-(SP)
        MOV  X$$$(R4),-(SP)
        FSUB SP ; E = S - X
        TST  A$$$(R4) ;POSITIVE ACTION
        BGE  VALV1E ;BRANCH IF YES
        MOV  R0,-(SP) ;SAVE A REGISTER
        MOV  100000,R0 ;OTHERWISE, CHANGE SIGN OF CONTROLLER
        XOR  R0,2(SP)
        MOV  (SP)+,R0 ;RESTORE REGISTER
VALV1E:  MOV  (SP),SE$(R4) ;STORE ERROR
        MOV  2(SP),SE$+2(R4)
;
;      SET UP ARGUYMENTS AND COMPUTE CONTROL ACTION
        MOV  P1$+2(R4),-(SP)
        MOV  P1$(R4),-(SP)
        FMUL SP ; U = P * E
;
.PAGE
;//////// IFIX //////////
;
;      THIS ROUTINE CONVERTS A PDP 11 FLOATING POINT
;      NUMBER STORED AS TWO WORDS ON THS STACK INTO
;      A 16 BIT SIGNED INTEGER. THE ANSWER IS RETURNED
;      ON THE STACK.
;
;      STACK POINTER CAN BE ANY REGISTER
;
;      ALL REGISTERS ARE MAINTAINED
;
;      M.MEALS 8'/77
;
        MOV  R0,-(SP) ;SAVE USED REGISTERS
        MOV  R1,-(SP)
        MOV  R2,-(SP)
;
        MOV  6(SP),R2 ;GET FLOATING POINT (HIGH)

```

```

MOV  R2,R0  ;GET FRACTION (HIGH)
ASL  R2  ;SHIFT INTO TRUE BYTES
BIC  377,R2  ;SET UP EXPONENT
SWAB R2
SUB  200+24.,R2  ;SUBTRACT OFF EXCESS NOTATION
;
BIC  177400,R0  ;SET UP FRACTION (HIGH)
BIS  200,R0  ;SET UP HIDDEN BIT
MOV  10(SP),R1  ;GET FRACTION LOW
ASL  R1
ADC  R0
;
ASHC R2,R0
TST  R0  ;IS ANSWER MORE THAN 16 BITS?
BNE  IFIV$1  ;BRANCH IS YES
;
TST  R1  ;IS ANSWER MORE THAN 15 BITS ?
BGE  IFIV$2  ;BRANCH IF NO
;
IFIV$1:  MOV  77777,R1  ;SET MAXIMUM COUNT
;
IFIV$2:  TST  6(SP)  ;CHECK ORIGINAL SIGN
        BGE  IFIV$3  ;BRANCH IF POSITIVE
        NEG  R1  ;NEGATIVE ANSWER
;
IFIV$3:  MOV  R1,10(SP)  ;STORE ANSWER
        MOV  (SP)+,R2  ;RESTORE REGISTERS
        MOV  (SP)+,R1
        MOV  (SP)+,R0
        TST  (SP)+  ;POP EXTRA CELL
.PAGE
        MOV  (SP)+,R1  ;GET INTEGER
        BMI  VALVE2  ;BRANCH IF NEGATIVE
;
        JSR  PC,VALV4  ;CHECK LIMITS
        BR  VALVE3  ;EXIT
;
VALVE2:  NEG  R1  ;MAKE IT POSITIVE
        JSR  PC,VALV4  ;CHECK LIMITS
        BEQ  VALVE3  ;EXIT IF ZERO
        BIS  200,R1  ;MAKE IT LOOK NEGATIVE
        BR  VALVE3  ;EXIT
;
UP:  MOV  177,R1  ;LOAD MAXIMUM OPEN COUNT
     BR  DOWNA  ;EXIT
DOWN:  MOV  377,R1  ;LOAD MAXIMUN CLOSE COUNT
DOWNA:  CMP  (SP)+,(SP)+
;
VALVE3:  MOV  R1,U$(R4)  ; AND STORE IT AWAY
OUT1:  MOV  (SP)+,R1  ;RESTORE THE REGISTER
       RTS  PC  ;RETURN

```

```
;
VALV4:  CMP  R1, 177 ; COMPARE WITH MAXIMUM VALUE
        BLE  VALV4A ; IF (R1 - HL) OK,OK,LIMIT
        MOV  177,R1
VALV4A:  CMP  R1,TH$(R4) ;COMPARE WITH THRESHOLD
        BLE  VALV4B ; IF (R1-TH)LIMIT,OK,OK
        RTS  PC ;RETURN
VALV4B:  CLR  R1
        RTS  PC
.ENDM VALV$$
```

APPENDIX G

MACRO PICO\$\$

PICO\$\$ is a proportional plus integral controller. It operates on a control table containing setpoint, process value, tuning parameters, high and low limits, and other dynamic controller values.

The first and second tuning parameters are the proportional and integral gains, respectively. The high and low limits are bounds for the setpoint.

The output of the controller is cascaded into another control table (pointed to by REG 5). The controller output is limited by the high and low limits of the cascaded control table.

```

;
.MACRO PICO$$
;-----PI.MAC-----
;A GENERAL PI ALGORITHM R4=IMPOT TABLE, R5=OUTPUT TABLE
; ALL REGISTERS SAVED
;
;
; BEGINING OF PROGRAM
PICO$: TST ST$(R4) ;TEST IF LOOP IS IN AUTO
      BGE OUT2 ;EXIT IF NOT
;
; CHECK SET POINT FOR UPPER LIMITS
MOV RO,-(SP)
MOV R1,-(SP) ;SAVE A REGISTER
MOV R2,-(SP)
MOV R3,-(SP)
MOV HL$+2(R4),-(SP)
MOV HL$(R4),-(SP)
MOV $$$+2(R4),-(SP)
MOV $$$ (R4),-(SP)
FSUB SP
BGE PIA ;IF (HL-S) RESET,OK,OK
MOV HL$(R4),$$$ (R4)
MOV HL$+2(R4),$$$+2(R4)
;
; CHECK SET POINT FOR LOWER LIMITS
PIA: CMP (SP)+,(SP)+ ;POP CELLS
MOV LL$+2(R4),-(SP)
MOV LL$(R4),-(SP)
MOV $$$+2(R4),-(SP)
MOV $$$ (R4),-(SP)
FSUB SP
BLE PIB ;IF (LL-S) OK,OK,RESET
MOV LL$(R4),$$$ (R4)
MOV LL$+2(R4),$$$+2(R4)
PIB: CMP (SP)+,(SP)+ ;POP CELLS
; SET UP ARGUYMENTS AND COMPUTE CONTROL ERROR
MOV $$$+2(R4),-(SP)
MOV $$$ (R4),-(SP)
MOV X$$+2(R4),-(SP)
MOV X$$ (R4),-(SP)
FSUB SP ; E = S - X
; TEST IF POSITIVE OR NEGITATIVE ACTING CONTROL
TST A$$ (R4) ; + OR - WORD
BGE PI1 ;BRANCH IF POSITIVE ACTING
MOV 100000,R0
XOR RO,(SP) ;REVERSE SIGN OF ERROR
PI1: MOV 2(SP),R1 ;STORE RESULTS IN TEMPORARY CELL
MOV (SP),R0

```

```

.PAGE
; COMPUTE PROPORTIONAL ACTION
MOV P1$+2(R4),-(SP)
MOV P1$(R4),-(SP)
FMUL SP ; PROP = P * E
MOV 2(SP),-(SP)
MOV 2(SP),-(SP) ;SAVE RESULTS IN TEMPORARY CELL
;
; SET UP ARGUYMENTS AND COMPUTE INCREMENTAL SUM-OF-ERROR
MOV R1,-(SP) ;GET ERROR
MOV R0,-(SP)
MOV P2$+2(R4),-(SP) ;GET INTEGRAL GAIN
MOV P2$(R4),-(SP)
FMUL SP ; I * E
; ADD RESULTS TO PREVIOUS SUM
MOV SE$+2(R4),R1 ;GET OLD SUM
MOV SE$(R4),R0
MOV R1,-(SP)
MOV R0,-(SP)
FADD SP ; SUM = OLD SUM + I*E
MOV (SP),SE$(R4) ;STORE RESULTS
MOV 2(SP),SE$+2(R4)
FADD SP ;OUT = PROP + SUM
MOV 2(SP),R3 ;STORE OUTPUT
MOV (SP),R2
;
; CHECK OUTPUT FOR WIND UP
;
MOV HL$+2(R5),-(SP) ;HIGH LEVEL FIRST
MOV HL$(R5),-(SP)
FSUB SP ; IF (U-HL) OK,OK, RESET
BGT PI3 ;RESET
;
CMP (SP)+,(SP)+ ;CLEAR CELL
MOV R3,-(SP) ;GET OUTPUT
MOV R2,-(SP)
MOV LL$+2(R5),-(SP) ; GET LOW LEVEL
MOV LL$(R5),-(SP)
FSUB SP ; IF (U-LL) RESET,OK,OK
BLT PI3 ;RESET
CMP (SP)+,(SP)+ ;POP CELL
PI2: MOV R2,U$$+2(R4) ;STORE RESULTS
MOV R3,U$$+2(R4)
; COMPLETE CASCADE COMMUNICATION
MOV R2,S$$+2(R5)
MOV R3,S$$+2(R5)
CMP (SP)+,(SP)+ ;POP CELL
MOV (SP)+,R3
MOV (SP)+,R2
MOV (SP)+,R1
MOV (SP)+,R0

```

```

OUT2:  RTS   PC
      ;
      .PAGE
      ;
      ; RESET ROUTINE
      ;
PI3:   CMP   (SP)+,(SP)+ ;POP CELL
      MOV   R1,SE$+2(R4) ;RESTORE OLD SUM
      MOV   R0,SE$(R4)
      MOV   R1,-(SP)      ;CALCULATE NEW OUTPUT
      MOV   R0,-(SP)
      FADD  SP      ; OUT  = PROP + OLD SUM
      MOV   2(SP),R3    ;SET UP FOR EXIT
      MOV   (SP),R2
      BR    PI2        ;EXIT
      ;
      .ENDM  PICO$$

```

APPENDIX H

MACRO SCAN\$\$

The scan macro has three functions: 1) to drive the A/D hardware, 2) to convert the resulting binary to engineering units, and 3) to store the results in the appropriate control table.

Conversion to engineering units is accomplished by changing the binary count to floating point, optionally performing a square root extraction for flows, and completed by linear extrapolation. Digital filtering is the final option before storing the result.

The macro call must have five arguments (the number of channels to be converted), three tables with extrapolation constants and option switches, and one table with locations for result storage.

```

;
.MACRO SCAN$$ CHAN$,SLOPE$,INTER$,FILCO$,ENTRY$
.NLIST
; CHAN$ = NUMBER OF CHANnELS
; SLOPE$ = START ADDRESS OF LINEAR INTERPOLATION TABLE
; INTER$ = START ADDRESS OF LINEAR INTERPOLATION TABLE
; FILCO$ = START ADDRESS OF FILTER & SQUARE ROOT TABLE
; TWO WORDS PER CHANnel
; 1ST, 1 IF SQUARE ROOT REQUIRED, OTHERWISE, ZERO
; 2ND, FILTER CONST, CONST = 0 MEANS NO FILTERING
; ENTRY$ = START ADDRESS OF TABLE LISTING WHERE TO PUT
ENTRIES
;
; ALL REG SAVED
;
.LIST
;//////// SCAN //////////
;
; THIS ROUTINE IS AN A/D HANDLER FOR THE
; ADAC CONVERTER.
;
;
; M.MEALS 8'77
;
ADSTS$ = 176770 ;ADAC STATUS WORD ADDRESS
ADDAT$ = 176772 ;ADAC DATA WORD ADDRESS
;
;
STBIT$ = 1 ;ADAC START CONVERSION BIT
SQBIT$ = 4 ;ADAC SEQUENTIAL CONVERSION BIT
CHNZR$ = 000 ;ADAC CHANnel ZERO ADDRESS
.PAGE
SCAN$: MOV R0,-(SP)
MOV R1,-(SP) ;SAVE REGISTERS
MOV R2,-(SP)
MOV R3,-(SP)
MOV R4,-(SP)
MOV R5,-(SP)
;
;
MOV CHAN$,R0 ;SET UP LOOP COUNTER
CLR R1 ;SET UP POINTER
CLR R2
MOV ADDAT$,R3 ;SCRATCH EXISTING DATA
MOV STBIT$+SQBIT$+CHNZR$,ADSTS$ ;START & SEQUENTIAL &
CHANnel 0
SCAN1$: TSTB ADSTS$ ;CHECK FOR CONVERSION COMPLETE
BPL SCAN1$ ; COMPLETE CONVERSION
;
MOV ADDAT$,R3 ;GET THE BINARY NUMBER
MOVB STBIT$+SQBIT$+CHNZR$,ADSTS$ ;START ANOTHER

```

CONVERSION

```
BIC    170000,R3    ;STRIP IT TO 12 BITS
MOV    R3,-(SP)    ;STORE IT
;
```

.PAGE

```

;
;
; THIS ROUTINE WAS WRITTEN TO PERFORM A CONVERSION OF THE
; GENERAL ANALOG INPUTS TO ENGINEERING UNITS.
;
; M.MEALS NOV '76
;
;
;
;
; THIS MACRO CONVERTS AN UNSIGNED 16. BIT
; BINARY NUMBER STORED ON THE STACK, INTO
; A TWO WORD FLOATING POINT NUMBER IN PDP-11
; FORMAT. THE FP11 NUMBER IS RETURNED ON THE
; STACK, HIGH ORDER WORD FIRST.
;
; M.MEALS OCT.'76
;
EXPSH$ = 7 ;EXPONENT SHIFT
EXCES$ = 200 ;EXPONENT EXCESS
MANTI$ = -8. ;MANTICA SHIFT
BITCT$ = 16. ;TOTAL BIT COUNT
BINR6$ = 6 ;BINARY NUMBER LOCATION ON STACK
LOR6$ = 4 ;LOW FP11 WORD R6OT
;
;
SCON1$: MOV R1,-(SP) ;SAVE
MOV R2,-(SP) ; ALL
MOV R0,-(SP) ; REGISTERS
;
MOV BINR6$(SP),R0 ; LOAD BINARY NUMBER
CLR R1 ;CLEAR LOW FP11 WORD
TST R0 ; IS BINARY A CLEAN ZERO?
BEQ BISF3$ ;BRANCH IF YES.
MOV BITCT$,R2 ;LOAD EXPONENT COUNT
;
BISF1$: ASL R0 ;ROTATE BINARY NUMBER ONE BIT LEFT
BCS BISF2$ ;BRANCH IF IT LOST A BIT
SOB R2,BISF1$ ;DECREMENT EXPONENT AND LOOP (MAYBE)
;
BISF2$: CLC ;CLEAR CARRY (HIDDEN) BIT
ROR R0 ;ROTATE 0 INTO BIT 31
ROR R1 ;COMPLETE FIRST BIT SHIFT
MOV MANTI$,-(SP) ;THIS IS FOR EPROM BUG
ASHC (SP)+,R0 ;SHIFT MANTICA DOWN WITHOUT HIDDEN BIT
ADD EXCES$,R2 ;ADD IN EXCESS NOTATION
MOV EXPSH$,-(SP) ;THIS IS FOR EPROM ERROR
ASH (SP)+,R2 ;SHIFT EXPONENT INTO POSITION
ADD R2,R0 ;PUT 'UM TOGETHER

```

```

;
BISF3$: MOV R1,BINR6$(SP) ;LOAD LOW FP11 WORD
MOV LOR6$(SP),R1 ;RESTORE R1
MOV R0,LOR6$(SP) ;LOAD HIGH FP11 WORD
MOV (SP)+,R0 ;RESTORE R0
MOV (SP)+,R2 ;RESTORE R2
;
TST FILCO$+2(R1) ;WAS SQUARE ROOT REQUIRED ?
BEQ SCON3$ ;BRANCH IF NO
JSR PC,SQRT1$ ;OTHERWISE, DO IT TO IT !
;
.PAGE
SCON3$: MOV SLOPE$+2(R1),-(SP) ;SET IT UP
MOV SLOPE$(R1),-(SP) ;FOR SCALE FACTOR (SLOPE)
FMUL SP ;MULTIPLY IT OUT
MOV INTER$+2(R1),-(SP) ;SET IT UP
MOV INTER$(R1),-(SP) ;FOR INTERCEPT
FADD SP ;ADD IT IN
;
TST FILCO$(R1) ;WAS FILTERING REQUIRED ?
BEQ SCON4$ ;BRANCH IF NO
MOV ENTRY$(R2),-(SP) ;LOAD FILTER OUTPUT ADDRESS
MOV FILCO$(R1),-(SP) ;AND FILTER CONSTANT
JSR PC,FILT$$ ;FILTER IT
BR SCON5$ ;LOOP FOR MORE
;
SCON4$: MOV ENTRY$(R2),R3 ;GET ADDRESS
MOV (SP)+,(R3) ;STORE HIGH ANSWER
MOV (SP)+,2(R3) ;STORE LOW ANSWER
SCON5$: TST (R2)+ ;MOVE POINTERS
CMP (R1)+,(R1)+
DEC R0 ;LOOP UNTIL DONE
BNE SCON6$
;
MOV (SP)+,R5
MOV (SP)+,R4
MOV (SP)+,R3
MOV (SP)+,R2
MOV (SP)+,R1
MOV (SP)+,R0
RTS PC ;RETURN TO OPERATING SYSTEM
SCON6$: JMP SCAN1$ ;SOB WON'T FIT
;
; THIS MACRO PERFORMS A SQUARE EXTRACTION OF THE
; TWO WORD FLOATING POINT NUMBER STORED ON
; THE STACK.
; THE ANSWER IS RETURNED ON THE STACK
; HIGH ORDER WORD FIRST
;
; OPERATION: [(SP),(SP)+2] SQRT[(SP),(SP)+2]
; IF RESULT IS REAL, ELSE [(SP),(SP)+2] 0

```

```

;
; CONDITION CODES N:SET IF RESULT IMIGINARY (RESULT
CLEARED), ELSE CLEARED
;      Z:SET IF RESULT = 0, ELSE CLEARED
;      V:CLEARED
;      C:CLEARED
;
; M.MEALS NOV'76
;
RO = %0
SP = %6 ;REGISTER DEFINITION
;
SQR1$: MOV RO,-(SP) ;SAVE REGISTER
MOV 4(SP),RO ;GET HIGH ORDER ARGUMENT
;
BMI SQRER$ ;ERROR IF NEGATIVE NUMBER
BEQ SQRZR$ ;FAST EXIT IF ZERO
;
MOV 3,-(SP) ;LOAD ITERATION COUNT
;
ASR RO ;FORM INITIAL ESTIMATE
ADD 20100,RO ;
;
;USE ONLY THE HIGH ORDER HALF AT FIRST 'CAUSE IT MAKES
; FDIV & FADD EVER SO MUCH QUICKER THAT WAY !
;
CLR -(SP) ; E (LOW ORDER)
MOV RO,-(SP) ; E (HIGH ORDER)
;
CLR -(SP) ; X (LOW ORDER)
MOV 14(SP),-(SP) ; X (HIGH ORDER)
;
CLR -(SP) ; E (LOW ORDER)
MOV RO,-(SP) ; E (HIGH ORDER)
;
SQR2$: FDIV SP ; X/E
FADD SP ; X/E + E
SUB 200,@SP ; (X/E + E) /2
;
DEC 4(SP) ;DECREMENT LOOP COUNT
BEQ SQR3$ ;EXIT HOLE
;
MOV 14(SP),-(SP) ; X (LOW ORDER)
MOV 14(SP),-(SP) ; X (HIGH ORDER)
;
MOV 6(SP),-(SP) ; E (LOW ORDER)
MOV 6(SP),-(SP) ; E (HIGH ORDER)
;
BR SQR2$ ; ITERATE AGAIN
;
;

```

```

SQRT3$:  MOV  (SP)+,10(SP)  ; LOAD UP ANSWER FOR CLEAN EXIT
        MOV  (SP)+,10(SP)  ;  DITTO
        ;
        TST  (SP)+      ; FLUSH CELL
        ;
        MOV  (SP)+,RO  ; RESTORE REGISTER
        CCC  ; CLEAR STATUS WORD FOR CLEAN EXIT
        BR   SQRED$    ; LEAVE
        ;
SQRZR$:  MOV  (SP)+,RO  ;RESTORE REGISTER
        CLR  2(SP)      ; MAKE FOR CLEAN SWEEP
        BR   SQRED$    ; AND HIT THE ROAD
        ;
SQRER$:  MOV  (SP)+,RO  ;RESTORE REGISTER
        CLR  2(SP)      ; FLUSH JUNK
        CLR  (SP)      ; 'CAUSE SUMPTUM AINT RIGHT
        SEN          ;SET NEGATIVE BIT FOR ERROR CHECK
        ;
SQRED$:  RTS   PC      ;RETURN
        ;
        .ENDM  SCAN$$

```

APPENDIX I

MACRO PARI\$\$

PARI\$\$ is a handler for parallel input. The standard valve station for which this routine was written has several pushbuttons which the operator can use to turn controllers on or off. PARI\$\$ accesses these pushbuttons via parallel I/O hardware and determines if the request is valid. If the request is to turn on a controller, PARI\$\$ will perform a bumpless transfer for the appropriate control table.

The arguments for PARI\$\$ specify which valves and which higher level control loops are active (the first two arguments), a table containing bumpless transfer requirements, and two tables containing status on/off truth tables.

```

;
.MACRO PARI$$ ACTIV$,ACTIP$,BMPTB$,STON$,STOF$
.NLIST
; ACTIV$ = WORD WITH ACTIVE VALVE BIT PATTERN
; ACTIP$ = WORD WITH ACTIVE PBL BIT PATTERN
; BUPTB$ = ADDRESS OF BUMPLESS TRANSFER LINK TABLE
; 1ST CONTROL TABLE
; 2ND POINTER TO BUMPLESS DEFINITION
; STON$ = ADDRESS OF STATUS ON TABLE
; STOF$ = ADDRESS OF STATUS OFF TABLE
;
;
; ALL REGISTERS SAVED
;
.LIST
;
;//////////////////// PARALLAL IN //////////////////
;
; THIS ROUTINE MONITORS THE PARALLEL INPUT OR NETWORK
; PORT AND OPERATES ON ANY REQUESTED CHANGE IN STATUS
;
PARIN$: MOV RO,-(SP) ;SAVE EVERYTHING
MOV R1,-(SP)
MOV R2,-(SP)
MOV R3,-(SP)
MOV R4,-(SP)
MOV R5,-(SP)
;
; GET PARALLEL INPUT PORT CONTENTS
;
MOV DRINB1,R2 ;PORT 1
MOV DRINB2,R3 ;PORT 2
;
; COMPARE TO PREVIOUS INPUT
;
MOV DRB1$,RO ;GET PREVIOUS SETTING
MOV DRB2$,R1
;
; CHECK FOR RUN BIT
;
BIT RB$,R2 ;IS THE RUN BIT SET ?
BNE PARI1$ ;BRANCH IF YES
CLR STA1$ ;OTHERWISE, CLEAR
CLR STA2$ ; STATUS WORDS
CLR STA1A$
CLR STA2A$
BIT RB$,RO ;RB$ IS RUN BIT, WAS IT SET LAST TIME ?
BEQ PRIEX$ ;EXIT IF NOT
BIC RB$,DRB1$ ; OTHERWISE, CLEAR RB
BR REQ4$ ;AND EXIT
;

```

```

PARI1$:  MOV    ACTIV$,R4  ;GET ACTIVE BITS
        MOV    ACTIP$,R5
        COM    R4
        COM    R5
        BIC    R4,R2      ;CLEAR TO ACTIVE BITS
        BIC    R5,R3
        ;
        BIS    RB$,R2      ;SET RUN BIT FOR LATCH
        MOV    R2,DRB1$    ;SAVE THEM FOR LATCH
        MOV    R3,DRB2$
        ;
        ;
; CHECK FOR NEW BITS
        ;
        BIC    R0,R2      ;ANY THING NEW ?
        BNE    REQ$      ;BRANCH IF YES
        BIC    R1,R3
        BNE    REQ$
        ;
.PAGE
; ANYTHING FROM NETWORK ?
        ;
        TST    STA1A$      ;ANYTHING NEW ?
        BNE    NREQ$      ;BRANCH IF YES
        TST    STA2A$
        BNE    NREQ$
        ;
; NOTHING NEW, RESET STATUS WORDS
        ;
PRIEX$:  MOV    100000,R1
        MOV    STA1$,R2    ;GET CURRENT STATUS
        MOV    STA2$,R3
        MOV    ACTIV$,R4   ;GET ACTIVE LIST
        MOV    ACTIP$,R5
        MOV    BMPTB$,R0   ; AND POINTER
        ;
PRIX0$:  MOV    1,-(SP)
        ASHC   (SP),R4     ;CHECK FOR ACTIVE BIT
        BCS    PRX1$      ;BRANCH IF ACTIVE LOOP
        BEQ    PRIEE$      ;EXIT IF FINISHED
        ASHC   (SP)+,R2    ;SHIFT AGAIN
        BR     PRX0$      ;AND LOOP FOR MORE
        ;
PRX1$:   ASHC   (SP)+,R2    ;CHECK FOR ON LOOP
        BCS    PRX2$      ;BRANCH IF ON
        BIC    R1,@(R0)    ;OTHERWISE, TURN IT OFF
        BR     PRX3$
        ;
PRX2$:   BIS    R1,@(R0)    ;TURN IT ON
PRX3$:   CMP    (R0)+,(R0)+ ;NEXT LOOP
        BR     PRX0$

```

```
;  
; EXIT POINT  RESTORE REGISTERS  
;  
PRIEE$:  TST  (SP)+  ;POP CELL  
MOV  (SP)+,R5  
MOV  (SP)+,R4  
MOV  (SP)+,R3  
MOV  (SP)+,R2  
MOV  (SP)+,R1  
MOV  (SP)+,R0  
RTS  PC      ;RETURN TO WHERE-EVER !  
;  
.PAGE
```

```

;
; NETWORK REQUEST
;
NREQ$: MOV STA1A$,R2
MOV STA2A$,R3
CLR STA1A$ ;CLEAR IT FOR NEXT TIME
CLR STA2A$

;
BIC R4,R2 ;STRIP TO ACTIVE BITS
BNE REQ$ ;BRANCH IF VALID REQUEST
BIC R5,R3
BEQ PRIEX$ ;EXIT IF NOT VALID REQUEST
;
; REQUEST HANDLER
;
REQ$: CLR R0 ;THIS IS A COUNTER
CLR R1
REQ1$: INC R1
MOV 1,-(SP)
ASHC (SP),R2 ;IS THIS THE REQUESTED BIT ?
BEQ REQ3$ ;EXIT WHEN FINISHED
ASHC (SP)+,R4 ;ACTIVE LOOP ?
BCC REQ2$ ;IF YES, INCREMENT COUNTER
DEC R0
REQ2$: INC R0
BR REQ1$ ;LOOP TILL DONE
;
REQ3$: TST (SP)+ ; POP CELL
ADD -32.,R1 ;SET R1 UP AS
NEG R1 ; A LEFT SHIFT COUNTER
CLR R2 ;RECONSTRUCT REQUEST WORD (WITH ONLY ONE BIT !!!)
MOV 1,R3
ASHC R1,R2 ;PLACE IT
ASL R0 ;SPACE POINTER OUT
ASL R0
;
; TURN ON REQUEST ?
;
BIC STA1$,R2 ;TURN ON REQUEST ?
BNE TRON$ ;BRANCH IF YES
BIC STA2$,R3
BNE TRON$
;
;MUST HAVE BEEN A TURN OFF
;
BIC STOF$(R0),STA1$ ;TURN IT OFF
BIC STOF$+2(R0),STA2$
;
REQ4$:
BR PRIEX$ ;EXIT

```

```

;
; IS TURN ON ALLOWED ?
;
TRON$:  MOV  STON$(R0),R4  ;GET REQUIRED STATUS
        MOV  STON$+2(R0),R5
        BIC  STA1$,R4  ;DOES THE SHOE FIT ?
        BNE  PRIEX$    ;EXIT IF NOT
        BIC  STA2$,R5
        BNE  PRIEX$
;
; TURN ON WITH BUMPLESS TRANSFER
;
        BIS  R2,STA1$  ;SET NEW BIT
        BIS  R3,STA2$
;
        MOV  BMPTB$+2(R0),R2  ;GET BUMPLESS TRANSFER STARTING
ADDRESS
        MOV  BMPTB$+6(R0),R3  ;GET ENDING ADDRESS
TRON1$:  CMP  R3,R2  ;CHECK FOR DONE
        BLE  PRIEX$    ;EXIT WHEN DONE
        MOV  (R2)+,R4  ;LOAD ADDRESS
        MOV  (R2)+,R5  ; INPUT AND OUTPUT
        MOV  (R4)+,(R5)+ ;MOVE INFO
        MOV  (R4)+,(R5)+
        BR   TRON1$  ;LOOP
;
.ENDM  PARI$$

```

APPENDIX J

MACRO PARO\$\$

PARO\$\$ is a routine for driving the parallel output hardware. On the standard valve station for which this macro was written, valve relays and status relays are activated by parallel output bits. The function of PARO\$\$ is to collect, pack, and send the appropriate bit patterns for the parallel output hardware.

```

;
.MACRO  PARO$$
;
;////////// PARALLEL OUT //////////
;
;  THIS MACRO IS A PARALLEL OUT HANDLER FOR
;  THE DLV11 CARD AND STANDARD INTER CONNECTION BOARD
;
;
;  ALL REG SAVED
;
;  M.MEALS  3/78
;
PAROU$:  MOV  R0,-(SP)  ;SAVE 'UM ALL
MOV  R1,-(SP)
MOV  R2,-(SP)
MOV  R3,-(SP)
MOV  R4,-(SP)
MOV  R5,-(SP)
;
MOV  VLBUF$,R0  ;GET VALVE DRIVE WORD
MOV  STA1$,R1  ;GET VALVE AUTO WORD
ASR  R1  ; RUN BIT
MOV  4,R2  ;GET AUTO BIT
MOV  3,R3  ;GET VALVE DRIVE WINDOW
MOV  5,R4  ;SET LOOP CONTROL
CLR  R5  ;INITIALIZE OUTPUT WORD
;
PARO1$:  MOV  R0,-(SP)  ;SAVE VALVE DRIVE WORD
COM  R3
BIC  R3,R0  ;STRIP DRIVE WORD TO SINGLE VALVE
COM  R3
BIS  R0,R5  ;GENERATE OUTPUT WORD
MOV  (SP)+,R0  ;RESTORE FULL DRIVE WORD
ASL  R0  ;SHIFT FOR NEXT VALVE
;
ASR  R1  ;WAS IT IN AUTO?
BCC  PARO2$  ;BRANCH IF NO
BIS  R2,R5  ;SET AUTO BIT
PARO2$:  MOV  3,-(SP)
ASHC  (SP)+,R2  ;SHIFT AUTO BIT AND WINDOW
SOB  R4,PARO1$  ;LOOP THRU 5 VALVES
;
MOV  STA2$,R4  ;GET FIRST STATUS WORD
ASR  R1  ; WAS VALVE SIX IN AUTO ?
BCC  PARO3$  ;BRANCH IF NO
BIS  100000,R4  ;SET AUTO BIT
ASR  R0  ;GET DRIVE WORD
ASR  R0
BIC  117777,R0  ;STRIP IT OT VALVE 6
BIS  R0,R4  ;AND SET IT

```

```
    ;  
    ;  
PARO3$:  MOV  R5,DROUB1  ;SEND IT AWAY  
        MOV  R4,DROUB2  
    ;  
        MOV  (SP)+,R5    ;EXIT  
        MOV  (SP)+,R4  
        MOV  (SP)+,R3  
        MOV  (SP)+,R2  
        MOV  (SP)+,R1  
        MOV  (SP)+,R0  
        RTS   PC  
    .ENDM  PARO$$
```

APPENDIX K

MACRO STASK\$

STASK\$ is a software listing of three standard task definitions for the real time monitor. These three tasks call in SCAN\$\$, PARI\$\$, and PARO\$\$ for A/D conversion and parallel I/O.

APPENDIX L

MACRO TASK\$

TASK\$ is a standard task definition macro for calling in standard controllers with the real time monitor. It is written for cascaded controllers written in a table driven format. The arguments of the macro specify the controller and pointers to the master and slave control tables. It calls the controller via a JSR PC,CONTROLLER. Before returning control to the real time monitor, TASK\$ resets the task interrupt clock for the next pass.

```
.MACRO TASK$ CONT,MASTER,SLAVE
;THIS MACRO CALLS A STANDARD CONTROLLER VIA A JSR PC,CONT
;AFTER LOADING THE STANDARD PONTERS
;
; R1 MUST CONTAIN THE ADDRESS OF THE INTERRUPT CLOCK
; REG 4,5 SCRATCHED
;
MOV  MASTER,R4 ;LOAD POINTER TO MASTER
MOV  SLAVE,R5  ;LOAD POINTER TO SLAVE
JSR  PC,CONT
MOV  ST$(R4),(R1) ;RELOAD INTERRUPT CLOCK
RTS  PC
.EMDM TASK$
```

APPENDIX M

MACRO LELA\$\$

LELA\$\$ is the standard lead algorithm macro. It uses the normal parameter table format with the following exceptions:

1. The first corner frequency (a tuning parameter) is stored in position 2.
2. The second corner frequency (a tuning parameter) is stored in position 4.
3. The gain is stored in position 4.

```

.MACRO LELA$$
;----- LEAD LAG CONTROLLER -----
;
; GAIN G1$
; HL$ TAU1
; LL$ TAU2
;
; S$$ PAST INPUT
; SE$ PAST OUTPUT
; U$$ GAIN ( OUTPUT - PAST OUTPUT )
;
.MACRO PUT DUM
MOV DUM+2(R4),-(SP)
MOV DUM(R4),-(SP)
.ENDM
.MACRO ONE
MOV FLT1+2,-(SP)
MOV FLT1,-(SP)
.ENDM
;
LL:
TST ST$(R4) ; IS IT IN AUTO
BMI LL$2 ; BRANCH IF YES
RTS PC
;
LL$2: BIT GAP$,ST$(R4) ; DOES IT NEED NEED BUMPLESS
TRANSFER
BNE LL$3 ; BRANCH IF NO
MOV X$$$(R4),S$$$(R4) ; RESET OLD INPUT
MOV X$$+2(R4),S$$+2(R4)
MOV X$$$(R4),SE$(R4) ; RESET OLD OUTPUT
MOV X$$+2(R4),SE$+2(R4)
BIS GAP$,ST$(R4)
;
LL$3: ONE
PUT HL$
FADD SP ; (1 + TAU1)
;
PUT X$$
FMUL SP ; IN(1 + TAU1)
;
ONE
PUT HL$
FSUB SP ; (1 - TAU1)
;
PUT S$$
FMUL SP ; IN1(1 - TAU1)
;
FADD SP ; IN(1 + TAU1) + IN1(1 - TAU1)
;
ONE

```

```

PUT LL$
FSUB SP ; (1 - TAU2)
;
PUT SE$
FMUL SP ; OUT1(1 - TAU2)
;
FSUB SP ; IN(1 + TAU1) + IN1(1 - TAU1) - OUT1(1 - TAU2)
;
ONE
PUT LL$
FADD SP ; (1 + TAU2)
;
FDIV SP ; IN(1+TAU1) + IN1(1-TAU1) - OUT1(1-TAU2) /
(1+TAU2)
;
MOV 2(SP),-(SP) ;SAVE RESULTS
MOV 2(SP),-(SP)
;
PUT SE$
FSUB SP ; DEVIATION = OUT - OUT1
;
PUT G1$
FMUL SP ; OUTPUT = GAIN * (OUT - OUT1)
;
MOV (SP),U$$ (R4) ;SAVE OUTPUT
MOV 2(SP),U$$+2(R4)
;
MOV 200, -(SP)
MTPS (SP)+ ;SYNCHRONIZATION
;
MOV SE$+2(R5), -(SP)
MOV SE$(R5), -(SP)
FADD SP ;CASCADE OUTPUT
;
MOV (SP)+, SE$(R5)
MOV (SP)+, SE$+2(R5) ;STORE RESULT
;
CLR -(SP)
MTPS (SP)+
;
MOV (SP)+, SE$(R4)
MOV (SP)+, SE$+2(R4) ;STORE OUT
;
MOV X$$ (R4), S$$ (R4) ;STORE IN1
MOV X$$+2(R4), S$$+2(R4)
;
RTS PC
.ENDM LELA$$

```

VITA

The author was born at Hollywood, California on October 21, 1947 to Mr. and Mrs. Jack H. Meals. After moving to Memphis, Tennessee he attended Peabody Elementary School, Bellevue Junior High School, and Central High School, graduating in June 1965. He entered Christian Brothers College in Memphis, Tennessee in September 1966 and transferred to The University of Tennessee, Knoxville in September 1968, graduating in December 1971 with a Bachelor of Science degree in Chemical Engineering. He then worked for Chemical Separations Corporation in Oak Ridge, Tennessee until returning to The University of Tennessee, Knoxville for graduate studies in September 1973. He received a Master of Science degree in Chemical Engineering in March 1977 and a Doctor of Philosophy degree in Chemical Engineering in August 1984. He is currently working for Rockwell International in Golden, Colorado.