MASTER

**Design of Data Acquisition and Control System
for Environmental Monitoring and Microclimate Control¹**

DISCLAIMER

This book was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

D. P. Fuoco

**The Pennsylvania State University
University Park, PA 16802**

1979

EC-77-S-02-4397

Department of Energy under contract No.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Reg

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

The Pennsylvania State University
The Graduate School
Department of Electrical Engineering

Design of Data Acquisition and Control Systems
for Environmental Monitoring and Microclimate Control

A Thesis in
Electrical Engineering

by
Daniel Paul Fuoco

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 1979

We approve the thesis of Daniel Paul Fuoco.

Date of Signature:

4-19-79

Jon G. Bredeson
Jon G. Bredeson, Associate
Professor of Electrical
Engineering, Thesis Advisor

4-19-79

Paul T. Hulina
Paul T. Hulina, Associate
Professor of Electrical
Engineering, Thesis Committee
Member

4-20-79

Charles T. Morrow
Charles T. Morrow, Associate
Professor of Agricultural
Engineering, Thesis Committee
Member

4/23/79

William J. Ross
William J. Ross, Professor
and Head, Department of Elec-
trical Engineering

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

Abstract

Modern fruit growing techniques rely heavily on automated control equipment to improve yields and reduce operating expenses. This thesis is part of a long term project whose ultimate goal is the automation and optimization of fruit production. This thesis describes the equipment that has been developed to date. Also presented is a brief methodology that considers how this equipment came to be. It should be noted that there are many related uses for climate control, most of which could make good use of this same type of equipment. In fact, any physical process that requires close supervision of the natural elements is a prime candidate for this type of endeavor.

Table of Contents

	<u>Page</u>
ABSTRACT.....	iii
LIST OF FIGURES.....	v
I. INTRODUCTION.....	1
1.1 Purpose of the System.....	1
1.2 Methodology: How "Best" to Build (Design) the System.....	3
1.3 Elements of the System.....	5
1.4 Software Considerations.....	8
II. DETAILS THAT PERTAIN TO A PARTICULAR BUS STRUCTURE...	11
2.1 Parts of the S-100 BUS System.....	11
2.2 Parts of the MULTIBUS System.....	11
2.3 Which BUS is the better BUS.....	11
III. REAL-TIME PROCESS CONTROL.....	16
3.1 Real-Time Process Control Requirements.....	16
3.2 Hardware Description.....	18
3.3 Firmware Description.....	20
IV. APPLICATION ONE USERS GUIDE.....	28
4.1 APPL1 Start-Up.....	28
4.2 Operator Commands and Responses.....	28
4.3 Data Recording Format.....	31
V. CONCLUSION.....	33
APPENDIX A List of Materials.....	37
APPENDIX B.1 S-100 System BUS Structure.....	41
APPENDIX B.2 Intel Multibus.....	47
APPENDIX C Firmware Flow Diagrams.....	51

List of Figures

<u>Figure</u>		<u>Page</u>
1	Component Block Diagram.....	2
2	Functional Block Diagram.....	6
3	S-100 BUS System Block Diagram.....	12
4	MULTIBUS System Block Diagram.....	13
5	APPL1 Command Summary.....	29
6	Fixed Record Format.....	32

CHAPTER I

Introduction

1.1 Purpose of the System

This thesis describes two totally independent data acquisition and control systems. One is based on an INTEL Corporation μ Computer Development System Model MDS 220/230, the other is based on an IMSAI Corporation μ Computer Development System Model PCS-80/30. In both instances, the general function of the corresponding data acquisition and control system is similar.

The overall structure of the data acquisition and control equipment is depicted in Figure 1, Component Block Diagram. As shown, the "data" is supplied by a bank of sensors that convey information about environmental conditions. Temperature, humidity, barometric pressure, wind speed and directional data are fed into the system in the form of analog signals. These signals are converted to digital form for subsequent logging and analysis. When environmental conditions warrant it, the control section activates pumps, valves, heaters, or combinations thereof, that hopefully will make the environmental conditions favorable to the growing of the crop.

Actual details of the "analysis" portion of this system are not the purpose of this report as they are largely experimental at this time. What is of interest, is that the structure of the system is flexible enough to adapt to a

Figure 1. Component Block Diagram

wide range of analyses algorithms by virtue of the embedded processor. In fact, the first application of this system is intended for research purposes such that sufficient data may be acquired for development of suitable analysis and control algorithms. Later, when there are reliable analysis and control algorithms available they will be "programmed" into the system. This, in turn, will form a closed-loop acquisition and control system capable of autonomous operation.

1.2 Methodology: How "Best" to Build (Design) the System

From the outset, numerous factors played crucial roles that influenced design planning in pursuit of the system described in Section 1.1. Among them are:

- Initial efforts call for a low cost, low volume system.
- The profit motive is not large. Research and development is the higher goal (at first).
- On-line development time must be kept to a minimum.

These factors (plus others of lesser importance) did not of themselves rule out the use of commercial equipment. Unfortunately, a suitable "turn-key" system that has all the features needed is not available. The single most important feature desired is that the system be versatile. Commercial units are largely limited in scope.

As a result, it was decided to design the system. The question to be answered is where to start?

The initial level of the design opens up many interesting

trade-offs. For example, if resources were available such that a "start from scratch" approach were possible the design could have begun at the component level. In this case the work would most likely have commenced with the design of a micro-computer based subsystem. Specifically, the processor and memories would be built from components. The data acquisition and control hardware development would follow. Unfortunately, neither the resources nor the time required could be found to support the lower level development.

The next higher level in the chain is to design a system around circuit boards purchased from vendors. In essence, this is what was done. However, even here there were many decisions to make. There are an incredibly large number of manufacturers of processor boards and memories. There are few, however, that satisfy all of the following requirements:

- compatible micro-computer and system-expansion boards
- inexpensive and ready-to-use development systems
- good documentation supported by well trained service personnel

The stipulation that the system design center around a micro-computer should be explained. It is not essential that this be the case, but for all practical purposes there is no better alternative⁽¹¹⁾. The compatibility requirement between micro-computer and system-expansion boards is self-explanatory. Compatibility is assured if all circuit boards

conform to the rules of a standard interface. A BUS provides a standard interface between computer and expansion boards. BUS considerations are addressed in Appendix B.

Altogether, weighing each requirement in turn, it was decided (initially) to purchase an IMSAI PCS-80/30 μ Computer Development System. At the time of purchase it represented the lowest cost system available which filled all requirements. It has the advantage of being directly compatible with the S-100 BUS. This makes it extremely easy to buy the expansion boards, some purchased from IMSAI and others from a variety of manufacturers. Later in the project, additional money became available. Although it was felt that the IMSAI system's performance was satisfactory, it was decided that performance could be improved with an INTEL SBC-80 μ C-based system, the newly introduced MDS Model 220/230. Slightly higher in price than the IMSAI hardware it was felt that it would be somewhat more reliable, an observation borne out in practice. Again, the Single Board Computers (SBC) conform to a BUS standard, INTEL's own MULTIBUS, assuring that expansion boards would be readily available⁽¹¹⁾. A list of materials for each system is included in Appendix A.

1.3 Elements of the System

There are three types of hardware elements within the system (see Figure 2).

1. a micro-processor based processor and associated memory

Figure 2. Functional Block Diagram

2. processor to peripheral equipment interface circuitry (shown in the figure as dashed lines)
3. peripheral devices.

Processor: The role of the processor is that of the master controller - it allocates resources on a need basis, directing processor to peripheral, peripheral to processor, and peripheral to peripheral data transfers. It is the heart of the system.

Interface: Processor to peripheral equipment interface circuitry (later referred to as INPUT/OUTPUT or abbreviated I/O) couples the processor to the corresponding peripheral device. Processor and peripherals communicate with a well designed protocol allowing the processor to deal with each external device in a fashion that is amenable to both. For example, the device that serves as an operator console may well be a teletype, a CRT terminal or one of these same devices at the end of a telephone link (served by a MODEM).

In each case the processor is not aware of the device type or the distance apart, as all these devices have the same standard interface requirements. The three remaining peripherals (A/D Converter, Disk Storage, Digital Controller) also interface with the processor through a standard interface arrangement, but here the interface is the BUS, be it S-100 or MULTIBUS.

Peripheral: Peripheral devices are those system elements which the processor controls through combination interface circuitry and program interaction. These devices include,

but are not necessarily limited to the following:

1. an analog-to-digital (A/D) converter sub-system
2. a digital output controller
3. two operator consoles, one LOCAL, one REMOTE
4. a mass storage device

Each of these devices is fully described in the appropriate Hardware Reference Manual (4,5,6,7).

1.4 Software Considerations

Terminology: Software vs. Firmware

Before proceeding to describe the system software it is most important that a distinction be made between that software which is development oriented and that which is applications oriented. The following definitions are offered as an aid in that effort:

Definitions

1. Software - the group of computer programs which allow only well defined inputs and produce output which is independent of system application. For example, an assembler (or equivalently, a compiler for a higher level language such as FORTRAN) is a development tool that facilitates quicker and more efficient creation of other programs. For purposes of assembly, the contents of the program to be assembled does not matter, nor does its intended use. What does matter is that the form in which the program to be assembled is written must conform

to the rules as defined by the Assembler. Other examples of software are programs which transfer data (where the data is in an organized file-like form) between peripheral devices, file editors (similar to word processors) and debug facilities. All of these program types, plus others, are provided with both the INTEL and IMSAI Development System Software Packages^(1,2). The collection of all software programs is often called DOS - Disk Operating System in recognition of the fact that these programs are stored in disk files. They are brought into operation when a user issues a command that subsequently loads the desired file. INTEL refers to its Operating System as ISIS-II⁽¹⁾ while the IMSAI Operating System is known as IMDOS⁽²⁾.

2. Firmware - when programs cross the nebulous boundary from general purpose (software) into special purpose applications programming it is known as firmware. What makes the boundary separating categories nebulous is that trying to distinguish between what is general purpose and that which is special purpose is sometimes difficult. To some individuals only code that is resident within the machine at all times qualifies as firmware. Others hold that a less stringent qualification be invoked, namely, that code which is essential to the total performance of the application system be included as well.

For purposes of this thesis the later qualifications will be accepted for the simple reason that resident code in the form of Read Only Memory (ROM) currently exists within each development system for the sole purpose of loading program files from a peripheral device, usually the disk. When the loaded program is a development program (assembler, compiler, editor, debug or similar) it is referred to as software, but when the loaded program is an applications program it is called firmware. In effect, loading and executing the firmware program APPL1, described in Section 3.3, transforms the development system into a data acquisition and logging system. At some later date it will no longer be necessary to "load" the data acquisition program from disk as it itself will be stored in ROM. This, however, doesn't alter the fact that now, as then, APPL1 is most definitely firmware.

No additional mention of software will be made as these programs are fully described in the INTEL ISIS-II User's Guide⁽¹⁾ and the IMSAI IMDOS Manual⁽²⁾. Details of assembler concepts, editor usage, debug facilities plus more can be found in the appropriate operating systems manual. Firmware description is deferred to Section 3.3.

CHAPTER II

Details that Pertain to a Particular BUS Structure

2.1 Parts of the S-100 BUS System

Figure 3, S-100 BUS System Block Diagram is a redrawing of Figure 2 in which the generic labels have been replaced by specific component titles. A complete list of materials plus a brief description of each component is available in Appendix A. To compliment the list of materials, the S-100 System BUS Structures is presented in Appendix B.

2.2 Parts of the MULTIBUS System

Figure 4, MULTIBUS System Block Diagram shows the components that comprise the second system to be built. Again, as in Figure 3, manufacturer's titles replace the generic labels. Similarly, Appendix A contains a list of materials and Appendix B describes the MULTIBUS itself.

2.3 Which BUS is the Better BUS

When making judgements about the virtue of one BUS standard versus another innumerable secondary questions arise such as:

- Just how standard is this so-called standard BUS?

Do manufacturers take liberties with interpretation of BUS rules because they are vague or inconsistent?

- Equally important, how many manufacturers build circuit boards that conform to the BUS of your choice and can

Figure 3. S-100 Bus System Block Diagram

NOTE [1] MDS - 220 Contains a single-density disk drive
 MDS - 230 Contains a two double-density disk drives in a separate chassis.

Figure 4. Multibus System Block Diagram

you attain all necessary components (if you wish to buy rather than build)?

•Are the manufacturers reputable?

Of course the primary question remains: Which is the better BUS? The answer is possibly not all that simple. Reference articles 9 through 13 may provide some clues to which is potentially better but the real judgement must await actual usage. Having made use of both BUSSES to an equal extent, the answer is simple. The MULTIBUS is the superior BUS. Unlike S-100 BUS, MULTIBUS is truly STANDARD. S-100 leaves entirely too many details to chance and manufacturers invariably interpret the rules to their advantage with potentially adverse affects on system designers. Too much time was spent on STANDARDIZING S-100 circuit boards to ever be convinced that the STANDARD is adequately defined or enforced.

The manufacturers of BUSABLE circuit boards are (it seems) almost equally divided between S-100 and MULTIBUS. For flexibility you take your choice. Either way you can get all necessary components rather easily. In the literature S-100 is often synonymous with "hobbyest" while MULTIBUS conveys a much more reputable tone. The distinction is more than mere nomenclature. As a rule, MULTIBUS manufacturers are far more commercially oriented than their S-100 counterparts and consequently more reliable. S-100 circuit boards were purchased, assembled and tested and tapping errors on the boards seem highly unlikely. Tapping errors occur when circuit layout does not match circuit schematic because

traces (copper paths connecting components) are missing or misplaced. Not only wouldn't the boards work, but without modification they could never work.

Until S-100 advocates improve their performance considerably, they must rate a distant second to MULTIBUS. Even when they have removed the tarnished label of "hobbyist" from themselves they will be hard pressed to attain parity. MULTIBUS is inherently superior to S-100. The master-slave capabilities of the MULTIBUS are better suited for applications that call for the use of multi-processor configurations or high-speed direct-memory access (DMA) operations.

CHAPTER III

Real-Time Process Control

3.1 Real-Time Process Control Requirements (13)

To design or even specify a modern data acquisition system (DAS) for process control and monitoring at least six parameters are relevant:

- the required accuracy of the data
- the number of input channels
- the volume of the data
- the dynamic range of the data
- the duration of the test event
- the data bandwidth of each channel

The above parameters are listed in decreasing order of importance as seen by the data collection portion of this system. Since the measured signals are inherently static, the last three parameters are not of great importance. The duration of the test event will not have much bearing since data sampling rate can always be much faster than data rate of change. This is because data is a slowly changing DC (voltage) level.

The following specification summary is offered as a "quick-glance" into the range of operation envisioned for the DAS. These specifications are not to be considered as absolute limits, but rather, as a reasonable compromise of performance and economy. It remains to be seen if these parameters are totally adequate but they do reflect current equipment in use.

<u>Specification Summary</u>			<u>Unit</u>
accuracy of the data	12		BITS
number of channels (sensors)	variable: 24-96		----
input voltage range	adjustable: 1-1000		μ V
scan interval	adjustable: 1-9999		SEC

The term REAL TIME requires some clarification. All data acquisition and logging proceeds in real-time, i.e., the "moment" it is accessed it is recorded. Acquisition and recording times may be separated by milliseconds (10^{-3} SEC) but time intervals of this magnitude are not usually significant.

Analysis and control times, however, will necessarily lag acquisition. If the time lag is small, the acquisition, analysis and control operations are said to occur in REAL-TIME. If analysis is done long after acquisition, it is not in real-time and is often called OFF-LINE analysis. Similarly, real-time analysis is sometimes called ON-LINE analysis. REAL-TIME (ON-LINE) processing is essential for control applications.

Real Time Process Control demands that the acquisition and control unit be capable of ON-LINE data processing. Use of a microprocessor in the acquisition unit makes it possible to process the data ON-LINE. Consequently as events happen, they are:

- sensed
- recorded (optional)
- analysed
- controlled

Combining all these features in a single unit provides for Real-Time Process Control capability. Both of the data acquisition and control systems developed to date possess these capabilities. As mentioned in the introduction, initial efforts call for the first two capabilities to be employed (sensing and recording) with the later two (analysis and control) to be developed later.

3.2 Hardware Description

The data acquisition systems described in this document are table top assemblies with all electronics confined to a single enclosure. One system has a total of ten circuit boards, the other six. In both instances, the circuit boards are designed to interface directly with a standardized BUS. The bus concept permits boards of similar construction, both electrical and physical, to be used within a single unit. Unfortunately, the buses of the two systems are themselves incompatible. Therefore the basic components, circuit boards, cannot be intermixed.

Most of the hardware elements in one system have a functional equivalent in the other (see Figures 2, 3, 4). The actual design details, however, are quite different. It wouldn't do much good to try and itemize all the minute differences as they are literally countless. The single most telling difference is, as mentioned in the opening paragraph, that the bus structures are not compatible. Specifically, the INTEL based system uses what INTEL calls its MULTIBUS.

The IMSAI based system conforms to the rules of the S-100 BUS. Both bus structures are categorized in Appendix B. Little additional mention of bus structures will be made throughout the remainder of the text. Refer to Chapter II for BUS discussion. It should be kept in mind that there is always an intra-system BUS distinction.

Now that the incompatibilities have been mentioned, it is worthwhile to note that in most respects the data acquisition and control tasks performed by each system are remarkably similar. Naturally, when viewed on a microscopic level numerous differences appear. However, when viewed as a single large entity it is difficult to perceive any appreciable differences in system operations. This is a tremendous asset. There is no reason, and in fact there has been no implementation, that emphasizes dissimilarity. All attempts to date have drawn on common features to build application systems that are functionally equivalent. Care has been taken to minimize the impact of hardware incompatibilities by writing carefully structured firmware. The Firmware Description Section (Section 3.3) has additional information on this topic.

The most important virtue of the structured programming technique (at least as far as it impacts on the available hardware) is that practically 100% of the non-initialization, non-device driver code is directly transferable between systems. This code sharing is made possible only because the microprocessors used in each system, although not exact

duplicates, are practically identical. The INTEL MDS-220/230 employs an 8080 processor while the IMSAI PCS-80/30 makes use of the newer and upwards compatible 8085 processor. All firmware modules are coded in 8080 Assembler. Therefore, when an update is successfully made to the applications program of one system the other stands ready to accept the modification also. The savings in time and effort for this method vs. "re-programming" of two systems can be enormous. In fact, in chronological order, the IMSAI based system was brought up first, an effort that lasted approximately four months. The time from which the INTEL system was received until it attained a comparable level of performance was a period of approximately two months. Realizing that with a new system it requires time to achieve parity on the learning curve (time to read and understand manuals) this time differential is rather remarkable.

Appendix A is a list of "essential" materials. Essential materials do not include specification of the console devices. Console devices are not included as part of the basic system since any device that interfaces with EIA-RS-232C will suffice. Although the mainframes are manufactured by INTEL and IMSAI, additional hardware has been acquired from a variety of vendors.

3.3 Firmware Description

The firmware program, named APPL1, for Application's Program 1, is a collection of modules which, when combined,

execute the real-time data acquisition and logging functions. As of this writing (August 3, 1978) these modules occupy approximately 2K bytes of memory. A byte is equivalent to 8 bits. All modules are coded in 8080 Assembler. Firmware Flow Diagrams are in Appendix C. The breakdown of the major modules is as follows:

1. A preliminary section of code, initialization, is executed only once (at start-up). This code "configures" the system. Specifically, control indicators are preset to their inactive positions. This is readily accomplished by clearing all the read/write memory that APPL1 uses for temporary storage. Since the adapted convention used throughout is that an indicator is inactive when reset to 0 (clear), this provides a convenient means to reset all indicators. Next, the hardware devices are commanded to operate in a preselected mode. This is readily accomplished by "programming" the hardware support elements. In essence, each hardware device is configured to operate in one specific mode chosen from a variety of programmable selections. The mode is chosen that best suits the requirements of the total hardware system and the remaining firmware. Examples of programmable devices are interrupt controllers, direct-memory-access (DMA) controllers, baud rate generators, USARTS, and the gain selector of the analog/digital converter sub-system. Last,

a sign-on message is prepared for output to each of the consoles. The message serves as a verification confirming correct system start-up and also supplies the current firmware version number.

2. The main program follows. This code continuously monitors the control indicators and initiates whatever action is appropriate. Control indicators are set when an operator issues a command through either of the operator consoles. Command entry and command processing will be discussed shortly. For now it suffices to note that there are means by which an operator can request that the system execute a desired function. These functions are wide ranging. Examples of such are commands that cause all sensor inputs to be scanned and have the corresponding data readings recorded on diskette for retrieval at a later time. Others can be more specific. For instance, one command requests the reading of a particular sensor. These operations (plus others) may be performed concurrently. In each case an indicator is set by the command processor informing the main program that a command request has been issued. The main program is charged with responding to said request. Section Four, Application One User's Guide itemizes all commands and command responses.

The main program, which forms the dispatch logic allowing commands to be executed (plus the

corresponding command entry code), is by necessity the most likely section of APPL1 to be expanded.

When new system functions are desired, it will involve additions to the main program expanding the dispatch code to include subprograms that perform functions required by the newly created feature.

Following this procedure allows for systematic expansion of features with minimal effort. Only rudimentary programming is required to expand the repertoire of system features. Subprograms handle most of the functions that any new feature could demand. By sequencing through these subprograms in an orderly fashion, new features can be added with minimal difficulty. This type of code organization is sometimes referred to as Structured or Top-Down Programming.

3. Interrupts are generated whenever an input or output character transfer is desired by either console device. An interrupt is also generated by an internal counter which is used as a time reference. All interrupt processing is contained in a special section of code, the Interrupt Handler. The role of the interrupt handler can be likened to that of an arbiter. It decides on the basis of need (priority) and current system status which devices may issue interrupts. An interrupt, once issued, causes all other processor action to be suspended for the duration of

the interrupt servicing time. Since the console device transfer rates are so slow (relative to processor throughput) it would be wasteful to use up processing time waiting for input/output to finish. Interrupts provide the means by which the processor can go about doing other business totally mindless of external device status. Then, only when a device is ready for a transfer will the processor be distracted from its primary duties and directed to service the device. Interrupt servicing can be thought of as a subordinate of the main program. The main program runs on the non-interrupt level and is allocated all system resources except at those times that an interrupt is being serviced. Two dissimilar operations are performed on the interrupt level.

Time of day, sometimes erroneously called realtime, is kept. A fixed frequency, periodic signal serves as the stimulus to a software clock. When a preset interval has elapsed, an interrupt is generated alerting the processor that it must update the internal time counters. On each occurrence of this interrupt the software clock subprogram increments the internal timer. This interrupt can be selected to occur at a variety of frequencies by virtue of the "programmable" hardware counter which serves to generate the interrupt signal. In one

system the interrupt is issued 20 times per second. It could be made to occur less frequently, if possible, as it is not necessary to maintain such exacting precision. External resolution is restricted to one second which corresponds to the interrupt frequency of the second system. An important feature of data acquisition is that a time tag is added to the data readings. Without time correlation the collected data is much less informative.

• Console device input/output transfers are directed on the interrupt level. All input characters are buffered until an end-of-line character (carriage return) is detected. The entire input line is passed on to the Operator Interactions section for command processing. Input mistakes may be corrected with special editing characters before processing. Output lines are likewise buffered. Output transfers can be initiated from such diverse sources as the command processors, periodic summary reports of data collection operations or fault detection. All output is treated identically by the output program regardless of the source. Characters are taken from the output buffer and transmitted continuously (under interrupt control) until an end-of-transmission character

(EOT) is detected. Upon detection of EOT the source that initiated the transfer is forced to surrender the channel. Another source may then follow with its own data to transmit.

4. Operator Interactions is a modular structure that processes the buffered input commands from a console device. If the line can be processed, Operation Interactions initiates the commanded function and directs an acknowledgement to the same console. Unintelligible commands or improper command parameter data is responded to with a negative acknowledge. As previously mentioned, it is the modular program structure that permits well managed feature expansion in APPL1. Operator Interactions is itself composed of many modules. A single module is charged with processing one and only one command. Commands are of the nature as to set or display the internal software clock, set or display the scan interval etc. Additional commands are used to initiate data collection, alter or monitor system hardware parameters (such as A/D gain selection) or start a new file on diskette. When new features are desired, it is most probable that at least one command must be added to support it. However, the command processors themselves call upon a lower level of general purpose support modules that handle most of the necessary chores. These lower level modules, which

are the lowest level in the entire firmware packages, are diverse enough to fill almost any conceivable need. Therefore new features can be added without the burden of coding large programs or adding more hardware.

5. Operator Command Supporting Subroutines are the bottom level modules. They perform such operations as code conversion from ASCII to BCD and BCD to binary, etc. (useful in command data entry and output report preparation), test the limits of a variable input parameters vs. a fixed reference (useful in validating command input parameter data) plus additional items of this sort (see Appendix C.) If a new command is added and cannot be made to fit the range of available supporting subroutine capabilities, itself an unlikely event, probably all that would be needed is that similar subroutines be added. Alternatively, minor modification of some of these modules may be in order. The modular structure of the existing APPL1 Code lends itself well to modification and addition. It is this type of building block structure that best serves the interests of future development.

CHAPTER IV

Application One User's Guide

4.1 APPL1 Start-Up

Application Program One, APPL1, is brought into memory from diskette. The procedure is as follows:

- turn-on power
- hit the reset button on the front panel. Monitor sign-on message will be displayed
- load the system diskette. Again hit reset. DOS sign-on message will be displayed. (If using IMSAI system load FORMATted scratch diskette in disk B, this diskette will be used for data files.)
- connect console device(s) and turn-on power
- start APPL1:
 - INTEL system enter APPL1.OBJ (carriage return)
 - IMSAI system enter APPL1 (carriage return)
 - APPL1 sign-on message will be output to console(s).

This completes the start-up procedure. If start-up is not successful, repeat all steps. If still unoperable refer to hardware maintenance manuals. (1,2)

NOTE: Double check diskette. Make certain that it contains the system files and APPL1.

4.2 Operator Commands and Responses

Figure 5 lists a summary of all commands and responses.

A detailed description follows:

<u>COMMAND</u>	<u>MNEMONIC</u>	<u>FORMAT</u>
SET TIME	ST	ST DAY/HR:MM:SS
DISPLAY TIME	DT	-----
SCAN INTERVAL	SI	SI XXXX ^[1]
DISPLAY INTERVAL	DI	-----
NORMALIZE	NI	NI y ^[2]
OPEN FILE	OF	-----
CLOSE FILE	CF	-----

NOTE1: 0001 ≤ XXXX ≤ 9999 seconds

NOTE2: Y = 1 or 2 or 4

SPECIAL EDIT CHARACTERS

- RUBOUT - delete last character
- ESCAPE - delete entire line
- LINE FEED - print input line
- CARRIAGE RETURN - terminate input

Figure 5. APPL1 Command Summary

- SET TIME (ST): used to initialize the internal timer. Time input is validated before acceptance. Correct time input is echoed in like format. Incorrect entry prompts a DATA ERROR response.
- DISPLAY TIME (DT): the compliment of ST. Response is in the same format as ST entry.
- SCAN INTERVAL (SI): used to install the time (measured in seconds) between scans. Input times are validated before acceptance. Correct entry is echoed in like format. Incorrect entry prompts a DATA ERROR response.
- DISPLAY INTERVAL (DI): the compliment of SI. Response is in the same format as SI entry.
- NORMALIZE (NI): scan data readings may be averaged (normalized) before recording. Default value is 1 (no averaging). Other choices are 2 to 4 in which case the average of 2 or 4 readings, respectively, is recorded.
- OPEN FILE (OF): used to start a new file called FILE.DAT on diskette. The IMSAI system uses disk B for this file, the INTEL unit uses the same diskette that contains APPL1.
- CLOSE FILE (CF): the compliment of OF. Terminates FILE.DAT.

NOTE: CLOSING is required. A diskette must not be removed without first CLOSING FILE.DAT.

4.3 Data Recording Format

Data records are currently of fixed format only. Future work may make use of variable length records if:

- data scanning becomes more selective. Some channels may be scanned more frequently than others.
- variable length records make better use of diskette capacity, i.e., less overhead per data record.
- analysis is faster or more efficient with variable length records.

As of this writing, only fixed length records of 128 bytes are used. They are usually easier to work with than variable length records and packing density doesn't appear to be a problem as yet. Figure 6 lists the items that compose a single record.

<u>BYTE</u>	<u>ITEM</u>	<u>DESCRIPTION</u>
0-2	FILE NUMBER	FILE.DAT EXTENSION
3	BLANK	
4-7	SEQUENCE NUMBER	INDIVIDUAL RECORD NUMBER
8	BLANK	
9-20	TIME	DAY/HR:MM:SS
21	BLANK	
22-31	UNASSIGNED	
32-127	SCAN DATA	64 CHANNEL CAPACITY 12 BITS DATA/CHANNEL EVEN CHANNEL: 8, 4 ODD CHANNEL : 4, 8

Figure 6. Fixed Record Format

CHAPTER V

Conclusion

The work that has been completed to date forms the foundation for an advanced data acquisition and control system capable of autonomous operation. It remains to be seen how effective the end system can be in actual day-to-day usage, but it is clear that it has tremendous potential. Most of the concepts of acquisition and control have been tested and verified. The feasibility of the proposed system has been validated. Additional work in the "analysis and control" area need to be performed before the system can be expected to produce the overall results sought, namely, autonomous operation.

Additional work yet to be performed includes:

- analysis and control algorithms must be developed and tested to operate in REAL TIME.
- data collection must be proven reliable both in quantity and quality (resolution). Without statistically good data it is impossible to procede to the final phase
- integration of the data collection, analysis and control sections into one complete closed-loop system.

Using the existing hardware plus the data acquisition firmware now available as a nucleus, the final goal can be met in the near future. There is considerable work yet to be done but fortunately much of the preliminary difficulties

have been cleared. Future development should proceed smoothly.

REFERENCES

1. "Intellec Series II Library", Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051. Manuals included in the library are:
 - Intellec Series II Installation & Service Manual, 9800559
 - A Guide to Microcomputer Development Systems, 9800558
 - ISIS-II User's Guide, 9800306
 - Intellec Series II Hardware Reference Manual, 9800556
 - Intellec Series II Schematic Drawings, 9800554
2. "Imsai Owner's Manuals", Imsai Manufacturing Corporation, San Leandro, California. Related owner's manuals are:
 - PCS-80/30 Reference Manual
 - PCS-80/30 User Manual
 - IMDOS User Manual
3. Supplementary Software Manuals, from Intel:
 - 8080/8085 Assembly Language Programming Manual, 9800301
 - Fortran-80 Programming Manual, 9800481
 - ISIS-II Fortran-80 Compiler Operator's Manual, 9800480
 - ISIS-II 8080/8085 Macro Assembler Operator's Manual, 9800292
 - 8080/8085 Floating-Point Arithmetic Library User's Manual, 9800452
4. Supplementary Hardware Manuals, from Intel:
 - Intellec Double Density Diskette Operating System Hardware Reference Manual, 98-422
 - Intellec Double Density Diskette Operating System Schematic Drawings, 9800425
5. Supplementary Imsai Publications, from Imsai:
 - MIO User Manual
 - DIO Controller Reference Manual
 - DIO Controller User Manual
 - Persci Model 277 Diskette Drive Maintenance Manual
 - Imsai IKB-1 Intelligent Keyboard
 - Imsai Fortran User's Manual

REFERENCES (cont'd.)

6. Related Intel System Publications, from vendors:
 - Instruction Manual for ADAC Corporation Model 735-SBC Data Acquisition and Control System
 - Burr-Brown Research Corporation Microcomputer Digital Output System Model MP801 Preliminary Specifications
 - Ball Brothers Research Corporation Data Display Monitor Model TV-50/90/120 Service Manual
 - Shugart Associates Diskette Storage Drive Model SA800/801 Maintenance Manual
7. Related Imsai System Publications, from vendors:
 - Altair 8800 88-Process Control Interface Board Documentation
 - Altair 8800 Analog Conversion System Technical Manual
8. Related Intel and Imsai System Publications, from Intel:
 - MCS-80 User's Manual
 - MCS-85 User's Manual
9. Rolander, T., "Intel Multibus Interfacing", Application Note AP-28, \$1, Intel Corporation.
10. "EDN μ C Design Series - Bus Standardization", EDN, February 20, 1978, pps. 97-103.
11. "Electronic Design Technology - Reduce Your μ C-Bases System Design Time...", Electronic Design 3, February 1, 1978, pps. 56-66.
12. "Electronic Design Technology - Simplify Analog/Computer Interfacing" Electronic Design 17, August 16, 1977, pps. 92-99.
13. "Electronic Design Technology - Modern Data Acquisition is Complex", Electronic Design 22, October 25, 1977, pps. 60-66.

Appendix A

List of Materials

MULTIBUS System List of Materials (1,4,6)

INTEL Corporation
Santa Clara, California

- MDS-220 Inteltec Series II Model 220 Microcomputer Development System. Package includes CRT chassis with power supply, fans, cables, 6-slot cardcage, keyboard and single density floppy disk drive. System provides 32K RAM, 4K ROM, MULTIBUS with multiprocessor and DMA capability, interfaces for two serial ports, a high speed paper tape reader/punch, PROM burner and a line printer.
- MDS-230 Same as MDS-220 except single density floppy disk drive replaced by two double density drives and controller (requires 2 card slots).
- MDS-032 32K dynamic RAM memory.

ADAC Corporation
Woburn, Massachusetts

- SBC-735 Data Acquisition System (DAS)

Burr-Brown Research Corporation
Tuscon, Arizona

- MP801 16 channel relay output system (digital controller)

S-100 BUS System List of Materials (2,5)IMSAI Manufacturing Corporation
San Leandro, California

- PCS-80/30 Table top version of the basic computer system with 5 inch CRT. Includes MPU-8 processor, PS-28 amp power supply, V10-C video controller, IRB-1 intelligent keyboard plus necessary cables. 10 card capacity.
- PCS-80/25B Dual PerSci floppy disk drive, double density format with power supply in table top cabinet. Includes DIO disk interface (requires 2 slots) and necessary cables.
- MIO Multiple I/O Board (two parallel ports, one serial port, one control port and Tarbell tape cassette interface).

MITS
Albuquerque, New Mexico

- ALTAIR 8800 Analog Conversion System (DAS). Includes 88-ADC analog to digital converter and 88-MUX multiplier plus necessary cables.
- ALTAIR 8800 Process Control Interface (digital controller)

Artec Electronics, Inc.

- 32K-100 32K S-100 BUS Compatible static ram card

Appendix B

Bus Structures

B.1

S-100 SYSTEM BUS STRUCTURE

The S-100 system bus structure consists of 100 lines. These are arranged 50 on each side of the plug-in boards.

The following general rules apply to the S-100 system bus:

SYMBOLS: "P" prefix indicates a processor command/
control signal

"S" prefix indicates a processor status
signal

LOADING: All inputs to a card will be loaded with
a maximum of 1 TTL low power load.

LEVELS: All bus signals except the power supply
are TTL

S-100 Bus Pin Assignments

<u>PIN</u>	<u>MNEMONIC</u>	<u>DESCRIPTION</u>
1	+8V	Unregulated input to 5V regulators
2	+16V	Positive unregulated voltage
3	XRDY	For special applications: Pulling this line low will cause the processor to enter a WAIT state and allows the status of the normal Ready line (PRDY) to be examined
4	VIO	Vectored Interrupt Line #0
5	VI1	Vectored Interrupt Line #1
6	VI2	Vectored Interrupt Line #2
7	VI3	Vectored Interrupt Line #3
8	VI4	Vectored Interrupt Line #4
9	VI5	Vectored Interrupt Line #5
10	VI6	Vectored Interrupt Line #6
11	VI7	Vectored Interrupt Line #7
12 to 17	To be defined	
18	STA DSB/	Allows the buffers for the 8 status lines to be tri-state
19	C/C DSB/	Allows the buffers for the 6 output command/control lines to be tri-stated
20	UNPROT	Input to the memory protect flip-flop on a given memory board
21	SS/	Indicates that the machine is in the process of performing a single step
22	ADD DSB/	Allows the buffers for the 16 address lines to be tri-stated

23	DO DSB/	Allows the buffers for the 8 data output lines to be tri-state
24	Ø1	Phase 2 Clock
25	Ø2	Phase 1 Clock
26	PHLDA	Processor command/control output signal which appears in response to the HOLD signal; indicates that the data and address bus will go to the high impedance state
27	PWAIT	Processor command/control output signal which acknowledges that the processor is in a WAIT state
28	PINTE	Processor command/control output signal indicating interrupts are enabled: indicates the content of the CPU internal interrupt flip-flop; F-F may be set or reset by EI and DI instruction and inhibits interrupts from being accepted by the CPU if it is reset
29	A5	Address Line #5
30	A4	Address Line #4
31	A3	Address Line #3
32	A15	Address Line #15
33	A12	Address Line #12
34	A9	Address Line #9
35	D01	Data Out Line #1
36	D00	Data Out Line #0
37	A10	Address Line #10
38	D04	Data Out Line #4
39	D05	Data Out Line #5
40	D06	Data Out Line #6
41	DI2	Data In Line #2

42	DI3	Data In Line #3
43	DI7	Data In Line #7
44	SM1	Status output signal that indicates that the processor is in the fetch cycle for the first byte of an instruction
45	SOUT	Status output signal which indicates that the address bus contains the address of an output device and the data bus will contain the output data when PWR is active
46	SINP	Status output signal which indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when PDBIN is active
47	SMEMR	Status output signal which indicates that the data bus will be used for memory read data
48	SHLTA	Status output signal which acknowledges a HALT instruction
49	CLOCK?	Inverted output of the 2MHz oscillator that generates the 2 phase clock
50	GND	Signal ground
51	+8V	Unregulated input to 5V regulators
52	-16V	Negative unregulated voltage
53	SSW DSB/	Disables the data input buffers so the input from the sense switches may be strobed onto the bidirectional data bus right at the processor
54	EXT CLR/	Clear signal for I/O devices (front panel switch closure to ground)
55 to 67	To be defined	

68	MWRT	Indicates that the current data on the Data Out Bus is to be written into the memory location currently on the address bus
69	PS/	Indicates the status of the memory protect flip-flop on the memory board currently addressed
70	PROT	Input to the memory protect flip-flop on the memory board currently addressed
71	RUN	Indicates that the RUN/STOP flip-flop is Reset
72	PRDY	Processor command/control input that controls the run state of the processor; if the line is pulled low the processor will enter a wait state until the line is released
73	PINT/	The processor recognizes an interrupt request on this line at the end of the current instruction or while halted. If the processor is in the HOLD state or the Interrupt Enable flip-flop is reset, it will not honor the request
74	PHOLD/	Processor command/control input signal which requests the processor to enter the HOLD state; allows an external device to gain control of address and data buses as soon as the processor has completed its use of these buses for the current machine cycle
75	PRESET/	Processor command/control input; while activated the content of the program counter is cleared and the instruction register is set to 0
76	PSYNC	Processor command/control output provides a signal to indicate the beginning of each machine cycle
77	PWR/	Processor command/control output used for memory write or I/O output control: data on the data bus is stable while the $\overline{PWR}$ is active

78	PDBIN	Processor command/control output signal indicates to external circuits that the data bus is in the input mode
79	A0	Address Line #0
80	A1	Address Line #1
81	A2	Address Line #2
82	A6	Address Line #6
83	A7	Address Line #7
84	A8	Address Line #8
85	A13	Address Line #13
86	A14	Address Line #14
87	A11	Address Line #11
88	D02	Data Out Line #2
89	D03	Data Out Line #3
90	D07	Data Out Line #7
91	DI4	Data In Line #4
92	DI5	Data In Line #5
93	DI6	Data In Line #6
94	DI1	Data In Line #1
95	DI0	Data In Line #0
96	SINTA	Status output signal to acknowledge signal for INTERRUPT request
97	SWO	Status output signal indicates that the operation in the current machine cycle will be a WRITE memory or output function
98	SSTACK	Status output signal indicates that the address bus holds the pushdown stack address from the Stack Pointer
99	POC/	Power-On Clear
100	GND	GROUND

B.2

INTEL MULTIBUS (9)

The INTEL MULTIBUS includes the following signal lines:

- 16 three-state address lines
- 16 bidirectional three-state data lines
- 8 open-collector multi-level interrupt lines
- 17 active-low bus control lines
- 24 power supply or signal-ground lines

Modules that use the MULTIBUS have a master-slave relationship. A bus master can drive the control and address lines, i.e., command the bus. A Single Board Computer (SBC) is an example of a bus master. Memory and I/O expansion boards are examples of bus slaves.

A system may have a number of bus masters. Bus arbitration is needed when more than one master requests control of the bus at the same time. Usually, the bus clock is provided by one of the masters and is derived independently from the processor clock (of the same master). The bus clock provides a timing reference for resolving bus contention for multiple requests from bus masters. Actual transfers via the bus proceed asynchronously with respect to the bus clock.

Multibus Pin Assignments*

BOARD COMPONENT SIDE

	<u>PIN</u>	<u>MNEMONIC</u>	<u>DESCRIPTION</u>
Power Supplies	1	GND	Signal Ground
	3	+5	+VDC
	5	+5	+5 VDC
	7	+12	+12 VDC
	9	-5	-5 VDC**
	11	GND	Signal Ground
Bus Controls	13	BCLK/	Bus Clock
	15	BPRN/	Bus Priority In
	17	BUSY/	Bus Busy
	19	MRDC/	Memory Read Command
	21	IORC/	I/O Read Command
	23	XACK/	Transfer Acknowledge
	25	AACK/	Advanced Acknowledge
	27	BHREN/	Byte High Enable
Interrupts	29		Reserved
	31	CCLK/	Constant Clock
	33	INTA/	Interrupt Acknowledge
	35	INT6/	Interrupt Requests
	37	INT4/	
	39	INT2/	
	41	INT0/	
Address	43	ADRE/	Address Lines
	45	ADRC/	
	47	ADRA/	
	49	ADR8/	
	51	ADR6/	
	53	ADR4/	
	55	ADR2/	
	57	ADRO/	
Data	59	DATE/	Data Lines
	61	DATC/	
	63	DATA/	
	65	DAT8/	
	67	DAT6/	
	69	DAT4/	
	71	DAT2/	
	73	DAT0/	

	75	GND	Signal Ground
	77	-10	-10 VDC
Power	79	-12	-12 VDC
Supplies	81	+5	+5 VDC
	83	+5	+5 VDC
	85	GND	Signal Ground

BOARD CIRCUIT SIDE

	<u>PIN</u>	<u>MNEMONIC</u>	<u>DESCRIPTION</u>
	2	GND	Signal Ground
	4	+5	+5 VDC
Power	6	+5	+5 VDC
Supplies	8	+12	+12 VDC
	10	-5	-5 VDC**
	12	GND	Signal Ground
	14	INIT/	Initialize
	16	BPRO/	Bus Priority Out
Bus	18	BREQ/	Bus Request
Controls	20	MWTC/	Memory Write Command
	22	IOWC/	I/O Write Command
	24	INH1/	Inhibit (disable) RAM
	26	INH2/	Inhibit (disable) ROM or ROM
	28	ADR10/	
	30	ADR11/	Address
	32	ADR12/	Extension
	34	ADR13/	Lines
	36	INT7/	
Interrupts	38	INT5/	
	40	INT3/	Interrupt Requests
	42	INT1/	
	44	ADRF/	
	46	ADRD/	
	48	ADRB/	
Address	50	ADR9/	Address Lines
	52	ADR7/	
	54	ADR5/	
	56	ADR3/	
	58	ADR1/	
	60	DATF/	
	62	DATD/	
	64	DATE/	
Data	66	DAT9/	Data Lines
	68	DAT7/	
	70	DAT5/	
	72	DAT3/	
	74	DAT1/	

	76	GND	Signal Ground
	78	-10	-10 VDC
Power	80	-12	-12 VDC
Supplies	82	+5	+5 VDC
	84	+5	+5 VDC
	86	GND	Signal Ground

*A slash character (/) after the mnemonic indicates signal is active-low

**Not required for most Intellec Series II configurations

APPENDIX C

FIRMWARE FLOW DIAGRAMS

PROGRAM NAME: APPL1

SOURCE LANGUAGE: 8080 ASSEMBLER

DEVELOPMENT SYSTEM: INTEL ISIS-II

DATE: August 1978

INITIALIZATION

INITIALIZATION (cont'd.)

-
- ```
graph TD; A[INITIALIZATION (cont'd.)] --> B["- set programmable gain in A/D processor
- set internal time : 001/00:00:00
DAY/HR:MM:SS"]; B --> C["SIGN-ON
Prepare sign-on messages for console devices:
\"APPL1 VER1.N\""]; C --> D["- enable interrupt system
[this completes initialization]"]; D --> E["LOOP (next page)"]; E --> F["END OF INIT"];
```
- set programmable gain in A/D processor
  - set internal time : 001/00:00:00  
DAY/HR:MM:SS

SIGN-ON

Prepare sign-on messages for console devices:  
"APPL1 VER1.N"

- enable interrupt system  
[this completes initialization]

LOOP (next page)

END OF INIT

## MAIN PROGRAM

## DATA COLLECTION

## DATA COLLECTION (cont'd.)

## DATA LOGGING

## INTERRUPT SERVICE

| <u>IR</u> |                                  | <u>Page</u> |
|-----------|----------------------------------|-------------|
| 0         | RXR1: TTY receive ready          | 60          |
| 1         | TXR1: TTY transmit empty         | 62          |
| 2         | RXR2: MODEM receive ready        | 63          |
| 3         | TXR2: MODEM transmit empty       | 63          |
| 4         | RTC: Real Time Clock             | 64          |
| 5         | IERROR: not used-error condition |             |
| 6         | IERROR: not used-error condition |             |
| 7         | IERROR: not used-error condition |             |

## INTERRUPT SERVICE (cont'd.)

```
graph TD; IEXIT((IEXIT)) --> Steps[- acknowledge end of interrupt (EOI) to hardware interrupt controllers (8529)
- restore pre-interrupt processor status from STACK]; Steps --> RETURN[RETURN
(resume non-interrupt level)];
```

IEXIT

INTERRUPT EXIT
(from any of the above
service routines)

- acknowledge end of interrupt (EOI) to hardware interrupt controllers (8529)
- restore pre-interrupt processor status from STACK

RETURN

(resume non-interrupt level)

LOCAL CONSOLE RECEIVE INTERRUPT

LOCAL CONSOLE RECEIVE INTERRUPT (cont'd.)

LOCAL CONSOLE TRANSMIT INTERRUPT

REMOTE CONSOLE RECEIVE AND TRANSMIT INTERRUPT

RXR2

REMOTE CONSOLE RECEIVE
INTERRUPT

TXR2

REMOTE CONSOLE TRANSMIT
INTERRUPT

The "REMOTE" CONSOLES operate IDENTICALLY
to the "LOCAL" CONSOLES

1. all coding conventions are duplicated
2. all input is treated indentially
3. either or both may be in operation at
any time

IEXIT

REAL TIME CLOCK (DAY/HR:MM:SS)

REAL TIME CLOCK (cont'd.)
(DAY/HR:MM:SS)

OPERATING INTERACTIONS

OPERATOR INTERACTIONS (cont'd.)

OPCMMMD:SET TIME-DAY/HR:MM:SS

OPCMMD:SET TIME-DAY/HR:MM:SS (cont'd.)

OPCMMMD:DISPLAY TIME

OPCMD:DISPLAY TIME (cont'd.)

- restore registers to original content

CLSBUF
(page 67)

OPCMMD:OPENFILE-START NEW FILE

OPCMMD:CLOSFILE-CLOSE CURRENT FILE

ACCEPT
(page 67)

OPERATOR COMMAND SUPPORT SUBROUTINES

Note: The "chart" on the next three pages is just one of the many Operator Command Support Subroutines

I. DIGXX

(note: all DIGXX subroutines convert from ASCII coding to BCD coding)

ENTRY POINTS:

1. DIG4S: extract 4 BCD digits plus SLASH
2. DIG4 : extract 4 BCD digits
3. DIG3S: extract 3 BCD digits plus SLASH
4. DIG2C: extract 2 BCD digits plus COLON
5. DIG2 : extract 2 BCD digits
6. DIGIT: extract 1 BCD digits

OPERATOR COMMAND SUPPORT SUBROUTINES (cont'd.)

not shown

DIG3S: similar to DIG4S
DIG2C: similar to DIG2

OPERATOR COMMAND SUPPORT SUBROUTINES (cont'd.)

Other OPERATOR COMMAND SUPPORTING SUBROUTINES ARE:

II BCDAXX - convert BCD to ASCII Coding
III LIMITW - test 16 bit limits
IV LIMITB - test 8 bit limits
V SLASH - test character in input string = SLASH
VI COLON - test character in input string = COLON
VII WSPACE - write a SPACE character to output buffer
VIII WSLASH - write a SLASH character to output buffer
IX WCOLON - write a COLON character to output buffer
X PUTC - write any ASCII character to output buffer