NASA
Technical
Paper
2378

January 1985

MINDS

A Microcomputer Interactive Data System for 8086-Based Controllers

James F. Soeder

NASA

NASA Technical Paper 2378

1985

MINDS

A Microcomputer Interactive Data System for 8086-Based Controllers

James F. Soeder

Lewis Research Center
Cleveland, Ohio

NASA
National Aeronautics and Space Administration

Scientific and Technical Information Branch

Summary

A microcomputer interactive data system (MINDS) software package for the 8086 family of microcomputers is described. To enhance program understandability and ease of code maintenance, the software is written in PL/M-86, Intel Corporation's high-level system implementation language. The MINDS software is intended to run in residence with real-time digital control software to provide displays of steady-state and transient data. In addition, the MINDS package provides classic monitor capabilities along with extended provisions for debugging an executing control system. The software uses the CP/M-86 operating system developed by Digital Research, Inc., to provide program load capabilities along with a uniform file structure for data and table storage. A library of input and output subroutines to be used with consoles equipped with PL/M-86 and assembly language is described.

Introduction

As microprocessors have become more sophisticated, they have begun to take over applications formerly reserved for more costly minicomputers and low-end mainframe computers. This evolution has resulted in opportunities to apply digital computer technology to applications that would never have been considered a few years ago. The result of this explosion of computer applications has been an enormous demand for software to accomplish these many and varied tasks. This demand has led to what has been called the software crisis (ref. 1). This crisis has come about because there are too many applications requiring sophisticated software with too few programmers to write it. The problem has been somewhat alleviated in the business software environment by the adoption of standard operating systems (e.g., Digital Research's CP/M (ref. 2) and structured high-order programming languages such as Pascal.

In the application of microprocessors to real-time digital control systems, however, each application and hence each algorithm is unique. This means specialized software to do each task. This problem can be alleviated to some extent by standardizing, wherever possible, through the use of standard software modules as well as complete standard software systems. Even then additional means of improving software development efficiency will be required. This is especially true for control software development since the software must run in real time usually in a timed or interrupt-driven environment. This report describes one particular method for enhancing the debugging and evaluation of real-time control software. The approach employs an auxiliary software routine that operates in conjunction with the control software. This powerful software utility is called MINDS for "microcomputer interactive data system." It is designed to use spare unused computing time in between control computation update intervals. It allows an operator at a keyboard to interactively extract data from the computer's memory in several useful formats. These data can be used for software debugging or control software performance analysis.

This report gives an overview of the software package, describes the hardware and software environment necessary to use the program, details the internal structure of each of the program's sections, describes the currently implemented commands, and gives the extensions in scope and application that the current software makes possible.

Overview

The MINDS software utility is designed to run on a microcomputer in residence with a real-time direct digital control algorithm. To understand the philosophy on which MINDS is based, one must understand a simple, direct digital control application. A timing diagram for this type of task is shown in figure 1. In this application an interval timer generates an interrupt at a specified rate (determined by the bandwidth requirements of the system). When the central processing unit receives the interrupt, it samples sensor input signals with an analog-to-digital (A/D) converter. The CPU then uses these input signals to compute a control algorithm. When the control algorithm computation is complete, the results are output to control actuators with digital-to-analog (D/A) converters. Once this has been done, the computer is idle until the next sampling interrupt, at which time the process starts over again. When the CPU is idle, information can be input to and output from the CPU to

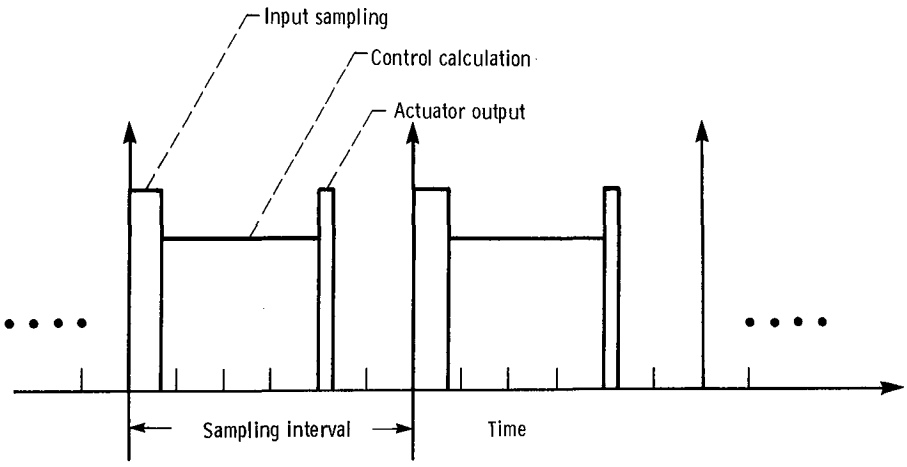

Figure 1.—Direct digital control timing diagram.

provide an operator interface and data-taking capability, etc., without disturbing the control process that takes place at each update interval. The MINDS software package to be described in this report gives exactly this capability. It provides a modular software package that can be called by a control algorithm's real-time executive routine to run in the spare time while the control algorithm is waiting for the next update interval timer interrupt.

MINDS is intended to provide both steady-state and transient data for variables inside and outside the control algorithm. This is accomplished by allowing an operator of the control equipment to assign names and other designators to storage locations in the control computer's memory, thereby creating a MINDS data element. These data elements can then be displayed separately or in tabular form to provide a representation of the control's steady-state condition. In addition, time histories of these data elements can be stored in the computer's unused memory, retrieved, and put on disks for later data analysis or plotting. MINDS also has a debug package that can set conditional breakpoints inside an executing real-time control and capture steady-state and register data when a predefined condition is satisfied.

Interactive data-taking software such as MINDS has been developed previously. One example would be INFORM (ref. 3), which was developed for the SEL 810B minicomputer. This program, although very powerful and heavily used, lacked certain features. First, it could not handle data elements that were real floating-point numbers. Second, since it did not interface with an operating system, it lacked the capability for memory management and standard file storage format. Third, it lacked buffered input, thereby making it difficult to input commands. Finally, the program was written in assembly language for a computer system produced in limited quantities and thus had little industry support and interest.

The MINDS data-taking software has attempted to address these deficiencies. First, MINDS is designed for a system based on a popular microprocessor, the Intel 8086/8087. Second, it uses the widely accepted CP/M-86 operating system to provide memory as well as file management. Finally, most of the MINDS software is written in PL/M-86 (ref. 4), a microprocessor version of the PL/1 higher level language, to aid in the programs understandability and transportability.

Support Hardware

The microcomputer hardware on which MINDS has been designed to operate is discussed in some detail. This is important since various aspects of the hardware (such as segmented memory addressing) affect the decisions made in the MINDS software design. The elements used to implement the MINDS software package include a single-board computer (Intel iSBC 86/12A) that incorporates the 8086/8087 microprocessor pair, a disk controller with floppy disks, and the CP/M-86 disk operating system. Each of these elements is discussed in the following sections.

8086 Microprocessor

The 8086 microprocessor is the first member in the family of Intel 16-bit microprocessors. The processor has a 16-bit data bus and a 20-bit address, resulting in a 1-megabyte memory address space. Figure 2 is a register diagram (i.e., assembly language programmers model) of the 8086. The processor contains thirteen 16-bit registers. These registers include general-purpose registers AX and DX, base pointer registers BX and BP, index registers DI and SI, counter register CX, instruction pointer IP, stack pointer SP, and segment registers ES, SS, DS, and CS. The four segment registers are critical in allowing the

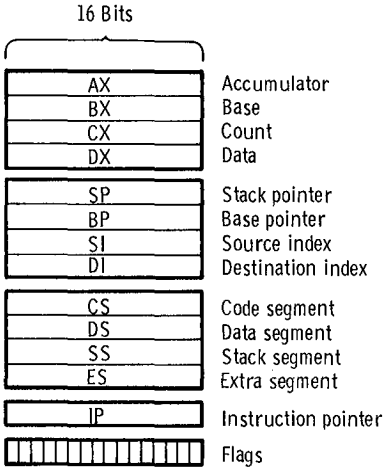

Figure 2.—Intel 8086 register structure.

8086 to achieve its full 1-megabyte address space. If the instruction pointer (a 16-bit register) alone is used to compute memory locations, only 2^{16} or 65 536 memory locations can be addressed. However, if memory addresses are computed by using both the instruction pointer and a segment register, the full 1-megabyte address space can be accessed by using the formula

$$(\text{Segment} * 10_H) + \text{IP} = \text{Memory location}$$

where H denotes a hexadecimal number. Furthermore, once the segment register has been specified, any location that lies between the segment value and the segment value plus 65 535 can be addressed directly by merely changing the value of the instruction pointer. An illustration of this is given in figure 3. The result of this type of addressing, however, is that to identify the location of any parameter in memory, one must specify not only the instruction pointer (i.e., the offset) but also the segment. Further information on this addressing scheme and the 8086 architecture in general can be found in reference 5.

The 8087 microprocessor chip is a transparent co-processor that can be added to the 8086 to augment the instruction set and the architecture. This chip adds eight 80-bit-wide registers to the 8086 structure defined previously, along with a status and mode register. This register augmentation is shown in figure 4. The coprocessor augments the 8086 instruction set with instructions that use the 80-bit-wide registers to do floating-point arithmetic, trigonometric functions, and logarithms. These calculations are done in accordance with the proposed IEEE floating-point standard. Further information on the 8087 coprocessor chip can be found in reference 6.

iSBC 86/12A Microcomputer

The iSBC 86/12A single-board computer (fig. 5), which operates the MINDS software, has a 5-MHz

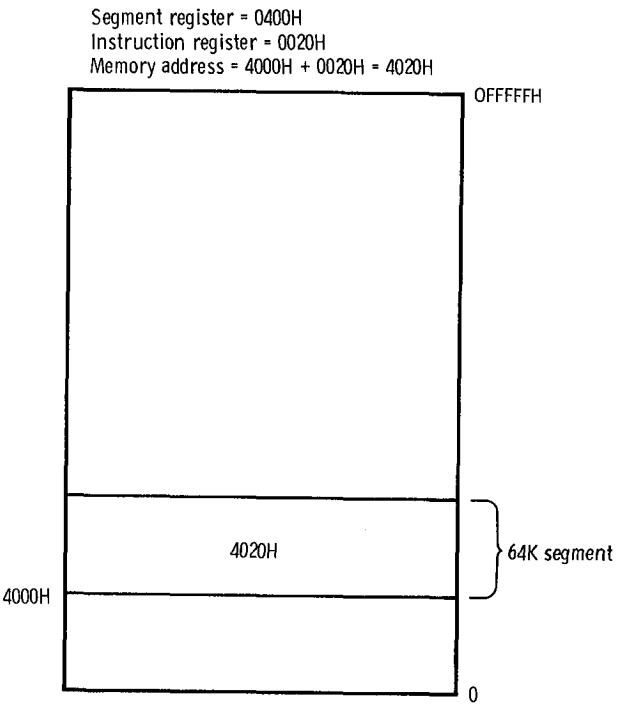

Figure 3.—8086 Memory addressing.

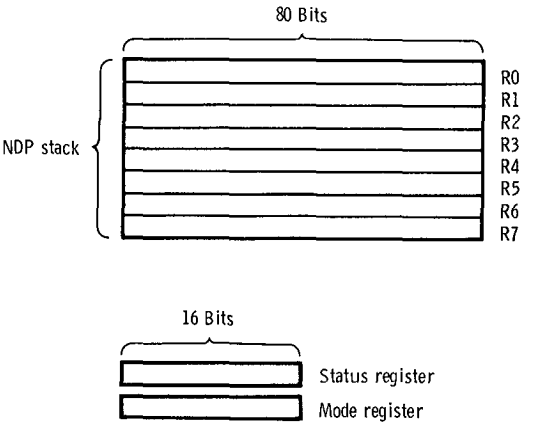

Figure 4.—Intel 8087 register structure.

8086/8087 microprocessor pair as its central processing element. In addition, the board contains 32 kilobytes of dynamic random-access memory (RAM), 32 kilobytes of expansion RAM, and 32 kilobytes of erasable, programmable read-only memory (EPROM). The board also has 24 parallel input/output lines for printer interfacing, a RS232 compatible serial input/output port used to interface to a cathode-ray tube, two programmable counter/timers for control and sample timing, and an interrupt controller capable of accepting eight external or internal interrupts. The board can accept an 8087 (iSBC 337) numerics coprocessor to augment the 8086 with floating-point numerics capability. Lastly, the 86/12A is Multibus compatible. The Multibus/IEEE 796 is a standard microprocessor backplane interface bus originally developed by Intel. Using this standard set of

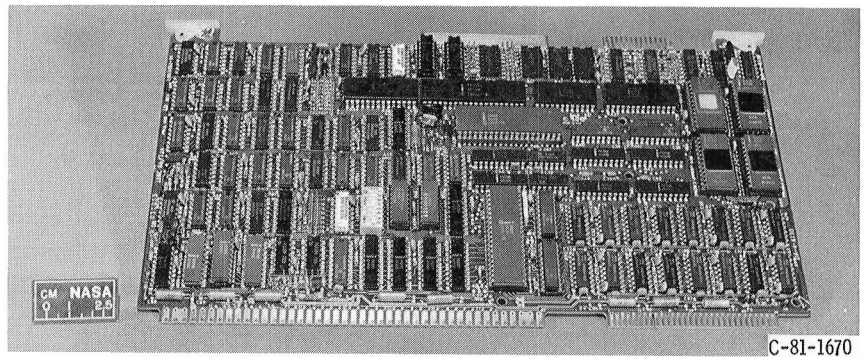

Figure 5.—iSBC 86/12A single-board computer.

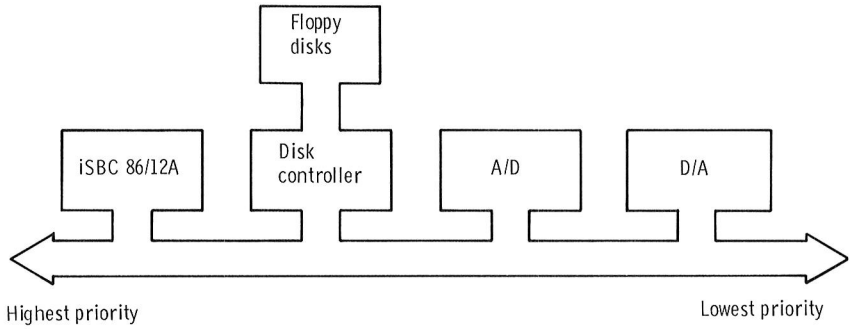

Figure 6.—MINDS hardware configuration.

address, data, power, and control lines facilitates the interface to a multitude of standard boards manufactured by a variety of vendors. Further information on the 86/12A and the Multibus can be found in references 7 and 8.

Successful operation of the MINDS package in a real-time digital control environment requires, in addition to the 86/12A board, Multibus boards of the following type: an analog-to-digital (A/D) converter board and a digital-to-analog (D/A) converter board. Figure 6, which illustrates this configuration in more detail, shows the single-board computer in the highest priority slot of the multibus. This is necessary so that under all bus conditions the computer can get access to the bus to use the A/D and D/A converters and thereby update the control law in the specified sampler network. As was stated previously, the MINDS package has been designed to use the disk controller during the control's spare time in making data and table transfers. Therefore, if the disk controller cannot be interrupted after every bus cycle, transparent operation with the direct control algorithm cannot be guaranteed. Further implementation information on this hardware configuration can be found in reference 9 and appendix A.

CP/M-86 Disk Operating System

As stated earlier, the MINDS program has been designed to use a permanent storage medium (floppy disk) to save data extracted from the computer's memory. The disk operating system with which MINDS has been designed to operate is CP/M-86.

The CP/M-86 operating system is a general-purpose, single-user operating system marketed by Digital Research, Inc., of Pacific Grove, California. The operating system provides facilities to do console communications, program load and unload, disk file management, and rudimentary computer memory management. A memory map of the 86/12A computer with the CP/M-86 operating system installed (fig. 7) shows that the

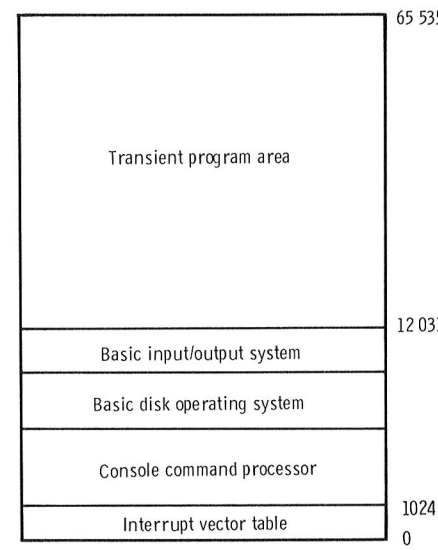

Figure 7.—iSBC 86/12A memory configuration.

CP/M-86 is located in the lowest part of memory and occupies approximately 12 kilobytes. The remaining 52 kilobytes is used as a transient program area for applications programs and an interrupt vector table for operating system and application use.

The operating system has three main portions:

(1) Console command processor (CCP)
(2) Basic disk operating system (BDOS)
(3) Basic input/output system (BIOS)

The CCP accepts all commands that are typed in from the console, interprets them, and takes the appropriate action to see that they are completed. The BDOS contains the facilities to open, format, read, and organize files on a floppy disk. In addition, it contains routines to do memory management (i.e., the reserving and releasing of memory) for the free or transient program area. The BDOS has an external entry point or "hook" such that most file and memory management functions can be accessed by applications programs. MINDS makes extensive use of this hook to provide an orderly format for data storage and retrieval.

The BIOS contains all of the hardware-dependent information necessary to allow CP/M-86 to operate in a particular computer configuration. This includes definition of the disk layout (i.e., number of tracks, number of sectors, number of bytes per sector, etc.), the address and input/output format of the console device and line printer, and the memory configuration of the transient program area. Because all of the hardware-dependent information is concentrated in one area, CP/M-86 can be easily reconfigured for a variety of hardware environments.

How the operating system loads programs is important in understanding MINDS. When the CP/M-86 operating system is given a command, the CCP processes it and determines if an external applications program must be loaded (i.e., a program on the disk). If this is the case, the program is loaded from the disk and placed in the topmost part of memory (i.e., the highest memory location). The free memory above the operating system and below the applications program is still available to be allocated (managed) by CP/M-86 for the applications program's use. Further information on CP/M-86 and its applications environment can be found in reference 10.

Software Overview

As stated earlier, CP/M-86 is a single-user operating system. It does not do multitasking. A real-time application such as direct digital control, however, requires that multiple tasks be performed. Thus, a real-time application using CP/M-86 must have a subexecutive, or real-time executive, routine to service interval timers, interrupt controllers, and A/D and D/A converters and to execute a control algorithm and MINDS. This is a straightforward software task in direct digital control, and for the system described in this report it requires less than 0.6 kilobyte of memory.

A diagram of the software hierarchy between CP/M-86, MINDS, and the real-time control (fig. 8) shows that a real-time executive is directly linked to the control algorithm and the MINDS software. This is represented by the solid lines connecting the three modules. This combined software module is loaded into the microcomputer main memory by using the CP/M-86 operating system, as shown by the dashed-line interconnection. In this environment the MINDS subroutines can provide a general-purpose operator interface, steady-state transient data collection, and a monitor debug package. Finally, as represented by the wide-arrow interconnection, the MINDS software can use the services provided by CP/M-86 to assist in orderly memory management and can read and write disk files of a standard configuration.

The MINDS software can be broken into four logical blocks (fig. 9). Each of these blocks consists of a number of subroutines. The important subroutines contained in each block are listed in appendix B. The MINDS main block takes care of variable name definitions, steady-state data table definitions, information storage and retrieval, and other miscellaneous tasks. The monitor and debug block can display and set any part of main memory as an integer, word, byte, or real number. In addition, the monitor functions as a software debug tool and sets a breakpoint in order to collect data tables and register data if certain conditions are met. The MINDS transient data block provides all that is needed to take and store transient data.

Finally, the input/output library contains the programs necessary to input and output information to the

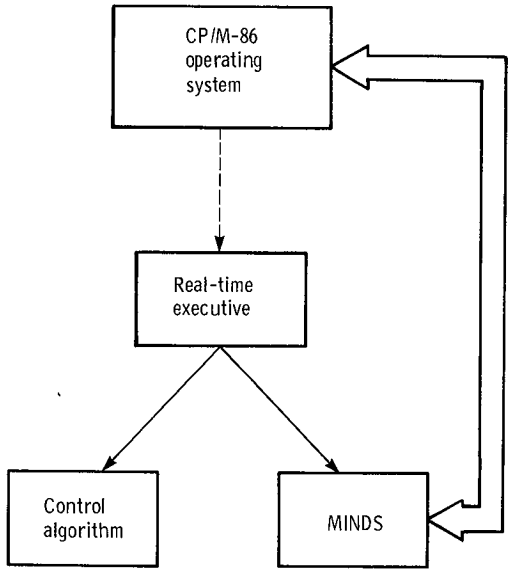

Figure 8.—Software hierarchy.

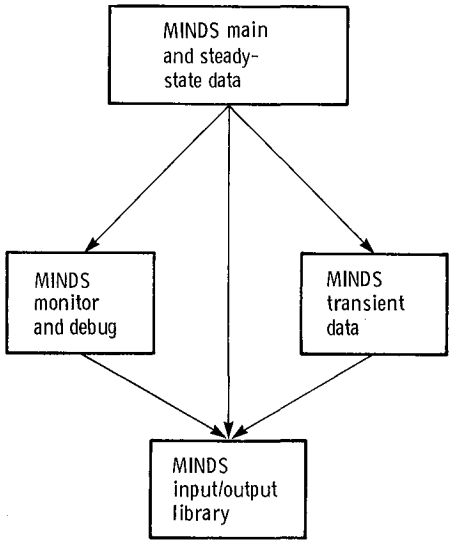

Figure 9.—MINDS package interconnection.

console and printer. In addition, it contains a series of assembly language routines to interface to CP/M-86. This type of library is necessary since PL/M-86 is a system implementation language and merely has an instruction to input or output a word or byte from or to a particular port. The language does not have a sophisticated input/output structure like Fortran or Pascal.

A detailed discussion of the workings and operation of each of these four large blocks of code, or software modules, follows.

Main Software Module

Data Element Definition

Before a discussion can begin on the operation of the main MINDS program, the variables and control structures must be defined. The MINDS software relies on the manipulation of data elements. A data element is simply a memory location or a series of memory locations that have been given a prescribed set of attributes. Data elements can be any one of three types (fig. 10): word, integer, or real. Every addressable location in the 8086 memory is eight bits, or one byte.

The figure shows that a word is an unsigned 16-bit piece of data, or two bytes of data. This data type can take on values of 0 to 65 535, (i.e., $2^{16}-1$). An integer is a signed 16-bit piece of data (two bytes) that can take on both positive and negative values since bit 15 is the two's-complement sign bit. Because of this the range of the number is $-32\,768$ to 32 767 (i.e., -2^{15} to $2^{15}-1$). Finally, a real number is a signed four-byte data type, consisting of a mantissa and exponent, that can range in value from 8.43×10^{-37} to 3.37×10^{38} (ref. 6).

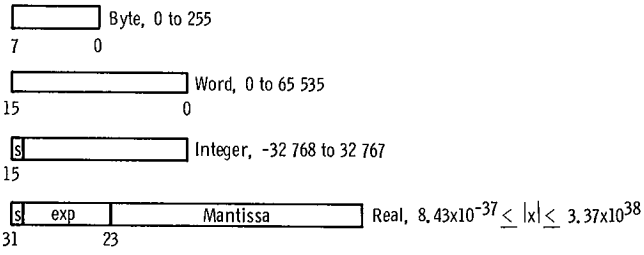

Figure 10.—MINDS data type definitions.

From these definitions of what data types are available, the format for designating a MINDS data element was devised (fig. 11). A data element designator comprises at least four and possibly five pieces of information:

(1) An element name consisting of one to six alphanumeric characters, the first of which must be alphabetic; when less than six, it is padded with space characters.

(2) A type designator, which identifies the data element as word, integer, or real

(3) The segment and offset (i.e., the location) designators for the data element. (Note that any memory location in the 8086 must be specified by using a segment and offset because of the architectural definition of the processor.)

(4) For integer data elements a scale factor is included. For a data element of the type "word" no scale factor is necessary since it is merely considered as an unsigned number that represents a pure hexadecimal value. Similarly, real numbers can be considered pure values since they vary over such a wide range that most values encountered in a physical system can be represented in the real format. Integers represent a different situation. In various applications it may be advantageous to represent some or all internal central variables as integers rather than as real numbers. This might be done for example when calculation speed becomes a factor since integer arithmetic is much faster than real-number arithmetic. In this case the integers can be thought of as scaled fraction numbers whereby a certain number of engineering units are represented by a certain number of machine units. For example, 15 000 rpm might be represented by the maximum integer number, or 32 767 machine units. To facilitate understanding of the controls operation, every integer data element contains a scale factor. Therefore, when any reference to that element's value is made, it is modified by the element's respective scale factor for display purposes. This allows the integers to be dealt with like real numbers. That is, for data

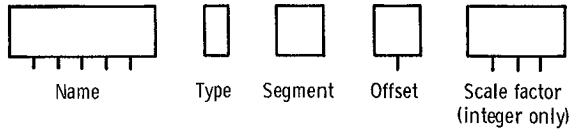

Figure 11.—MINDS data element format.

manipulation purposes they can be dealt with directly by using their engineering unit values.

Command Control Structure

The input command structures for the MINDS main and transient software blocks (fig. 9) follow either of two formats. Each of these formats must contain a command operator and may or may not contain the "name" of a data element. A command operator is either an ASCII punctuation character or an ASCII control character. Examples of ASCII punctuation characters are ".", ":", ";", and "!" control characters are "cntr P", "cntr 0", "cntr H", etc. A type I format involves simply a command operator along with an optional set of parameters if required. For example, a type I command is

? (CR)

where "?" is a request to print out certain information and "(CR)" is a carriage return necessary to signify the completion of a command sequence and to start the software interpreting and processing of the command. A type I input command would be

: H1, H2 (CR)

where ":" is a request to add numbers (or named data elements) H1 and H2. A type II format contains a data element name followed by a command operator; for example,

NAME = (CR)

where "NAME" is the name of a previously defined data element and " = " is the command operator requesting a display of the currently stored value of the memory location name.

Command Interpreter

The method by which input commands are interpreted and processed by MINDS is shown in figure 12. The start flag is the entry point to the MINDS command interpreter from the real-time subexecutive program. Once entered, a command is stored in the input read buffer. The buffer is located in the MINDS input/output library module (fig. 9). How the library works is described in detail later in this report. Once the input from the console is complete (signified by the occurrence of an ASCII carriage return), the first character is examined to determine if it is an alphabetic character. If it is not alphabetic, it is assumed to be a command operator of the type I format. The input character is then compared with a list of valid command operators. If there is a match, the command processor software will be permitted to read the remaining input string from the input buffer and process the complete command. The

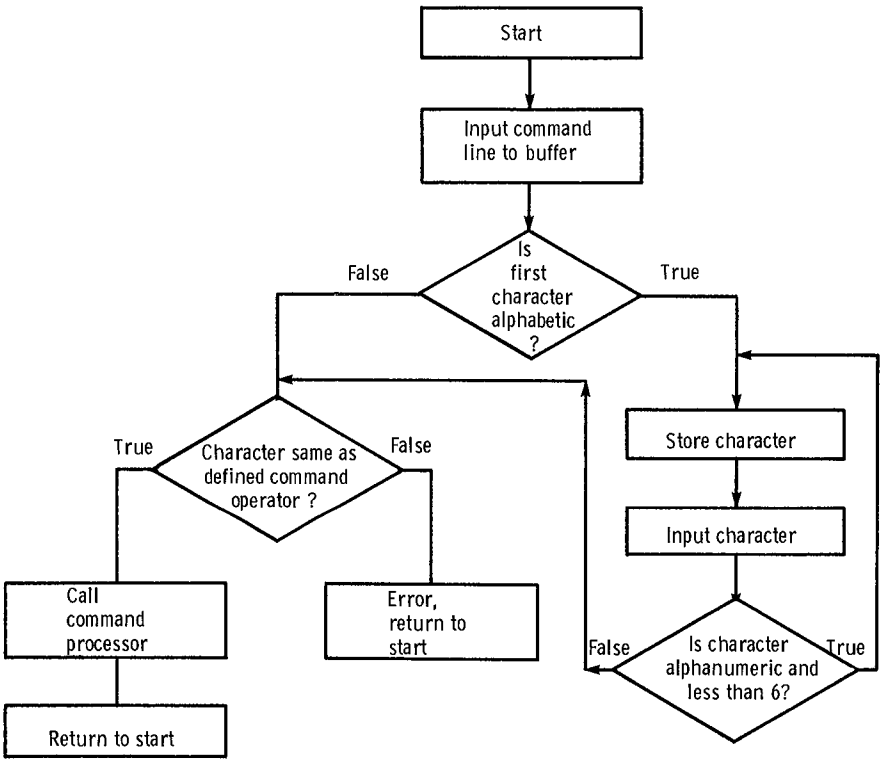

Figure 12.—MINDS control command structure.

program then returns to start for more command inputs. If there is no match to a command operator, an error is issued and the user is invited to enter another command. If the first character is alphabetic, the command is assumed to be of type II. Under these circumstances, the character is stored and the next one examined to determine if it is part of a name. This is done until either a nonalphanumeric character is encountered, or a six-letter name is created. In either case the command operator scan is performed to determine if a valid command has been input. Note that if a less-than-six-variable name is input, NAME is padded with spaces to make it six characters.

Each time a keyboard input is given to the MINDS software, the command interpreter goes through the steps just described. In the event an error is detected during the execution of a command, the command processor prints an error message. The MINDS error codes are listed in appendix C.

Data Element Designator Table Format

As discussed earlier, data elements are used in the steady-state and transient data collection processes. The MINDS main software module contains the routines necessary to define and manipulate data elements. When a data element is defined, the information is stored in a series of parallel data tables (fig. 13). The lengths of the tables as shown in the figure will allow for the definition of 256 data elements for each table. The NAME$TAB table reserves six bytes for each data element name. Each byte contains one ASCII character from the data element name. Since each data element name must contain six characters to provide correspondence with the other parallel tables, names that are less than six characters are padded with spaces. The VAR$TYPE table has 1 byte reserved for each data element. Corresponding to each data element in the VAR TYPE table is the ASCII character "W," "I," or "R" identifying the data element as a word, integer, or real number, respectively.

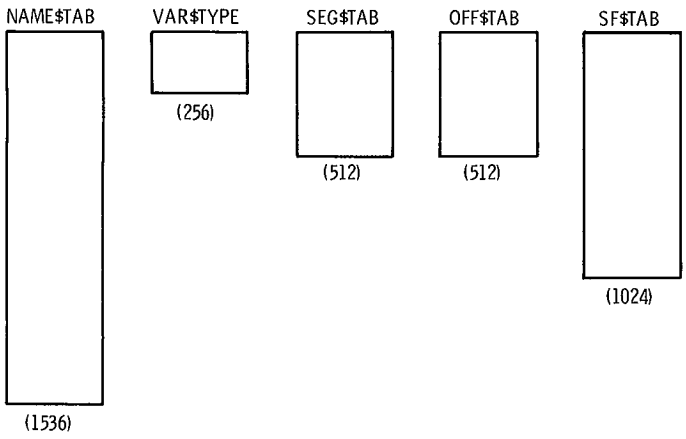

Figure 13.—MINDS data element storage format.

The SEG$TAB has two bytes reserved for each data element that identify which memory segment the data element occupies. Likewise, the OFF$TAB also has two bytes reserved for each data element that identify its respective offset. Finally, the SF$TAB allocates four bytes for each data element. If the data element is defined as integer data, the SF$TAB contains a real number requiring 32 bits (4 bytes), that corresponds to that particular data element's scale factor. If reference is made to the modification or display of the data element, the program will use the scale factor to manipulate the element. Note that only data elements of integer type have a scale factor. Word and real data elements have no scale factor; therefore manipulation of these two types of elements results in the use of the actual bit pattern currently stored in memory.

Command to Manipulate Tables

Nine commands allow for direct manipulation of data elements defined in the master table. Six of the commands use type II command format and three use type I.

The type II commands include " ", representing a space character, used to define a data element; "_", used to overlay the definition of a previously defined data element with a new data element; "&", used to display definition information of the data element; " = ", used to display the value in the appropriate units of the respective data element; "#", used to display the value of the data element in binary; and "'", used to set the data element to a particular value. The type I commands include "?", which displays the names and definitions of all of the defined data elements and "[", which displays the 8086 segment registers' values for the currently loaded program. Finally, because of the loader operation in CP/M-86, the possibility exists that a program containing MINDS could be loaded into a different portion of memory every time the size of the program changes. Therefore, the segment values of data elements will also change. Consequently, the command "%" has been implemented to allow for changing the values of all of the data element segment definitions. More detailed command formats and default options for each command are given appendix D.

Steady-State Data Collection Software

The ability to take and display tables of values of the control steady-state data is one of the most important capabilities of MINDS. To accomplish this task, provision must be made in the program to create tables of data elements, to collect the values for the data elements in a timely manner, and to output the collected data element values in a form that is useful to the control

operator. The following paragraphs discuss each of these needs and the implemented solutions.

Steady-State Data Table Format

To make the MINDS steady-state data collection as flexible as possible, provision has been made to store definitions of four logically separate steady-state data tables. The storage format structure (fig. 14) shows three arrays, DATA$TAB, DATA$TAB$LMT, and DATA$TAB$BIAS, that together define the steady-state data sampling structure. The DATA$TAB table stores the index values that a particular data element has in the master data element storage format (fig. 13). By using the DATATABBIAS table to define boundaries in DATA$TAB, all four of the logically separate data tables can be stored in DATA$TAB. For example, in DATATABBIAS the first element is 256 and the second element is 406 (array element numbering starts at zero), signifying that logical data table 1 runs from element 256 to, but not including, 406 in the DATA$TAB array. Similarly, tables 2, 3, and 4 run from 406 to 456, 456 to 572, and 572 to 768, respectively. The last word in DATATABBIAS, containing in this case 768 elements, defines the maximum size of DATA$TAB. The DATA$TAB$LMT keeps track of the number of data element indexes that are stored in each data table. In this case the first element in the DATATABLMT array can be as small as zero if no elements are in the table or as large as 150 if the table is full. This provides a convenient counter not only when collecting and printing data but also in determining when a data table is full. Finally, three facts should be noted. First, the numbers in the DATATABBIAS allow the logical data table size to be varied as desired. Second, the size of the three arrays can be increased to create as many logical data tables as desired. Third, although it cannot be referenced or manipulated directly, a table zero has been provided for in the data table structure. Every time a data element definition is added to the master table, the appropriate index value is automatically added to logical steady-state sampling table zero. This table can then be collected by the automated data collection command discussed in a later section.

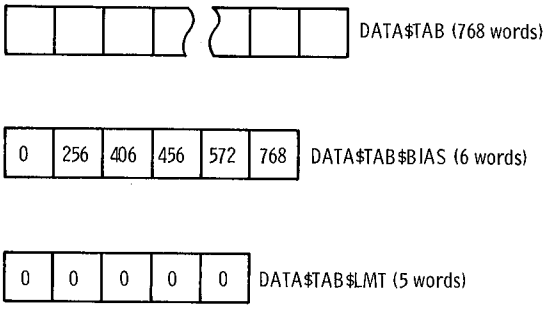

Figure 14.—MINDS steady-state sampling data structure.

Steady-State Data Collection Methods

To use the capabilities built into the MINDS package, the user must be aware of the problem of data skew when taking steady-state data. Since steady-state data are taken during the control calculation idle time, the control can preempt the data-taking process. When this happens, values inside the control can change. Therefore, if some of the values in the sampled data table are taken in one update interval and some in the next, all of the data may not match exactly. This phenomenon is known as data skew.

Several factors can affect how much data skew results during data collection. First, if the data are collected and output variable-by-variable, the amount of data skew is dependent on the speed of the output device. Therefore, a data collection buffer to collect data before they are printed is provided to eliminate this problem. Second, MINDS must be able to collect values of word, integer, and real data (fig. 10) in a transparent manner. This becomes a problem because word and integer data occupy two bytes, but real data occupy four bytes. For the lowest computation overhead a simple flag mechanism is used to indicate to the data collection program whether a data element consists of two bytes or four bytes. A DATA$TAB table (fig. 15) stores logical data element indexes that are simply numbers associated with variable names. By testing the flag bit in each index value it is easy to determine if two or four bytes of data should be stored. Finally, as indicated previously and shown in figure 2, the address of any data location is determined by the addition of a segment register and an offset register. Consequently, an increase in data collection overhead, and hence data skew, is realized if the segment register must be changed for every data element value collected. Since many programs are less than 64 kilobytes, a large number of situations arise

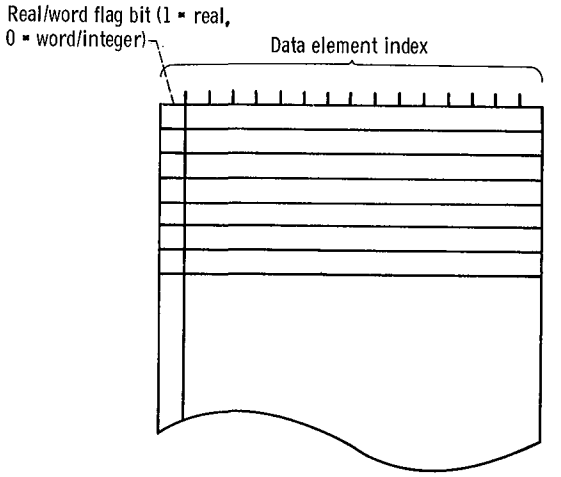

Figure 15.—Real/integer flag bit in DATA$TAB array.

where this segment register reload would be redundant and hence cost valuable time in data taking.

To accommodate all of these data-taking constraints, but also to make the MINDS program as flexible as possible, four data collection modes have been defined. Mode A is used when all of the data elements are either integers or words and they all are in the same 64K segment. Mode B is used when all of the data elements are either integers or words but are in different 64K segments. Mode C is used when the data elements are real, integer, or word and occupy the same 64K segment. Mode D is used when the data elements are real, integer, or word but occupy different 64K segments. Clearly, mode D is the most general mode since it provides for all eventualities. However, modes A to C have been provided to reduce data skew in cases where the data elements are of a specialized nature and advantage can be taken of this fact.

Data Display

Once the steady-state data have been collected, they can be displayed in three formats (fig. 16). Format 1 prints the data element names and beneath them the corresponding values of the data elements. Format 2 prints only the values of the data elements, and format 3 prints only the data element names (i.e., the heading). These table output formats provide a sufficiently rich selection to handle most steady-state data output when device printer speed is not critical.

Steady-State Data Collection Commands

Seven commands are used to manipulate the steady-state data tables and to print steady-state data. Six of the commands are of type I format and one of them is of type II. The type I commands include "(", to open a steady-state data table to allow data elements to be input; ")", to close the currently open data table; "@", to erase all of the data elements defined in the currently open table; "^", to erase the last data element entered in the currently open data table; "]", to add a new line to the currently open data table, and "\", to collect data and print the currently specified data table. The type II command is ",", which adds a currently defined data element to a currently open data table. More detailed command formats and default options for each command are given in appendix D.

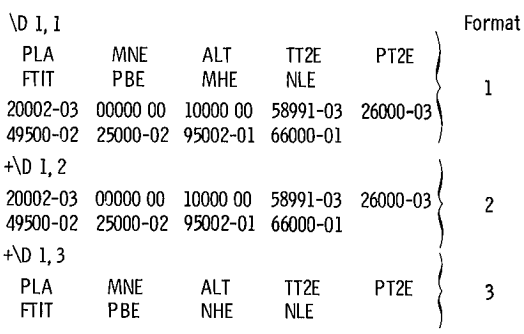

Figure 16.—Form of steady-state data display.

File Manipulation Commands

CP/M-86 File Structure

One of the most important reasons for using an off-the-shelf operating system like CP/M-86 is the ability to use its file control system to retrieve and store data from a floppy disk. Communication to this disk file management capability can be done through a 36-byte data structure known as the file control block (FCB). The FCB contains such information as the drive number the file resides on, the file name, and various file attributes. Further information on the operation of the file control block can be found in references 2 and 10. Since PL/M-86 is a system implementation language, no provision is made for direct manipulation of FCB's. That is, no operating system interface libraries exist to directly link such things as CP/M-86 and PL/M-86. Therefore, routines had to be written to accomplish this interface. These routines are documented in appendix B, along with instructions on how to use their calling sequences.

Storage and Retrieval of Program Definitions

A program such as MINDS would not be very useful if data element definitions and steady-state data table sampling definitions had to be typed in fresh every time the program was reloaded. Therefore, provision has been made in the form of two commands "?" and "*" to store and retrieve, respectively, all of the definitions that have been input through the MINDS software. Consequently, once MINDS has been "taught" definitions for a particular control, these can be retrieved and modified at any time. These commands are described in appendix D.

Automated Steady-State Data Acquisition

In testing situations where steady-state data must be acquired rapidly or must be archived for a long period of time, perhaps for reprocessing, an automated data acquisition process has been defined. This is done by using logical data table zero in the MINDS data collection structure. Since this table contains all of the index numbers for all of the data elements defined in the MINDS master tables, the values of all of the data elements can be stored in the MINDS steady-state data collection buffer. Once this has been done, these data can be stored in a CP/M-86 data file on disk. The name of this file contains a number preassigned by the operator to correspond to an operational reading number that these

particular data represent. The format in which each of these readings is stored on the disk is discussed in appendix E. The MINDS program also contains provisions for retrieving the steady-state data file from the disk and restoring it in an intermediate data collection buffer. The predefined logical data tables (i.e., tables 1 to 4) can then be printed out by using the data retrieved from the disk.

Commands for Automated Data Display

Five commands are used in automated steady-state data taking. All five commands are of type I format and include a special form of "\" to take the steady-state data and store them in a temporary buffer; "$", to store the data in a predefined file on floppy disk; "/", to recall a data file from disk and store it into a temporary buffer; ";", which will allow logical data tables 1 to 4 to be printed out by using the data retrieved from the disk; and finally the "″" command, to update the name of the predefined file where data are to be stored. Further information on these commands is given in appendix D.

Miscellaneous Commands

Six commands, all of type I format, help in program operation but do not play any role in data element manipulation. First, "-" and "<" allow entry into the MINDS transient sampling program and monitor, respectively. These portions of the MINDS package are shown in figure 9. Second, "." allows one to exit from the MINDS main command structure entirely. Third, the ">" and "{" commands allow the MINDS package to display a CP/M-86 disk directory and to delete a file from a CP/M-86 disk, respectively. Finally, the ":" command allows for the addition and subtraction of two hexadecimal or decimal numbers. Further information on these commands is given in the appendix D.

Data Collection Software

Overview

In the analysis of a control or simulation system its transient behavior is second in importance only to its steady-state set point. Therefore, the MINDS program provides a method whereby data element transient data can be collected. This capability is represented in figure 9 as the MINDS transient data block. In addition to sampling transient data this software module can automatically set up the transient timer and interrupt controller, automatically manage memory for the transient variables, and provide a simple storage and recall procedure for transient data.

Sampling of Transient Data

The process of sampling transient data is shown in figure 17. The bottom half of this figure is the direct control timing diagram explained previously as figure 1. Because this task of sampling data, control calculation, and digital-to-analog converter (DAC) output occurs at the highest priority available, it proceeds as it must, without outside interference. Where transient data are required, a separate timer is used as a transient sampling signal (top half of fig. 17). Because this timer interrupt has a lower priority than the one used in the control, the transient sampling process does not interfere with the control. Each time that a transient sampling interrupt

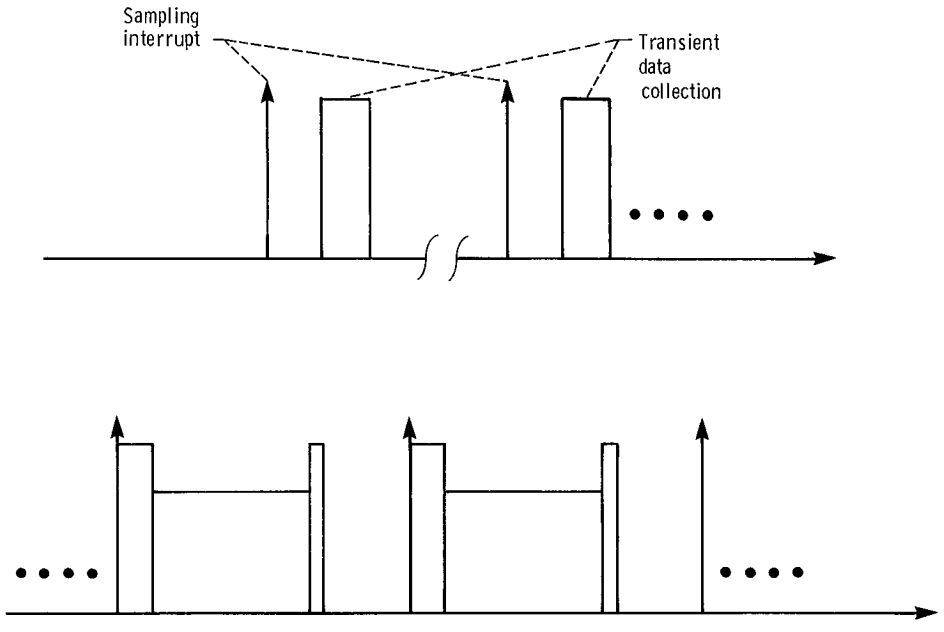

Figure 17.—Direct control timing with transient sampling.

occurs, one frame of transient data is collected and stored in the control computer's free memory. However, as figure 17 illustrates, even though the transient sampling interrupt can occur anytime, the actual collection of the data can only take place during control calculation idle time. Transient sampling interrupts occur until such time as enough data frames have been collected at the prescribed sampling interval to account for the full length of the transient.

Since a separate timer is used to trigger the transient sampling process, it can run asynchronously to the control. Consequently, the transient sampling interval need not be an integer multiple of the control interval. Generally, the transient sampling interval is greater than five times the control interval. This choice allows the collected data to have adequate fidelity for plotting and transient analysis purposes without using large amounts of memory to store intermediate points that would be of questionable value.

Transient Sampling Data

Structures and Memory Allocation

There are two distinct data structure issues in transient data sampling. The first is the MINDS dynamic sampling data structure. This is the structure of the internal tables telling which data elements are to be sampled. The second is the structure and allocation of the free memory in the microcomputer to allow the storage of data element values during the transient.

The MINDS dynamic sampling data structure (fig. 18) is similar in many ways to the one used to store steady-state data collection tables. Like the steady-state data collection structure the TRANTABNAME array stores 256 data element definition indexes to show which data element is to be sampled. The TRANTABBIAS array divides the one TRANTABNAME table into four logical data collection tables with partitions at the values shown. The TRANTABLMT array gives a count of the number of elements in each table. This allows for easy counting of the number of variables to be sampled. Note that in the transient data tables there is no logical table zero to allow collection of all defined variables since it is unlikely that time histories of more than a small subset of variables would ever be needed. The TRAN$STORE$TAB table stores the addresses of the starting locations of each of the memory partitions for storage of the respective data element transients. Since TRAN$STORE$TAB is only 64 locations long, it can only hold the starting locations for one logical transient table at a time. The reasons for this become clear in the next section.

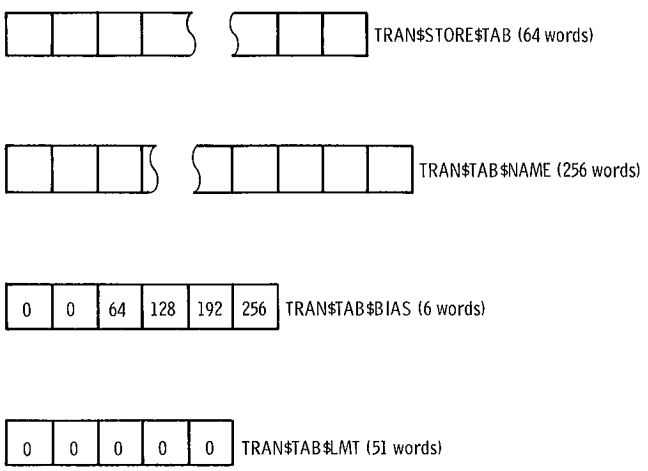

Figure 18.—MINDS dynamic sampling data structure.

During a transient each of the time histories of the respective MINDS data elements is placed in free memory. Because the size of this free memory can vary between 64 kilobytes and 1 megabyte depending on the processor board being used, it is necessary to use the memory management features built into the CP/M-86 operating system to allocate the memory in an intelligent fashion. Consequently, the memory model shown in figure 19 represents microcontroller memory as allocated by CP/M-86 when transient data are being collected. As shown in figure 7 the interrupt vector table and the CP/M-86 operating system occupy the lower 12K of memory. The real-time executive, the control algorithm, and the MINDS package are loaded into the highest memory locations available in the respective microcomputer. The free memory for storing transient variables is between the upper portion of the operating system and the lower portion of the executive/control/MINDS program. Each transient variable that is to be collected is allocated a portion of this memory for

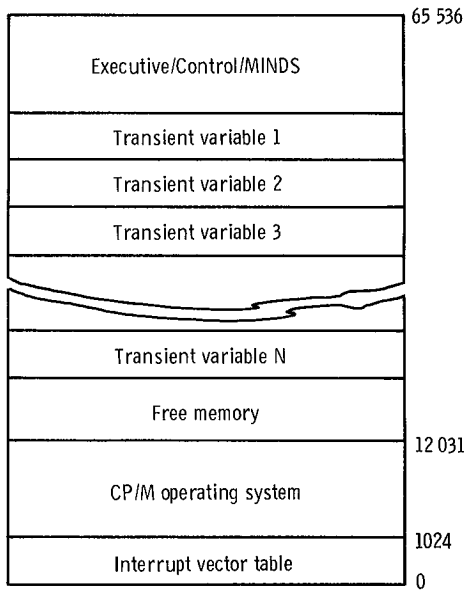

Figure 19.—MINDS transient sampling memory map.

its time history (fig. 19). This is done by the MINDS transient software (fig. 9) in conjunction with the CP/M-86 operating system. The partition assignments start just below the executive/control/MINDS program and "build down" in main memory toward the top of the operating system. The size of the partition for each time history is a function of the data element type (real, integer, or word), the length of the time history requested, and the sampling interval between data points.

Because each of the logical transient data tables can contain any mix of word, real, or integer data elements, clearly the memory partitions that one logical data table would require may be quite different from the requirements of another. Therefore, each time that a different logical transient data table is sampled or the length or sampling interval of the transient time history is changed, the memory storage partitions must also be changed. This is done by arming or disarming a particular transient table. This process sets and resets the memory portion addresses in the TRAN$STORE$TAB to accommodate the particular logical transient data table being sampled. Once the arming process is done for one of the logical transient's storage tables, transient data sampling can commence.

Transient Data Software Commands

The command structure for the transient data taking is the same as the one for the main MINDS program. This command structure is entered by using the "-" command in the MINDS main command structure and listed under Miscellaneous Commands in appendix D. Once this command is executed, the MINDS transient data block is entered. The command structure in this block is identical to that shown in figure 12. This command structure has type I and II commands as described previously. Many of the commands use the same format as in the main MINDS program, but in this case they influence items and data in the transient sampling area. A description of the four classes of commands follows. Additional information on these commands is given in appendix D.

Table Definition Commands

The transient table definition commands are similar to the ones described in a previous section for steady-state data. Five of the commands are type I commands; the last two are type II. The type I commands include the following: "(", opens a transient data table to enter data element names; ")", closes the currently opened transient data table; "^", deletes the last member from the currently open transient table; "@", deletes all of the members from the currently open transient table; and "?", displays the current entries in a designated transient table. The two type II commands are ",", enters an element in a transient data table, and "_", overlays a member of a currently open data table with another.

Sampling Parameter Commands

Four type I commands are used to control the sampling parameters. These commands are used to display the amount of memory available for sampling and to display and alter the transient sampling interval and the transient sampling length. The four commands are as follows: "%", invites the operator to enter the desired sampling interval and sampling length; "&", displays the currently selected sampling parameters; "[", displays the maximum memory available with the program loaded and all transients disarmed; and "]", displays the free memory still available with a logical transient data table armed. Note that the "]" and "[" commands give equal answers in the event that no logical data tables are armed to take a transient.

Data Collection Commands

Three type I commands are used in transient data collection. These include "{", which arms (i.e., allocates free memory to) one of the four logical transient tables for sampling; "}", which disarms (i.e., deallocates memory to) a logical transient table; and "\", which takes the actual transient sample and stores it in the allocated memory.

Data Storage Commands

The data storage commands allow the sampled data to be retained for later analysis. There are four of these commands, once again all type I. The "$" command allows sampled data residing in free memory to be stored in a disk file; "*" allows the recovery of sampled data stored on disk to be restored into free memory; ";" allows the sampled data residing in free memory to be printed out on a console device; and "'" updates the name of the disk file where the collected data are to be stored.

Operation of Transient Data Package

The operation of the transient data package is fairly straightforward because most of the critical data management functions are taken care of automatically. To use the program, a logical transient data table must be filled with data element names by using the "," command. Next the sampling time and sampling interval must be set by using the "%" command. Then the particular logical data table defined must be armed. As explained earlier this means that each of the data elements in the logical data table is assigned to a particular partition in free memory. Next the actual data sample is taken by using the "\" command. Finally the data are archived on a disk by using the "$" command. These data can be read back later by using the "*" command, printed out, or uploaded to a mainframe system for data analysis. The "." command allows

exiting from the transient sampler to the main MINDS command structure.

Clearly, although the "\" command can be used to take data, many situations require that the data be synchronized with an external signal. This can be done quite easily since the sampling routine that is called by the "\" command is the public entry point SAMPINT1 (appendix B). Therefore, if the real-time executive is constructed such that it recognizes when an external sample signal is present, the SAMPINT1 subroutine can be called to sample the data elements with the parameters set up by the MINDS transient block (fig. 9).

Monitor Package

The third operational block of the MINDS package shown in figure 9 is the MINDS monitor and debug. A monitor is usually the first piece of software acquired for a microprocessor; it lets the user display and change memory, display and change processor registers, execute programs, and set breaks or traps in these programs for debugging. This MINDS software module provides the user with some of the features typically found in monitors, it extends some of these features, and it modifies some features to make the software more useful in debugging the real-time control discussed earlier. All of these features and important facts on the monitor operation are highlighted in the following sections.

Command Structure

The command structure for the monitor is somewhat different from the command structure used in the main and transient blocks (fig. 12). In the monitor block command structure (fig. 20) only type I commands are recognized. As shown in the figure, once the command has been completely entered into the input buffer, the first character is examined to determine if it is a valid command. If it is valid, the respective command processor is called on to parse the rest of the command and execute it. Although the data element names are known to the monitor block, they are not used to generate a type II command format. This then makes this portion of the program behave like a classic monitor.

Standard Monitor Functions

The MINDS monitor has the four standard monitor commands to manipulate memory. These commands are "H", to add and subtract two 16-bit values; "D", to display a segment of memory; "S", to set a series of memory locations one at a time; and "F" to fill a series of memory locations with the same value. Although these

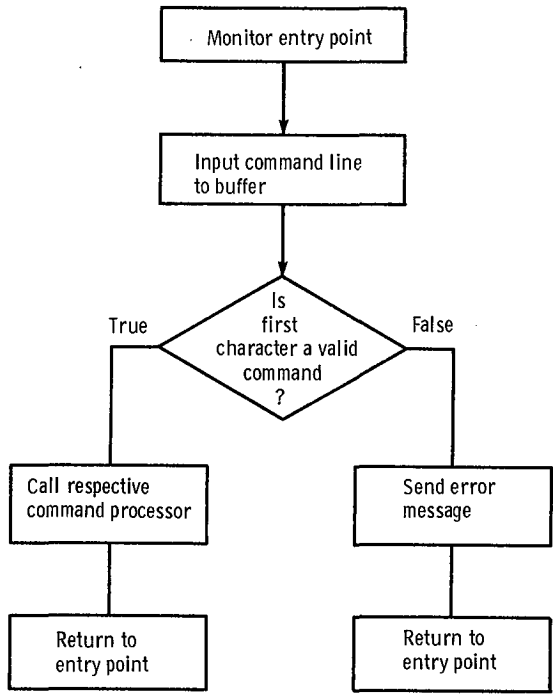

Figure 20.—MINDS monitor command structure

functions are found in any standard monitor, each has been slightly extended to make it more useful in a control environment. The "H" command allows the entry of any two numbers in decimal or hexadecimal format and gives their sum and difference in decimals or hexadecimals. The "D" command allows the display of any region of memory. The memory can be displayed as bytes, words, integers, or real numbers. Displaying a byte results in a hexadecimal number between 0 and 255. Displaying a word results in a hexadecimal number between 0 and 65 535. When a real number is displayed, a memory image value between $10+38$ and $10-37$ is output. Finally, when an integer value is displayed, an optional scale factor can be defined in the command line. Therefore, a scaled fraction number is output with a range dependent on the defined scale factor.

The "S" command operates in much the same way as the display command. It allows the setting of byte, word, integer, or real values one at a time in sequential memory locations. Like the display command, the "integer" variable type in this command has an associated scale factor. This allows the integer variables to be treated as scaled fraction numbers with their display subsequently being in engineering units. The "F" command allows for the filling of a large block of sequential memory locations with a particular piece of data. Finally the "." command allows exiting from the monitor to the main MINDS command structure. In the interest of brevity in the

command structure only bytes, words, and real data can be used in this instruction.

Breakpoint Debugging

Most monitors allow the user to set breakpoints, start program execution, and then trap to the monitor once the breakpoint has been reached. This allows for step-by-step assembly language debugging of the application software. Controls, however, are a somewhat different story. Because they execute repeatedly, they operate on different values of input data each time through the code. Since errors can occur transiently or only at certain values of the sensed inputs, it would be convenient to be able to trap to the monitor only at times when a particular condition is fulfilled. To attempt to set conditional breakpoints with a standard monitor while not altering the control code execution in the cases where the condition is not satisfied would require substantial code patching and possible reassembly for every new condition that is tried. The MINDS breakpoint set and breakpoint execution routines address these shortcomings. First, the breakpoint set routine allows the user to set a breakpoint anywhere in the microcomputer's 1-megabyte address space. Second, once this breakpoint is reached, it allows for the collection of the 8086 registers and status flags along with any one of the four steady-state data tables. Finally, the breakpoint data are taken only in the event that a certain condition is met. The breakpoint specification is set by using a data element and specifying a condition that it must be "greater than," "less than," or "equal to." Before the breakpoint specification is met, the control process proceeds unimpeded. However, the breakpoint condition is checked each time that the breakpoint memory location is executed. Once the breakpoint specification has been satisfied, the appropriate register and table data are collected and the breakpoint is removed from the code.

This technique of inserting dynamic conditional breakpoints allows for the quick isolation and debugging of transient errors found in a real-time control. These errors would be extremely difficult to isolate with a conventional monitor.

Breakpoint Command

Four commands are used to implement and evaluate breakpoints. Each command is of type I format and is implemented by using an alphabetic character. The "B" command allows all of the breakpoint parameters to be set. This includes the setting of the breakpoint location, the using of a data element to set the breakpoint condition, and the choice of a data table to be collected. The "G" command causes the actual implementation or insertion of the breakpoint into the executing code. The "X" command displays all of the processor registers as they appear when the breakpoint is reached. The "T" command allows the display of the data table that is collected when the breakpoint is reached. The "R" command prints out the current values of only the segment registers so that locations can be quickly determined.

Input/Output Library

Overview

The last block shown in figure 9 that is critical to the operation of the MINDS package is the input/output library. This library is a collection of routines that facilitates the input and output of commands, numbers, and messages to a console or line printer and allows high-level command of the CP/M-86 file system. The writing of a library like this is necessary because PL/M-86 is a system implementation language. Consequently, no software input/output drivers or libraries are typically present in languages such as Fortran or Pascal. The following discussion gives the overall conceptual structure of this library. Appendix B gives the calling sequences used to access the routines at the library software interface.

Library Structure

The overall structure of the MINDS input/output library is shown in figure 21. The diagram shows the MINDS instruction processors (main, transient, and monitor blocks) interfacing into the input/output library software through a set of subroutine calls. The library can be easily divided into four parts: the hardware-specific routines, the output routines, the input routines, and the CP/M-86 operating system interface. The hardware-specific routines contain hardware control sequences for controlling the actual physical output and input ports. These routines, although very small, must be tailored to the specific hardware being used. A listing of the routine used for the serial input/output port on the iSBC 86/12A board is included in appendix F.

The output routines consist of two major blocks of code. The character output routine allows an ASCII character string of any length to be output to the console. The numerical output routines have the following capability: output of 8-bit memory locations as hexadecimal numbers or unsigned decimal numbers; output of 16-bit memory locations as hexadecimal numbers or signed or unsigned integers; output of 32-bit memory locations as plus or minus exponential numbers. The

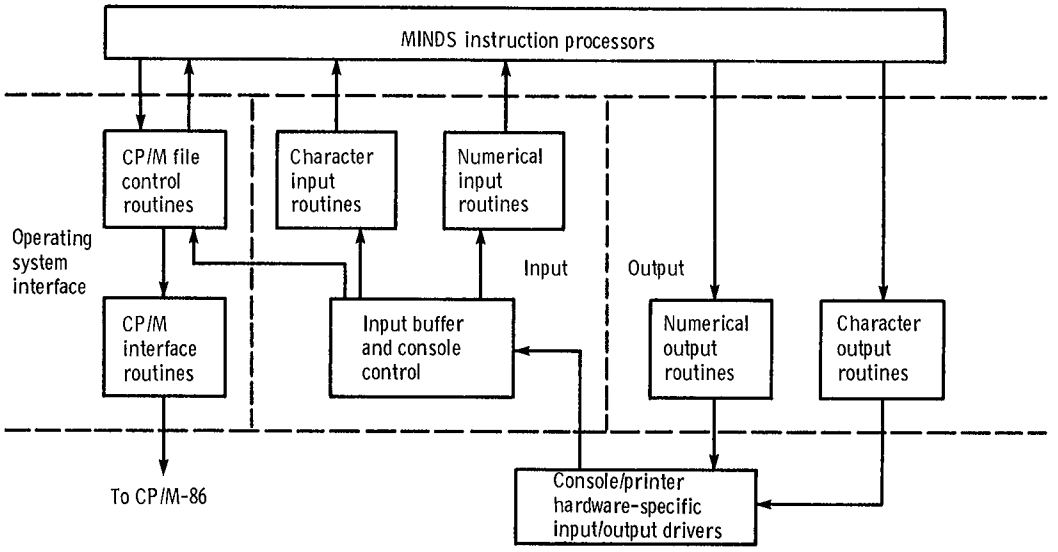

Figure 21.—MINDS input/output library structure.

numerical output routines for each case take the binary bit patterns in memory and translate into an ASCII character string for printing on the console.

The input routines consist of three major blocks of code. The input buffer and console control do the major storage and editing of the ASCII character strings from the operator. The editing function includes backspacing, line deletion, console bell control, and printer control. Further information can be found in appendix D. Once the input buffer's full command signal has been received (i.e., an ASCII carriage return), the character input routine block allows the MINDS command processors to access the input buffer. Furthermore, the numerical input routines in the third block can access ASCII strings in the buffer and translate them into the proper memory image of the number. These routines can take numbers that are input as hexadecimal, plus or minus decimal, or even plus or minus floating point with optional exponents and translate them into the proper memory images.

The operating system interface routines consist of two blocks of code that allow the MINDS command processor to directly interface into the CP/M-86 operating system. The CP/M-86 interface routines are a set of small assembly language programs that allow high-level languages like PL/M to set up the 8086 registers so that various operating system services can be accessed. The next block, CP/M file control routines, allows for higher level access to the operating system. These routines will create file control blocks, read and write files in various formats, close files, delete files, print disk directories, etc. By using these routines the MINDS instruction processors can easily access the CP/M-86 file system.

The MINDS instruction processors interface into this input/output library through subroutine calls that send or receive information in the manner shown by the arrows in figure 21. The input/output library forms flexible console input/output system and operating system file interfaces that work not only with MINDS but with other programs as well.

Extensions and Modifications

The previous sections describe a flexible data-taking package that can be used in a variety of applications. Several projects that have already successfully used this software are described in reference 11 and 12. During the conduct of these projects several modifications to this package have become apparent. First, various versions of the program can be implemented that use some or all of the blocks shown in figure 9. All that is necessary for the program to have the minimum steady-state data collection ability is to link the MINDS main program and the MINDS input/output library. The transient data collection block and the monitor and debug block can be added to the program as the need arises. Furthermore, the number of variables that can be defined (the standard version has 256) can be increased or decreased as the program demands dictate. This allows program compaction in cases where available computer memory may be tight.

Finally, all that can be done with transient data is to collect them, store them, recall them, and print the numerical data points on a console device. To plot the data, they must be transferred to a large mainframe computer, reduced into proper form, and subsequently plotted by using a standard graphics package. A plotting capability could be very easily added to the transient data block or as another complete data block to facilitate the on-line manipulation of control parameters. Note that this can be accommodated easily since all of the pertinent transient table information is public.

Concluding Remarks

This report has described a program that can be loaded with a control algorithm to provide steady-state and transient data collection. The support hardware and software necessary to run the program have been described. The program was broken down into its four major component parts: the main program, the transient block, the monitor and debug block, and the input/output library block, with each one being discussed in detail. Finally, extensions and modifications that can be made to enhance the program capabilities were discussed. Detailed appendixes are provided on commands, data table formats, public program entry points, etc.

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio, October 7, 1984

Appendix A
Interrupt Map

Location (hexadecimal)	Function
0-3	divide interrupt
4-7	single-step interrupt
8-B	nonmaskable interrupt
C-F	1-byte instruction interrupt (type 3)
10-13	overflow interrupt
14-7F	Intel-reserved interrupts (mostly for floating-point mathematics emulation)
80-83	instruction timeout interrupt (8259 interrupt 0)
84-87	unused (8259 interrupt 1)
88-8B	control timer interrupt (8259 interrupt 2)
8C-8F	unused (8259 interrupt 3)
90-93	unused (8259 interrupt 4)
94-97	multiplexer A/D data ready (8259 interrupt 5)
98-9B	unused (8259 interrupt 6)
9C-9F	transient sampling timer (8259 interrupt 7)
A0-A3	debug intermediate condition
A4-A7	debug less than condition (word)
A8-AB	debug equal to condition (word)
AC-AF	debug greater than condition (word)
B0-B3	debug less than condition (integer)
B4-B7	debug equal to condition (integer)
B8-BB	debug greater than condition (integer)
BC-BF	debug less than condition (real)
C0-C3	debug equal to condition (real)
C4-C7	debug greater than condition (real)
C8-DF	unused
E0-E3	CP/M-86 operating system BDOS call

Appendix B
MINDS Module Description

The MINDS data-taking package can be logically broken into the four blocks shown in figure 9. However, each of these blocks consists of several PL/M or assembly language object modules to fulfill the required functions. In addition each of the object modules may have several public entry points or public variables. Therefore, each of the load modules falls into one of four logical areas:

(1) MAIN program—MINDS.OBJ; INIT.OBJ; TABOUT.OBJ; and GETVAR.OBJ.
(2) Transient data—MTRAN.OBJ; SMPTRN.OBJ; and MTRANS.OBJ.
(3) Monitor and debug—MNDMON.OBJ and INTRBK.OBJ.
(4) Input/output library—IOCON.OBJ; MESSAGE.OBJ; OUTH.OBJ; IOUT.OBJ; OUTR.OBJ; INBUFF.OBJ; RDATA.OBJ; ININT.OBJ; INHEX.OBJ; INREAL.OBJ; CPMIO.OBJ; and DISKIO.OBJ.

The function of each of the object modules, the calling sequences for the pertinent public routines, and any public variables that they contain are discussed here.

Main Program

MINDS.OBJ

MINDS.OBJ, a PL/M routine, is the main MINDS program. It contains the main command interpreter routine, the routines to define data elements, the routines to define steady-state data tables, the routines to automatically store transient and steady-state data in a CP/M-compatible file, and the routines to recall the steady-state data from the CP/M file and display them on the console. This module contains 6827 bytes of code and 7220 bytes of data.

Public Entry Points

CALL MINDS—main entry point into the command interpreter (no calling parameters; no return parameter)
CALL ERROR (CHAR)—procedure to print error message followed by CHAR, where CHAR denotes an ASCII alphanumeric character
CALL SPACE (NUM)—procedure to print spaces on the console, where NUM denotes the number of spaces to be printed

Public Variables

NAME$TAB—array to store data element names
SEG$TAB—array to store data element segments
OFF$TAB—array to store data element offsets
SF$TAB—array to store data element scale factors
VAR$TYPE$TAB—array to store data element types
DATA$TAB—array to store steady-state data table definitions
DATATABBIAS—array to store steady-state data table partitions
DATATABLMT—array to store number of variables in each steady-state data table
TRANTABNAME—array to store transient data table definitions
TRANTABBIAS—array to store transient data table partitions
TRANTABLMT—array to store number of variables in each transient data table
TRAN$STORE$TAB—array to store memory locations to be used in storing transient history

INIT.OBJ

INIT.OBJ is an assembly language program to be called by MINDS for initialization. It prints the sign-on message, initializes the 8087 coprocessor, and finds the segment register values for the currently loaded program. The program contains entry point INITIAL1, which is called by the IOCON program (appendix F). This entry point reinitializes the entire MINDS package if the program is interrupted during a long printout.

TABOUT.OBJ

TABOUT.OBJ is an assembly language program that collects steady-state data and prints them out to a terminal. The procedure has four collection methods and three table output formats, as outlined in the command appendix D.

Public Entry Points

CALL BRKCOLLECT—procedure to collect steady-state data. Collection method and table number are determined by public variables TABCOLLECT and TABNUM, respectively.

CALL TABOUT—procedure to print steady-state data table previously collected. Table output format is specified by public variable TABCON.

Public Variables

TABCOLLECT—steady-state data table collection method, specified as A to D (appendix D; command "\"; parameter P1)

TABNUM—steady-state data table number to be collected, specified as 1 to 4 (appendix D; command "\"; parameter P2)

TABCON—steady-state data table output format, specified as 1 to 3 (appendix D; command "\"; parameter P3)

GETVAR.OBJ

GETVAR.OBJ is a set of assembly language routines that provide MINDS with a machine-dependent interface to allow manipulation (loading and storing) of data throughout the 8086 1-megabyte address space.

Public Entry Points

CALL LSVARIW (DSEGMENT, DOFFSET, ADDIW, FUNC)—procedure to load or store a word or integer variable

DSEGMENT—segment address of variable to be loaded or stored
DOFFSET—offset address of variable to be loaded or stored
ADDIW—offset address of variable where result is to be placed or data are to be taken
FUNC—"L" if data are to be loaded from address DSEGMENT:DOFFSET and placed in location ADDIW; "S" if data currently at address ADDIW are to be placed at address DSEGMENT:DOFFSET

CALL LSVARR (DSEGMENT, DOFFSET, ADDIW, FUNC)—loads or stores a real variable in the same fashion as CALL LSVARIW, with same calling parameters

CALL LBITE (DSEGMENT, DOFFSET, ADDIW, FUNC)—loads or stores a byte variable in the same fashion as CALL LSVARIW, with the same calling parameters

CALL HEXPACK (APTR, HPTR)—routine to take 2 bytes of absolute data and pack it into 4 bytes of hexadecimal data

APTR—pointer to source data (absolute)
HPTR—pointer to result data (hexadecimal)

CALL UNHEXPACK (APTR, HPTR)—routine to take 4 bytes of hexadecimal data and pack it into 2 bytes of absolute data

APTR—pointer to source data (absolute)
HPTR—pointer to result data (hexadecimal)

Minds Transient Data

MTRAN.OBJ

MTRAN.OBJ, a PL/M-86 program, contains the command interpreter for all of the transient data sampling. In addition, it contains the routine to set the transient sampling interval and transient sampling length, the routines to manipulate the transient sampling tables, the routines to arm and disarm transients by allocating and deallocating memory, and the capability to recall and print transient data from a CP/M file. This module contains 3940 bytes of code and 998 bytes of data.

SMPTRN.OBJ

SMPTRN. OBJ is an assembly language routine that does the actual data sampling and storage of transient data tables that have been set up by the routines in MTRAN.OBJ. These routines are particular to the interval timers and interrupt controllers on the Intel 86/30 or Intel 86/12A microprocessor boards. Use on boards with different hardware would give unpredictable results.

Public Entry Points

CALL SAMPINT0—procedure called by original initialization sequence in executive to set up the interrupt vector. It is based on using interrupt vector 7 on the Intel 86/12A or Intel 86/30 board.

CALL SAMPINT1—procedure called by executive when it has detected that transient sampling should commence. This routine initializes memory counter, flags, counter hardware, etc., every time that a transient sample is to start.

CALL SAMPLE—program that does the actual sampling of the transient in three steps. First, it initializes memory that is going to store the transient. Second, it collects the data specified in the logical transient data table. Third, it stores the data in locations specified in TRAN$STORE$TAB. The location of this procedure is the one that SAMPINT0 stores in interrupt vector 7.

MTRANS.OBJ

MTRANS.OBJ, a PL/M-86 module, computes the value for the counter on the Intel 86/12A or 86/30 single-board computers. This routine assumes that the 153.6-kHz clock is connected to counter 2 of the 8253 integrated circuit on these boards. For this routine to work, the sample interval passed must be between 0.013 and 426.67 ms.

Public Entry Point

ECHK = COUNT$SET (@SAMP$CNT, SAMP$INT)—procedure to compute number of counts for the 8253 counter to generate proper transient-sampling interval

ECHK = 0 if error occurred

ECHK = 1 if interval set properly

@SAMP$CNT—address of word variable that returns number of timer counts

SAMP$INT—real number describing the request sample interval in seconds

Monitor and Debug

MNDMON.OBJ

MNDMON.OBJ, a PL/M-86 module, contains the monitor command interpreter along with the routines to allow the display, filling, and changing of memory locations. The variable types can be byte, word, integer, or real, with the integer data type having the provision for an optional scale factor. In addition, this module contains the command processors to define a dynamic breakpoint.

Public Entry Point

CALL MONITOR—procedure to interpret monitor commands

INTRBK.OBJ

INTRBK.OBJ, an assembly language module sets up the interrupt vectors that implement the dynamic breakpoints. In addition, it contains the routines that check, execute, and reset the dynamic breakpoints.

Input/Output Library

IOCON.OBJ

IOCON.OBJ is a hardware-specific module that drives the RS232 part on the Intel 86/12A or Intel 86/30 boards. This routine provides the logical input/output device that all of the other higher level input/output drivers use. A listing of this routine is provided in appendix F.

Public Entry Points

CALL OUTCON (CHAR)—routine to output "CHAR" byte to console
CHAR = INCON—routine to return ASCII "CHAR" from console

Public Variables

CONCONTR—word variable to determine output device
1 = output to console device only
2 = output to console and list device

MESSAGE.OBJ

MESSAGE.OBJ, an assembly language routine, prints an ASCII string on the console.

Public Entry Point

CALL MESSAGE (@STRING), where STRING is the address of an ASCII character string to be output to the console. The string must be terminated with a null character (i.e., 0).

OUTH.OBJ

OUTH.OBJ, an assembly language routine, outputs variables in hexadecimal format.

Public Entry Points

CALL OUTH (VAR)—outputs a 2-byte memory location in hexadecimal format, where VAR denotes a word variable
CALL OUTHB (VAR)—outputs a 1-byte variable in hexadecimal format, where VAR denotes a 1-byte variable

IOUT.OBJ

IOUT.OBJ, a set of assembly language routines, outputs one and two variables in decimal format.

Public Entry Points

CALL OUTISW(VAR)—outputs a 2-byte variable as a signed number between −32 768 and +32 767, where VAR denotes a 2-byte variable
CALL OUTIUW(VAR)—outputs a 2-byte variable as a signed number between 0 and 65 535, where VAR denotes a 2-byte variable
CALL OUTISB(VAR)—outputs a 1-byte variable as a signed number between −128 and 127, where VAR denotes a 1-byte variable
CALL OUTIUB(VAR)—outputs a 1-byte variable as an unsigned number between 0 and 255, where VAR denotes a 1-byte variable

OUTR.OBJ

OUTR.OBJ, an assembly language routine, outputs a 4-byte memory location as a real number.

Public Entry Point

CALL OUTREAL(VAR)—outputs 4-byte memory location as a real number, where VAR denotes a 4-byte variable

INBUFF.OBJ

INBUFF.OBJ, a set of PL/M routines, does buffered input to the console, controls the console, and reads the console input buffer.

Public Entry Points

CALL INPUT BUFF—inputs ASCII characters from the console by using subroutine INCON and stores them in a

buffer. In addition, this routine permits backspacing, line termination and reset, bell control, etc.

CHAR = READBUFF—accesses the next character in the input buffer and places it in byte variable "CHAR," where CHAR denotes a 1-byte variable

RDATA.OBJ

RDATA.OBJ, an assembly language routine, reads ASCII characters from the input buffer, decides what type of number was input, and calls the proper conversion routine to create the correct core image corresponding to the input string.

Public Entry Points

RCHK = RDATA—same description as RDATA.OBJ, where RCHK denotes a word of returned data corresponding to the type of data that was input. The following six conditions are possible:

0 = error
1 = default
2 = integer between (−32 768 and −1)
4 = integer or word between (0 and 32 767)
8 = word between (32 768 and 65 535)
16 = real number

Public Variables

RWORD, RINT, RREAL—public variables where the RDATA routine places the results of the converted input string.

ININT.OBJ

ININT.OBJ is a set of assembly language routines used by RDATA to do decimal integer or word conversions.

INHEX.OBJ

INHEX.OBJ is a set of assembly language routines used by RDATA to do hexadecimal word conversions.

INREAL.OBJ

INREAL.OBJ is a PLM/86 routine used by RDATA to do real-number conversions.

CPMIO.OBJ

CPMIO.OBJ, a set of assembly language routines, provides a high-level language interface to the functions of the CP/M-86 operating system. Each public entry point and the corresponding CP/M-86 function number listed in reference 2 are described here. Reference 2 should be consulted if further description of an individual function is needed.

Public Entry Points

All of the following routines return a "1" if successful and a "0" if unsuccessful in variable RTRN, except where otherwise noted. Furthermore, the calling arguments @MCB and @FCB are addresses of CP/M-86 memory control and file control blocks, respectively. These control block definitions are defined in reference 2.

CALL RESETCPM (Function 0)—calls the CP/M-86 operating system but does not release the allocated program memory

CALL RESETCPMS (Function 0)—calls the CP/M-86 operating system and releases the allocated program memory

RTRN = RESETDSK (Function 3)—resets the entire disk file system to read/write status

CALL SELDSK (NUM) (Function 14)—selects disk drive "NUM" as currently logged in and defaulted disk drive, where NUM = 1 corresponds to disk A, etc.

RTRN = OPNDSK (@FCB) (Function 15)—opens a currently existing disk file for reading or writing

RTRN = CLOSDSK (@FCB) (Function 16)—closes a currently open disk file

RTRN = SRCHF (@FCB) (Function 17)—searches for first disk directory entry corresponding to FCB

RTRN = SRCHN (Function 18)—searches for next disk directory entry

RTRN = DELETE (@FCB) (Function 19)—deletes a disk file directory entry

RTRN = READSK (@FCB) (Function 20)—reads a 128-byte record from currently open disk file

RTRN = WRITEDSK (@FCB) (Function 21)—writes a 128-byte record to a currently open disk file

RTRN = MAKEDSK (@FCB) (Function 22)—opens a new file on disk for writing

CALL SETDMA (@BUFF) (Function 26)—sets the disk DMA offset address for disk file transfer, where @BUFF denotes address of disk file input/output buffer

CALL ALLOMAP (@SEG, @OFF) (Function 27)—returns segment and offset of current disk allocation map
@SEG = address of location in which to place allocation map segment
@OFF = address of location in which to place allocation map offset

CALL DSKPARMS (@DSKMP) (Function 31)—returns disk parameters for currently logged-in disk, where @DSKMP denotes address of data structure to store disk parameters

RTRN = RESETDRV (NUM) (Function 37)—resets disk drive "NUM" to read/write status, where NUM denotes number corresponding to drive code: NUM = 1 resets disk A, etc.

CALL SETDMAB (@SEG) (Function 51)—sets the disk DMA segment or paragraph address for disk file transfer, where @SEG denotes segment and paragraph address of disk buffer

RTRN = GMAXMEM (@MCB) (Function 53)—returns maximum amount of free memory available
MCB = address of memory control block
RTRN = OFFH if no memory is available; 0 if some memory is available

DISKIO.OBJ

DISKIO.OBJ, a set of PL/M-86 routines, performs high-level disk file control to allow a programmer to interface with the CP/M-86 operating system by reading a file name, opening a file, and reading from and writing to the opened file. It is also possible to display a disk directory and to delete any file from that disk.

Public Entry Points

RTRN = RD FILE NAME (@FCB, @DFCB, DFLT, DSK)—routine to read in a file name from the input buffer or place a defaulted name in proper file control block for later operation
@FCB = pointer to operational file control block
@DFCB = pointer to default file control block that may be used
DFLT = "Y" if default file control block is to be used; "N" if file name is to be read from input buffer
DSK = "A", "B" etc.—corresponds to drive to be designated for disk file

RTRN = ALLOCMEM (@MCB) (Function 55)—allocates memory as per MCB
RTRN = 0 if request is successful; OFFH if request is unsuccessful

CALL FREEMEM (@MCB) (Function 57)—frees allocated memory

RTRN = OPN$FILE$CNM (@FCB,TYPE)—routine to open a disk file with the name specified in FCB
@FCB = pointer to file control block
TYPE = "Y" if new file is to be opened; "N" if old file is to be opened

RTRN = DSKRW (@DATA, @FCB, RLTCH)—routine to input or output a 128-byte record from a disk with the memory address of the record pointed to by @DATA
@DATA = pointer to 128-byte record in memory
@FCB = pointer to file control block
TYPE = "Y" if new file is to be opened; "N" if old file is to be opened
RLTCH = "W" if write to disk is desired, "R" if read from disk is desired

RTRN = HL$DSKIO (@DATA, @FCB, RLTCH, FLTCH, NUM)—routine to read and write 128-byte records to and from a disk file in either absolute format or hexadecimal format
@Data = pointer to 128-byte record memory location
@FCB = pointer to file control block
RLTCH = "W" if write to disk; "R" if read from disk
FLTCH = "A" if absolute data to be read/written; "H" if hexadecimal data to be read/written
NUM = number of words to be output in hexadecimal format, with nulls being padded for the rest

Appendix C
Error Summary

Error	Explanation
A	data element already defined in master table
B	undefined or incorrect data type in input sequence. This error is usually generated by the input/output library
C	steady-state or transient data tables closed
D	invalid default attempted in command sequence
E	transient sampling parameters not set
F	master data element definition table or currently open steady-state or transient sampling table full
G	improper command sequence for overlay
H	improper argument sequence in command string
I	inproper steady-state data collection command sequence
J	requested steady-state transient data table has no elements defined in it
K	input number type incompatible with type of number required for command sequence
L	input too large to convert to an integer
M	unable to open a disk file on read
N	variable not defined in master definition table
O	open table error (1) Table number not between 1 and 4 (2) Another table (steady state or transient) open
P	unable to open disk file on write
Q	transient sample size requested is too large for current memory configuration
R	read error on disk
S	memory allocation error in CP/M-86
T	transient sampling armed; no changes possible in variable or table definitions at this time
U	undefined command request
V	error code not currently implemented
W	write error on disk
X	steady-state or transient data table already empty
Y	variable types not matched for proper transient overlay
Z	close file error in CP/M-86
1	transient sampling parameters not set
2	memory configuration cannot accommodate present number or mix of variables
3	not enough memory for present number of variables at current sample length
4	transient sampler not armed
5	printout requested for an undefined transient sample
6	length of variable name requested or table length requested not compatible with current program version
7	table version requested for read-in not compatible with current version of program
8	table format (hexadecimal or absolute) not compatible with format requested
9	table format (steady-state or transient) not compatible with format requested

Appendix D
Users Command Summary

Main Static Data Display Section

Data Element Definition and Display Commands

Command	Description
NAME TY,SEG,OFF,SFN,SFD	teaches the program a data element definition (NAME denotes name (up to six letters) of a selected location; TY denotes type of data element (I = integer; W = word; R = real); SEG denotes a data element memory segment; OFF denotes a data element memory offset; SFN denotes a numerator scale factor; and SFD denotes denominator scale factor). Note that SFN and SFD can be replaced directly by one real number representing the scale factor if only the SFN entry is made. The TY, SEG, OFF, SFN, and SFD parameters can all be defaulted, if desired, to the previous element's definition. If offset is defaulted, the appropriate offset is computed from the previous one depending on its variable type
NAME1,NAME2,TY,SEG,OFF, SFN,SFD	overlays "NAME1" data element definition in master table with "NAME2." Defaults can be used but offset is updated from "NAME1"
NAME=	displays value of data element "NAME" in engineering units. The character "=" can be entered by itself to display the value of the previously referenced data element.
NAME#	displays value of a word or integer data element in binary units. The character "#" can be entered by itself to display the value of the previously referenced data element.
NAME 'P1	sets value of data element "NAME" to P1. P1 must be a real number if NAME is real. If P1 is a word, P1 must be between 0 and 65 535. If P1 is an integer, P1 divided by the scale must be between −32 768 and 32 767.
NAME&	displays definitions parameters of data element "NAME"
?	displays master table and all data element definitions
[	displays current segment registers being used by the executive/control algorithm/MINDS program load
%	allows alteration of segment register values in the master data element definition table

Steady-State Data Table Manipulation and Collection Commands

Command	Description
(P1	opens a steady-state data table for input of data elements. (P1 denotes table numbers 1 to 4.)
)	closes current open steady-state data table
@	erases all definitions from current open steady-state data table
]	starts new output line in current open steady-state data table
\P1,P2,P3	outputs a data table to screen in desired format. P1 denotes collection method, specified as a letter A to D (method A: all data are integer or word and all data lie in same data segment; method B: all data are integer or word but can be in different data segments; method C: data can be integer, word, or real but all data must lie in same data segment; method D: data can be integer, word, or real and can be in different data segments.) P2 denotes number of

	table to output to CRT; must be between 1 and 4. P3 denotes format of table to be output (1—output heading and data; 2—output only data; 3—output only heading). Note that parameters P1, P2, and P3 can be defaulted to their previous values by using commas.
NAME,	puts variable "NAME" in current open data table
!D:FILENAME	saves all data element definitions along with steady-state and transient table definitions in a CP/M-compatible file (D denotes disk A or B; FILENAME denotes CP/M-compatible file name with optional extension). Disk and file name can be defaulted.
*D:FILENAME	recalls data element definition and steady-state or transient table information from a CP/M-compatible file. (D denotes disk A or B; FILENAME denotes CP/M-compatible file name with optional extension). No disk or name default is allowed in this command since more than one data element definition could be stored on a disk.

Automatic Static Data Collection and Retrieval Commands

Command	Description
\P1,0	special version of "\" command to collect the values of all defined data elements but not print them out; that is, to collect table 0. This command should be used in conjunction with the "$" command for automated data taking. (P1 denotes data collection methods A to D). P1 and 0 can be defaulted after the first entry.
$P1,P2	stores the respective collected data in a CP/M-compatible file with a defaulted name. The defaulted name depends on the type of data being stored (transient or steady state) and the reading number. (P1 = S means store steady-state data; P1 = T means store transient data; P2 = A means store data in absolute format; P2 = H means store data in hexadecimal format.) P1 and P2 can be defaulted to their previous values.
\D:FILENAME,P1	recalls steady-state data from a CP/M-compatible file. (D denotes disk A or B; FILENAME denotes CP/M-compatible file name with optional extension; P1 = A if data are stored in absolute format; P1 = H if data are stored in hexadecimal format.) No disk or name default allowed since a large number of data files may exist on the same disk.
;P1,P2	prints the recalled steady-state data. Command is used in conjunction with the "/" command to display stored data that have been taken previously. (P1 denotes number of table to be output to CRT; must be between 1 and 4. P2 denotes format of table to be output: 1 = output headings and data; 2 = output data only; 3 = output headings only).
	updates transient and steady-state reading numbers for default file names assigned in "$" command. This command also updates the default reading number for the MINDS data element definition tables. This number is used in conjunction with a default in the "!" command.

Miscellaneous Commands

Command	Description
>D	displays disk directory and remaining free space of selected disk. (D denotes disk A or B.)
{D:FILENAME	deletes CP/M file from selected disk. (D denotes disk A or B; FILENAME denotes CP/M-compatible file name with optional extension.) No disk or name default allowed for safety reasons.

:H1,H2	adds or substracts two numbers. (H1,H2 denotes two hexadecimal or decimal integers or words.)
<	calls monitor/debug package
-	calls transient data sampling package
.	exits main command structure and returns to main calling program (i.e., real-time control executive)

Transient Sampling Package

Transient Data Table Definition Commands

Command	Description
(P1	opens transient data table for input of data elements. (P1 denotes table number; must be between 1 and 4.)
)	closes current transient data table
NAME,	puts data element "NAME" in current open data table
NAME1_NAME2	overlays "NAME1" data element in current open transient data table with "NAME2"
^	deletes last data element defined in current open transient data table
@	deletes all data element entries in current open transient data table
?P1	displays selected transient sampling table data element members. (P1 denotes table number; must be between 1 and 4.)

Transient Sampling Parameter Commands

Command	Description
%	sets transient sampling parameters (sample interval, sample length)
&	displays current sampling parameters and current sampling memory allocation
[	displays maximum memory available for storage of transient data. The value displayed by this command is the same whether a transient table is armed or not.
]	displays memory left for further storage of transient data. The value displayed by this command is equal to that displayed by "[" if a transient data table is unarmed. However, if a table is armed, this command only displays the amount of free memory actually left.

Transient Data Collection Commands

Command	Description
{P1	arms a transient data table in preparation for a transient sample. (P1 denotes table number; must be between 1 and 4.) This command allocates the proper amount of memory for each variable in the respective table.
}	disarms a transient data table so sampling parameters, table data element members, or table requested for sampling can be changed
\	activates currently armed transient data table and takes transient data

Transient Data Storage Commands

Command	Description
$P1,P2	stores respective collected data in CP/M-compatible file with defaulted name. The defaulted name depends on type of data being stored (transient or steady-state and reading number). (P1 = S stores steady-state data; P1 = T stores transient data; P2 = A stores data in absolute format; P2 = H stores data in hexadecimal format.) P1 and P2 can be defaulted to previous values.
"	updates transient and steady-state reading numbers for default file names assigned in the "$" command. This command also can update the default reading number for the data element definition tables. This number is used in conjunction with a default in the "!" command
*D:FILENAME,P1	recalls transient data from CP/M-compatible file. (D denotes disk A or B; FILENAME denotes CP/M-compatible file name with optional extension; P1 = A if data stored in absolute data format; P1 = H if data stored in hexadecimal data format.) No disk or name default allowed since a large number of data files may exist on the same disk. The transient table definition corresponding to the respective transient must be armed for the recall to take place.
;P1	prints transient data at console of respective variable number in currently armed transient table. (P1 denotes transient variable number in currently armed table.) If P1 is defaulted, all variables in currently armed transient table will be printed.

Transient Package Miscellaneous Commands

Command	Description
.	returns to main steady-state collection program

Monitor/Debug Package

Standard Monitor Memory Display and Set Commands

Command	Description
.	returns to main steady-state data display program
H V1,V2	adds or substracts V1 and V2 (V1,V2 denote two hexadecimal or decimal integers or words)
D T,SEG,OFF1,OFF2,SFN,SFD	displays main memory from OFF1 to OFF2. (T denotes data type: byte, word, integer, or real; SEG denotes data segment; OFF1 denotes starting data offset; OFF2 denotes ending data offset; SFN denotes numerator scale factor; SFD denotes denominator scale factor.) SFN and SFD are ignored unless variable type is integer. Variable type and segment can be defaulted; however, if T is defaulted, byte data are assumed.
F T,SEG,OFF1,OFF2,DATA	fills data from SEG:OFF1 to SEG:OFF2. (T denotes data type: byte, word, or real; SEG denotes data segment; OFF1 denotes starting data offset; OFF2 denotes ending data offset; DATA denotes data to be filled.) If T is defaulted, byte data are assumed.
S T,SEG,OFF,SFN,SFD	sets memory locations with input data. (T denotes data type: byte, word, integer, or real; SEG denotes data segment; OFF denotes data offset; SFN

	denotes scale factor numerator; SFD denotes scale factor denominator.) If T is defaulted, byte data are assumed. SFN and SFD are ingored except for integer data. Inputting a comma will not change current memory location but will increment to next location. Input is terminated by entering an error.

Breakpoint Set, Display, and Execute Commands

Command	Description
R	displays current segment registers of load program; used to provide information for setting breakpoints
B SEG,OFF,OPT,TABC,NAME <=>VAL	sets a dynamic breakpoint at a particular location with a set of characteristics (SEG denotes breakpoint segment address; OFF denotes breakpoint offset address; OPT denotes breakpoint option (0 = collect 8086 registers; 1 = collect 8086 registers and table 1; 2 = collect 8086 registers and table 2; 3 = collect 8086 registers and table 3; 4 = collect 8086 registers and table 4); TABC denotes table collection method A to D. See command "\" for complete explanation of data collection methods; NAME denotes name of any word, integer, or real data element to be examined for a conditional break; <=> means that less than, equal to, or greater than are the three permissible conditions; VAL denotes appropriate value for the conditional.) If "NAME" parameter is omitted, an immediate breakpoint is set (i.e., a breakpoint is initiated as soon as the location is reached). SEG, OFF, and TABC can be defaulted.
G	commands to begin above breakpoint. Current conditional in force and breakpoint location are printed out.
X	displays collected data registers
T P1	outputs collected data table (P1 denotes table output formats 1 to 3 described in the "\" table output command.)

Keyboard Commands

Command	Description
^H	backspaces input buffer
^G	turns bell on or off
^X	deletes entire input line
^P	turns optional output to line printer on or off
ANY KEY	striking any key during output will terminate printing

Appendix E
Data Table Format

The data collection program can store three types of data table on disk. These data tables include parameter definition tables, steady-state data tables, and the transient data tables. The steady-state and transient data tables are output to the disk in CP/M format with default names STDYXXXX.DAT and TRANXXXX.DAT. The XXXX stands for a number between 0 and 9999 that is automatically incremented every time a table is written. In addition, the data tables can be output to the disk in either hexadecimal or absolute format. The absolute format takes up less space and stores the memory image of the steady-state or transient data in a CP/M file.

The hexadecimal format converts each memory byte into a 2-byte ASCII representation of the contents of the memory location. In addition, it adds carriage return and line feed characters at appropriate places in the data to create data records of the proper length. This hexadecimal file can then be uplinked directly into a mainframe computer for data reduction and plotting since all of the characters are ASCII between "0" and "F." Therefore, no spurious control codes will be given to the mainframe computer. The data format for data writing and readback is chosen in the data storage and retrieval commands.

The parameter definition tables are handled in a different manner than the data tables. The parameter definition tables can only be written to the disk in absolute format since there will never be any necessity to uplink them to a mainframe computer. The name of the disk file to which the information is being written may be defaulted to MINDXXXX.TAB or may be chosen by the user. As with the steady-state and transient data files the XXXX number is incremented every time a file is written or modified to an appropriate user-selected value by using the "" command.

The format of each of the three data tables is given here. All of the data in the file header and transient preamble are automatically stored.

Transient Data Tables

Record 1—file type record

(1) Space
(2) File type (H = hexadecimal file; F = absolute file)
(3) Check sum

Record 2—transient parameter record

(1) Transient number
(2) Number of transient variables in file
(3) Total number of memory paragraphs used for transient sampling (paragraph = 16 bytes)
(4) Number of sample points per variable
(5) Number of sample paragraphs per real variable
(6) Number of sample paragraphs per word variable
(7) Counts for sampling interval timer
(8) Check sum

Record 3—transient data record preamble for variable 1

(1) Integer, word, or real variable flag (27 = integer or word transient; 22 = real transient)
(2) Full-scale scale factor (SF)
(3) Full-scale modifier (SM); i.e., full scale = SF *2 *SM
(4) Name of transient variable
(5) Check sum

Scale factor and full-scale modifier only apply if integer data are being written. If data are word or real, the scale factor is zero (SF = 0).

Record 4—transient data record for variable 1

For an integer or word variable, one record consists of 30 data points (i.e., 60 bytes) and a check sum. For a real variable, one record consists of 15 data points (i.e., 60 bytes) and a check sum.

A series of type 4 records are written until the entire transient trajectory has been written to the file. For the next transient variable a type 3 record is written along with the requisite number of type 4 records to completely output that transient. This procedure is followed until all of the transient variables have been written.

Steady-State Data Tables

Record 1—file type record

(1) Space
(2) File type (H = hexadecimal file; F = absolute file)
(3) Check sum

Record 2—steady-state parameter record

(1) OFFFFH
(2) Steady-state reading number
(3) Number of steady-state variable values collected
(4) Size of steady-state data collection buffer

Record 3—steady-state parameter data element values

Integer, word, and real numbers are collected and output in a series of records. Each record consists of 60 bytes and a check sum. The variables are collected and stored in the same order as they are defined in the master table.

Parameter Definition Tables

Record 1—Data table version number

Record 2

(1) Steady-state data table partitions
(2) Number of variables stored in each steady-state data table
(3) Free
(4) Number of variables defined in master table
(5) Maximum size of each name
(6) Maximum size of table to define data element names
(7) Maximum size of data element definition tables
(8) Steady-state data table print parameter

Record 3

(1) Transient data table partitions
(2) Number of variables stored in each variable data table
(3) Number of paragraphs necessary to store a word data element transient sample
(4) Number of paragraphs necessary to store a real data element transient sample
(5) Transient parameter set latch
(6) Total number of memory paragraphs used for transient sampling
(7) Counts for sampling interval timer

Record 4

(1) Total transient sampling time
(2) Transient sampling interval

Records 5 to 16

Store array of data element definition names (NAME$TAB).

Records 17 to 24

Store array of data element scale factors (SF$TAB).

Records 25 to 32

Store array of data element offsets (OFF$TAB).

Records 33 to 40

Store array of data element segment (SEG$TAB).

Records 41 to 42

Store array of data element types (VAR$TYPE$TAB).

Records 43 to 54

Store array of steady-state data table definitions (DATA$TAB).

Records 55 to 58

Store array of transient data table definitions (TRANTABNAME).

Appendix F
Routines for Console Input and Output

```
SERIES-III 8096/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE IOCON
OBJECT MODULE PLACED IN :F2:IOCON.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F2:IOCON.SRC XREF


LOC  OBJ                  LINE     SOURCE

                             1 +1  $TITLE('IOCON-- ROUTINES FOR CONSOLE INPUT/OUTPUT')
                             2     ;
                             3     ;
                             4     ;
                             5     ;       IOCON IS A SET OF TWO PROCEDURES TO DO CONSOLE,
                             6     ;       LIST AND GRAPHICS DEVICE INPUT AND OUPUT.
                             7     ;
                             8     ;       INCON--DOES CHARACTER INPUT FROM THE CONSOLE
                             9     ;
                            10     ;       OUTCON--DOES CHARACTER OUTPUT TO THE CONSOLE DEVICE
                            11     ;               LISTING DEVICE AND GRAPHICS DEVICE DEPENDING
                            12     ;               ON THE SETTING OF THE VARIABLE CONCONTR
                            13     ;
                            14     ;               CONCONTR = 1 OUTPUT TO CONSOLE DEVICE ONLY
                            15     ;               CONCONTR = -1 OUTPUT TO CONSOLE AND LIST DEVICES
                            16     ;               CONCONTR = 2 OUTPUT TO GRAPHICS DEVICE
                            17     ;
                            18     ;       THE ROUTINES AS THEY NOW STAND ARE CUSTOMIZED TO DO
                            19     ;       HARDWARE DEPENDENT CONTROL OF THE SERIAL PORT ON THE
                            20     ;       INTEL SBC 86/12 OR 86/30 BOARDS.  THESE TWO BOARDS BOTH
                            21     ;       USE AN 8259 FOR SERIAL PORT CONTROL.  THESE ROUTINES DRIVE
                            22     ;       THE 8259 INTERFACE DIRECTLY, HOWEVER, THEY COULD BE CONFIGURED
                            23     ;       TO WORK THROUGH THE BIOS IN THE CP/M-86 OPERATING SYSTEM.
                            24     ;       THE INTERFACE TO THE LISTING PORT AND THE GRAPHICS PORT USE
                            25     ;       PORTS 1 AND 2 RESPECTIVELY ON THE INTEL ISBC 534 SERIAL CARD
                            26     ;
                            27     ;       NOTE: SET-UP OF THE CONSOLE, LISTING AND GRAPHICS PORT ON THE
                            28     ;             THE SBC 86/30 AND SBC 534 BOARDS IS ASSUMED TO BE
                            29     ;             DONE BY THE CP/M-86 OPERATING SYSTEM
                            30     ;
                            31     ;
                            32     ;       CALLING SEQUENCE FOR CHARACTER OUPUT ROUTINE 'OUTCON'
                            33     ;       AND CHARACTER INPUT ROUTINE 'INCON'.
                            34     ;
                            35     ;       PUSH WORD (BYTE) TO BE OUTPUT
                            36     ;               CALL OUTCON
                            37     ;
                            38     ;
                            39     ;               CHAR = IOCON
                            40     ;       CHARACTER RETURNS IN THE AL REGISTER IF ASSEMBLY LANGUAGE
                            41     ;       USED
                            42     ;
                            43     ;
                            44     ;;      NOTE: ROUTINE DESTROYES THE AX, CX, AND DX REG
```

```
                        45      ;
                        46      ;*****************************************************************
                        47 +1   $EJECT
                        48      ;
                        49      ;
                        50              NAME    IOCON
                        51      CGROUP  GROUP   CODE
                        52      DGROUP  GROUP   DATA
                        53              ASSUME  CS:CGROUP,DS:DGROUP
                        54              PUBLIC  OUTCON,CONCONTR,INCON
                        55              EXTRN   INITIAL1:NEAR
                        56      ;
                        57      ;
                        58      ;       DEFINE EXTERNAL I/O PORTS
                        59      ;
                        60      ;       CONSOLE I/O PORTS
00DA                    61      OUT_CON_STAT    EQU     0DAH    ; OUTPUT STATUS PORT ADDRESS ON 8251
00D8                    62      OUT_CON_CHAR    EQU     0D8H    ; CHARACTER OUTPUT PORT ADDRESS ON 8251
00DA                    63      IN_CON_STAT     EQU     0DAH    ; INPUT STATUS PORT ADDRESS ON 8251
00D8                    64      IN_CON_CHAR     EQU     0D8H    ; CHARACTER INPUT PORT ADDRESS ON 8251
                        65      ;
                        66      ;
                        67      ;
                        68      ;       LIST I/O PORTS
00A1                    69      OUT_LIST_STAT   EQU     0A1H    ; LIST STATUS PORT
00A0                    70      OUT_LIST_CHAR   EQU     0A0H    ; LIST CHARACTER PORT
                        71      ;
                        72      ;       GRAPHICS OUTPUT PORT DEVICES
00A3                    73      GRAPH_OUT_STATUS        EQU     0A3H    ; GRAPHICS STATUS PORT
00A2                    74      GRAPH_OUT_DATA          EQU     0A2H    ; GRAPHICS DATA PORT
                        75      ;
                        76      ;
----                    77      DATA    SEGMENT PUBLIC  'DATA'
0000 0100               78      CONCONTR DW     1               ; CONSOLE/LIST DEVICE CONTROL WORD
                        79      ;
----                    80      DATA    ENDS
                        81 +1   $EJECT
                        82      ;
----                    83      CODE    SEGMENT PUBLIC  'CODE'
                        84      ;
0000                    85      OUTCON:
0000 55                 86              PUSH    BP              ; SAVE BP REGISTER
0001 8BEC               87              MOV     BP,SP
                        88                                      ; DO CHARACTER OUTPUT TO THE
                        89                                      ; CONSOLE DEVICE
                        90      ;
0003 A10000         R   91              MOV     AX,CONCONTR     ; GET CONSOLE CONTROL FLAG
0006 3D0200             92              CMP     AX,2            ; CHECK IF TEK OUTPUT DESIRED
0009 744A               93              JE      GO              ; GO TO TEK OUTPUT
                        94      ;
000B                    95      LOOP1:
000B BADA00             96              MOV     DX,OUT_CON_STAT ; GET CONSOLE STATUS PORT ADDRESS
000E EC                 97              IN      AL,DX           ; GET CONSOLE STATUS
000F 2401               98              AND     AL,1            ; CHECK IF CONSOLE IS FREE
```

```
0011 74F8                99          JZ      LOOP1           ; LOOP TILL CONSOLE BUFFER EMPTY
0013 BAD800              100         MOV     DX,OUT_CON_CHAR ; LOAD CHARACTER OUTPUT PORT ADDRESS
0016 8A4604              101         MOV     AL,[BP+4]       ; GET OUTPUT CHARACTER FROM STACK
0019 EE                  102         OUT     DX,AL           ; OUTPUT CHARACTER TO CONSOLE
                         103     ;
                         104     ;
                         105                                 ; CHECK IF LINE PRINTER OUTPUT
                         106                                 ; REQUESTED
001A A10000        R     107         MOV     AX,CONCONTR     ; MOV CONSOLE CONTROL LATCH TO AX
001D 3D0000              108         CMP     AX,0            ; CHECK IF SET FOR PRINTER OUTPUT
0020 7F0B                109         JG      OUT1            ; JUMP IF NO PRINTER OUTPUT REQUESTED
                         110     ;
                         111                                 ; PRINTER OUTPUT DRIVER
0022                     112     LST1:
                         113     ;
0022 E4A1                114         IN      AL,OUT_LIST_STAT        ; GET LISTING DEVICE STATUS
0024 2401                115         AND     AL,1                    ; CHECK LISTING DEVICE STATUS
0026 74FA                116         JZ      LST1                    ; STATUS NOT READY CHECK AGAIN
0028 8A4604              117         MOV     AL,[BP+4]               ; GET CHARACTER TO BE OUTPUT
002B E6A0                118         OUT     OUT_LIST_CHAR,AL        ; OUTPUT CHARACTER
                         119     ;
                         120     ;
                         121                                 ; ROUTINE TO DETECT CHARACTER
                         122                                 ; PRESENT DURING INPUT.  THIS
                         123                                 ; ROUTINE ABORTS FURTHER OUTPUT
                         124                                 ; AND REINITIALIZES THE PROGRAM
                         125     ;
002D E4DA                126     OUT1:   IN      AL,IN_CON_STAT  ; INPUT CONSOLE STATUS
002F 2402                127         AND     AL,2            ; CHECK IF CHARACTER PRESENT
0031 7405                128         JZ      OUT2            ; NO CHARACTER CONTINUE
0033 E4D8                129         IN      AL,IN_CON_CHAR  ; CLEAR CHARACTER BUFFER
0035 E80000        E     130         CALL    INITIAL1        ; CALL REINITAILIZATION SEQUENCE
                         131     ;
0038 5D                  132     OUT2:   POP     BP              ; RESTORE BP REGISTER
0039 C20200              133         RET     2               ; RETURN TO CALLING PROGRAM
                         134 +1  $EJECT
                         135     ;
003C                     136     INCON:                          ; CONSOLE INPUT ROUTINE TO
                         137                                     ; TO ALLOW CHARACTER INPUT
                         138                                     ; FROM KEYBOARD AND ECHO BACK
                         139                                     ; TO CONSOLE
                         140     ;
003C 55                  141         PUSH    BP              ; STORE BP REGISTER
003D 8BEC                142         MOV     BP,SP           ; GET STACK POINTER
                         143     ;
003F                     144     LOOP2:
003F BADA00              145         MOV     DX,IN_CON_STAT  ; GET INPUT CONSOLE STATUS ADDRESS
0042 EC                  146         IN      AL,DX           ; GET INPUT CONSOLE STATUS
0043 2402                147         AND     AL,2            ; CHECK IF CHARACTER PRESENT
0045 74F8                148         JZ      LOOP2           ; LOOP TILL CHARACTER PRESENT
                         149     ;
0047 BAD800              150         MOV     DX,IN_CON_CHAR  ; GET CHARACTER INPUT PORT ADDRESS
004A EC                  151         IN      AL,DX           ; INPUT CHARACTER
004B 247F                152         AND     AL,7FH          ; STRIP OFF PARITY BIT
```

```
                        153     ;
                        154                                     ; ECHO CHARACTER BACK TO CONSOLE
004D 50                 155             PUSH    AX              ; SAVE CHARACTER FOR RETURN
004E 50                 156             PUSH    AX              ; LOAD CHARACTER FOR ECHO BY OUTCON
004F E8AEFF             157             CALL    OUTCON          ; OUTPUT CHARACTER TO CONSOLE
0052 58                 158             POP     AX              ; GET CHARACTER FOR RETURN TO CALLING
                        159                                     ; PROGRAM
0053 5D                 160             POP     BP              ; RESTORE BP
0054 C3                 161             RET                     ; RETURN TO CALLING PROGRAM
                        162 +1  $EJECT
                        163     ;
                        164
                        165                                     ; ROUTINE TO OUTPUT CHARACTERS TO
                        166     ;                               ; TO THE TEKTRONIX GRAPHICS TERMINAL
0055                    167     GD:
0055 E4A3               168             IN      AL,GRAPH_OUT_STATUS     ; GET SERIAL PORT STATUS
0057 2401               169             AND     AL,1            ; CHECK IF PORT EMPTY
0059 74FA               170             JZ      GD              ; IF NOT EMPTY CHECK AGAIN
005B 8A4604             171             MOV     AL,[BP+4]       ; GET CHARACTER TO BE OUTPUT
005E E6A2               172             OUT     GRAPH_OUT_DATA,AL       ; OUTPUT CHARACTER
0060 5D                 173             POP     BP              ; RESTORE BP REGISTER
0061 C20200             174             RET     2H              ; RETURN TO ROUTINE START
----                    175     CODE    ENDS
                        176             END
```

XREF SYMBOL TABLE LISTING

```
NAME                TYPE     VALUE  ATTRIBUTES, XREFS

??SEG . . . . . SEGMENT         SIZE=0000H PARA PUBLIC
CGROUP. . . . . GROUP           CODE  51# 53
CODE. . . . . SEGMENT         SIZE=0064H PARA PUBLIC 'CODE'  51# 83 175
CONCONTR. . . . V WORD   0000H  DATA PUBLIC  54 78# 91 107
DATA. . . . . SEGMENT         SIZE=0002H PARA PUBLIC 'DATA'  52# 77 80
DGROUP. . . . . GROUP           DATA  52# 53
GD. . . . . L NEAR   0055H  CODE  93 167# 170
GRAPH_OUT_DATA. . NUMBER   00A2H    74# 172
GRAPH_OUT_STATUS. NUMBER   00A3H    73# 168
IN_CON_CHAR . . . NUMBER   00D8H    64# 129 150
IN_CON_STAT . . . NUMBER   00DAH    63# 126 145
INCON . . . . . L NEAR   003CH  CODE PUBLIC  54 136#
INITIAL1. . . . . L NEAR   0000H  EXTRN  55# 130
LOOP1 . . . . . L NEAR   000BH  CODE  95# 99
LOOP2 . . . . . L NEAR   003FH  CODE  144# 148
LST1. . . . . L NEAR   0022H  CODE  112# 116
OUT_CON_CHAR. . . NUMBER   00D8H    62# 100
OUT_CON_STAT. . . NUMBER   00DAH    61# 96
OUT_LIST_CHAR . . NUMBER   00A0H    70# 118
OUT_LIST_STAT . . NUMBER   00A1H    69# 114
OUT1. . . . . L NEAR   002DH  CODE  109 126#
OUT2. . . . . L NEAR   0038H  CODE  128 132#
OUTCON. . . . . L NEAR   0000H  CODE PUBLIC  54 85# 157

END OF SYMBOL TABLE LISTING

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

References

1. Posa, John G.; and LeBoss, Bruce: Intel Takes AIM at the '80s. Electronics, vol. 53, no. 5, Feb. 28, 1980, pp. 89-95.
2. CP/M-86 Operating System User's Guide. Digital Research, 1981.
3. Cwynar, David S.: INFORM—An Interactive Data Collection and Display Program with Debugging Capability. NASA TP-1424, 1980.
4. PL/M-86 Programming Manual. Intel Corp. (Manual Order No. 9800466A), 1978.
5. Morse, Stephan P.: The 8086/8088 Primer, an Introduction to Its Architecture, System Design, and Programming. Second Ed., Hayden Book Company, Inc., 1982.
6. The 8086 Family User's Manual, Numerics Supplement. Intel Corp. (Manual Order No. 121586-001), July 1980.
7. The iSBC 86/12A Single Board Computer Hardware Reference Manual. Intel Corp. (Manual Order No. 9803074-01), 1979.
8. Intel Multibus Specifications. Intel Corp. (Manual Order No. 9800683), 1978.
9. Delaat, John C.; and Soeder, James F.: Design of a Microprocessor-Based Control, Interface, and Monitoring (CIM) Unit for Turbine Engine Controls Research. NASA TM-83433, 1983.
10. Miller, Alan R.: Mastering CP/M. Sybex Corp., 1983.
11. Blech, R.A., et al.: A Real-time, Portable, Microcomputer-Based Jet Engine Simulator. NASA TM-83550, 1984.
12. Baez, A.: Design Description of Microprocessor-Based Engine Monitoring and Control Unit (EMAC) for Small Turboshaft Engines. NASA TM-86860, 1984.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TP-2378		

4. Title and Subtitle	5. Report Date
MINDS - A Microcomputer Interactive Data System for 8086-Based Controllers	January 1985
	6. Performing Organization Code: 505-43-3

7. Author(s)	8. Performing Organization Report No.
James F. Soeder	E-2172
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	
	13. Type of Report and Period Covered: Technical Paper
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

A microcomputer interactive data system (MINDS) software package for the 8086 family of microcomputers is described. To enhance program understandability and ease of code maintenance, the software is written in PL/M-86, Intel Corporation's high-level system implementation language. The MINDS software is intended to run in residence with real-time digital control software to provide displays of steady-state and transient data. In addition, the MINDS package provides classic monitor capabilities along with extended provisions for debugging an executing control system. The software uses the CP/M-86 operating system developed by Digital Research, Inc., to provide program load capabilities along with a uniform file structure for data and table storage. Finally, a library of input and output subroutines to be used with consoles equipped with PL/M-86 and assembly language is described.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Microprocessors Digital control Data systems	Unclassified - unlimited STAR Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	39	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161 NASA-Langley, 1985

National Aeronautics and
Space Administration

Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

THIRD-CLASS BULK RATE

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

NASA

POSTMASTER: If Undeliverable (Section 158
Postal Manual) Do Not Return