MINC-11

Book 6:
MINC Lab Module Programming

November 1978

This book describes program routines that control the MINC lab modules. Part 1 explains conventions and definitions in lab module programming. Part 2 contains detailed reference descriptions for the routines.

Order Number AA-D575A-TC

MINC-11

VERSION 1.0

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, November 1978

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
MINC-11	DECSYSTEM-2020	

CONTENTS

PART 2 ROUTINES

FIGURES

TABLES

PART 1

INTRODUCTION TO LAB MODULE PROGRAMMING

CHAPTER 1
LAB MODULE CAPABILITIES

LAB MODULE ROUTINES

This manual describes the routines that control the MINC lab modules. Programs use these routines to control data transfer from an instrument to the MINC workspace or vice versa.

This manual contains two major parts. Part 1 describes the operations performed by the lab module routines, fundamental programming concepts in transfer and control, the terminology and conventions used throughout the manual, and some program examples. Part 2 describes how to use the routines; these descriptions are arranged alphabetically for convenient reference use.

With the appropriate system configuration, the lab module routines provide six general classes of capabilities. These capabilities are analog signal processing, digital sampling and control, measuring time intervals, controlling processes, transferring ASCII characters, and providing maintenance functions.

The following paragraphs briefly describe each of these general capabilities.

Analog Signal Processing

Analog signal processing involves data transfer to and from analog instruments. "Analog" refers only to the characteristics of the signal and instrument. That is, the value of an analog signal is continuously varying and capable of assuming any of an infinite number of values within a small range. A digital value is capable of assuming a limited number of discrete values within a small range. MINC represents the data internally in digital form, so that analog data transfers require conversion between the analog value and the digital representation of the value.

Input MINC routines read the values of analog signals from instruments in a process called *analog-to-digital (A/D) conversion.* For example, voltage values are analog signals that MINC can read. The program can either hold the values in the workspace for immediate analysis or store them on a storage volume for a future program to analyze.

Three routines control analog input: AIN (*A*nalog *IN*put), AIN_HIST (*A*nalog *IN*put *HIST*ogram), and AIN_SUM (*A*nalog *IN*put *SUM*mation).

AIN collects input from instruments connected to the A/D converter. It can collect a single data value (a *point*), a specified number of values (a *sweep*), or any number of values where the number is not specified in advance (a *stream*). AIN_HIST collects a sweep of analog values and generates a histogram using the values. AIN_SUM collects multiple sweeps and accumulates the values, a process commonly known as signal averaging.

AIN and AIN_SUM can control the sampling process in several ways. In one case, a trigger line from the instrument itself causes each conversion. In another case, they use the clock module to control the input, collecting values at regular intervals. In another case, they use periodic signals from an external source to control the sampling.

Output MINC routines send values in the workspace to instruments that accept analog signals in a process called *digital-to-analog (D/A) conversion.* Storage oscilloscopes and analog plotters are examples of instruments which accept analog signals.

AOUT can control the output process in a variety of ways. In one case, called burst output, AOUT sends the values as fast as the D/A converter is capable of converting them. In another case, AOUT uses the clock module to control the output, sending values at regular intervals. In another case, AOUT uses periodic signals from a source other than the D/A converter or the clock module to control the D/A conversion process. These signals can come from a function generator connected to Schmitt trigger 1 (ST1) of the clock module.

Analysis MINC routines can perform a Fast Fourier Transform (FFT), an inverse transform (FFT) or a power spectrum analysis (POWER) on computed values or on data acquired by A/D conversion routines. The program can transfer the results of the analysis to a storage volume, display the results using

MINC graphic routines, or output the results to analog instruments using AOUT.

Digital Sampling and Control

Digital sampling and control involves data transfer between MINC and digital instruments or digital switches. Digital apparatus produces or responds to a limited number of discrete values, typically binary logic values.

Each MINC digital transfer unit consists of 16 lines. Digital data transfers can transfer the values of all lines simultaneously — for example, to or from BCD instruments. They can also control the states of individual lines independently for reading status lines or setting control switches.

Input Four routines control input from the digital units: DIN (*D*igital *IN*put), DIN_MASK (*D*igital *IN*put *MASK*), DIN_EVENT (*D*igital *IN*put *EVENT* enable) and TEST_LINE (*TEST* input *LINE*). DIN collects data from digital input units in either single points, sweeps, or continuous streams. DIN can record the time when values are read from the unit. In combination with DIN_MASK, DIN can screen out unwanted signals. In combination with DIN_EVENT, DIN can respond immediately to an instrument's change in state. The preceding routines read all 16 lines simultaneously. TEST_LINE can determine the state of an individual line.

Output Three routines control output to the digital units: DOUT (*D*igital *OUT*put), DOUT_MASK (*D*igital *OUT*put *MASK*), and SET_LINE (*SET* output *LINE* condition). DOUT sends values to instruments under clock control or under control from the instruments. In combination with DOUT_MASK, DOUT can send signals only to relevant lines. The preceding routines send information to all 16 output lines simultaneously. SET_LINE specifies the condition for an individual output line.

Measuring Time Intervals

MINC routines can measure the time elapsed between two events. Either of the events can be internal (caused by the program) or external (caused by an instrument).

START_TIME and GET_TIME provide capabilities for measuring elapsed time — for example, the interval between sending a start signal and receiving a response. DIN, in combination with MAKE_TIME, can automatically measure elapsed time during digital sampling.

TIME_HIST (*TIME* interval *HIST*ogram) and PST_HIST (*P*ost-*S*timulus *T*ime *HIST*ogram) measure the time intervals

between signals from instruments and construct histograms based on the interval data.

Controlling Processes

MINC routines can coordinate program execution with the time of day or time intervals (time events) or with signals from instruments (external events). SCHEDULE and PAUSE coordinate the program with time events. Program subroutines can be scheduled to execute at a specified time of day, regardless of what is happening in the main program. SCHMITT coordinates the program with external events on the clock module Schmitt triggers. For example, signals from an instrument can control what part of the program executes next and can determine what control signals are sent to other instruments.

These process control routines are very powerful because they allow the program to perform several logically distinct operations apparently simultaneously. For example, a program could perform computations on data already received and interrupt those computations only when an instrument signals that it requires attention from the program.

Transferring ASCII Characters

The ASCII character transfer routines control communication with instruments which transmit characters using the standard protocol for serial transfer (bit serial transfer). Most terminals (including the MINC terminal) use this protocol. All MINC systems include the serial transfer units needed for this kind of transfer. (The IEEE bus routines also transfer ASCII characters, the difference being the protocol used in the transfer, which is known as a bit parallel, byte serial transfer protocol. See Book 5 for description of the IEEE bus routines.)

Input The CIN routine (*C*haracter *IN*put) collects character strings from instruments transmitting serial ASCII characters.

Output The COUT routine (*C*haracter *OUT*put) sends character strings to instruments accepting serial ASCII characters.

Maintenance Tools

Conversions For reasons of speed and efficiency, several of the input routines collect data in nonstandard formats. Other routines convert the data to standard numeric form so that the program can perform computations using the data. (See "Format Conversion," page 25.)

MAKE_TIME converts elapsed-time measurements. MAKE_BCD converts standard numeric values to BCD format values and MAKE_NUMBER converts BCD format values to standard numeric values.

Manipulations Several MINC routines require information in special formats. Other MINC routines provide the formatting capabilities.

SCAN_BIT, SET_BIT, and TEST_BIT provide capabilities for inspecting input data (particularly from digital units) in detail and for creating required output control values (for analog or digital output).

LOCATING INFORMATION ABOUT LAB MODULES

Two books in the MINC set explain how to use the lab modules. Book 6 describes how to write programs to control communications with external instruments. Book 7 describes how to connect external instruments to the devices installed in the chassis. These two books are closely related. You can start with either, depending on your preference (programs first, or instruments first). Throughout, each book provides guidance to relevant sections in the other.

Book 6 In addition to describing how to use the lab module routines, Book 6 summarizes which lab modules are required by each routine. The configuration summary appears at the beginning of each routine description as a set of function diagrams. Of course the function diagrams lack the detail necessary to serve as instructions for connections. However the configuration description does serve two important purposes:

1. If you prefer to begin by planning a program, the configuration diagram serves as a reference guide to Book 7 by showing you exactly which modules are necessary. You can then study only the relevant sections of Book 7 to make proper connections.

2. When you are using a program, the summary serves as a convenient way of verifying that no necessary modules are missing from the chassis.

Book 7 In addition to describing how to connect instruments to the lab modules, Book 7 summarizes the routines that can control each module. This serves two important purposes:

1. When you are making connections, Book 7 refers you to the correct routine so that you can use that routine to verify that the connections are correct.

2. If you prefer to begin by planning the instrumentation, the summaries in Book 7 serve as references to Book 6. You can

then study only the relevant routines in Book 6 after the connections are complete.

LAB MODULE PROGRAMMING

Control Programs

Programs that control data transfer are called *control programs*. Control programs differ from programs that perform only computations. The essential characteristic of a control program is that it coordinates its activities with activities outside the computer system. That is, time, and the time intervals between events, are fundamental program parameters.

A control program can send signals to start or to stop data transfer, can start data transfer on receiving signals from instruments, or can examine inputs from instruments and modify the conditions for other instruments based on the results. In short, when you want to use MINC to control any external process, the program you write is a control program.

The tools MINC provides for writing control programs are the lab module routines and the IEEE bus control routines. The lab module routines supervise communications between the laboratory modules and the instruments connected to them. The IEEE bus routines supervise communications between IEEE standard instruments and MINC via the IEEE bus (see Book 5).

Planning a Control Program

Designing an experiment and writing a control program have the same prerequisite: planning. Planning the control program is an integral part of the whole measurement plan. In fact, the control program itself is an important part of the instrumentation and must be planned, tested, and calibrated in the same way as the rest of the instrumentation.

The most difficult step for novices in designing a control program is translating the measurement, or instrumentation, plan into a control program plan. You must identify and characterize all sources of *input* to the program and all destinations for *output* required by the program.

Three distinct sources provide three fundamental classes of input. First, input can come from people, who provide the identifiers or parameters unique to a particular run of the program or experiment. Second, input can come from stored files which can contain general parameters for an experiment and various sequences of codes or stimuli required by the experiment. Third, input can come from instruments connected to the system, which provide signals that can be interpreted by the program either as data or as controlling signals.

The set of possible output destinations is the same: people, stored files, and instruments. You can send output to people in the form of messages, requests for input, or summary displays on the screen. You can store output on files containing records of experiment conditions, raw data, or processed data. You can send output to instruments in the form of control signals.

To formulate a control program plan, you have to consider the measurement plan as a sequence of input and output events. By identifying and diagramming all of the relevant inputs, outputs, and their interactions, you construct a program plan.

The computational requirements of the program (which we have not mentioned yet) are completely defined by the input and output relationships. For example, if one of the output requirements is to store the means and standard deviations for the input data, you know that the program must calculate these statistics at some point after the input is complete. When you have completed this planning stage, study the relevant example in Chapter 3 and the appropriate reference sections on routines in order to translate the plan into a program.

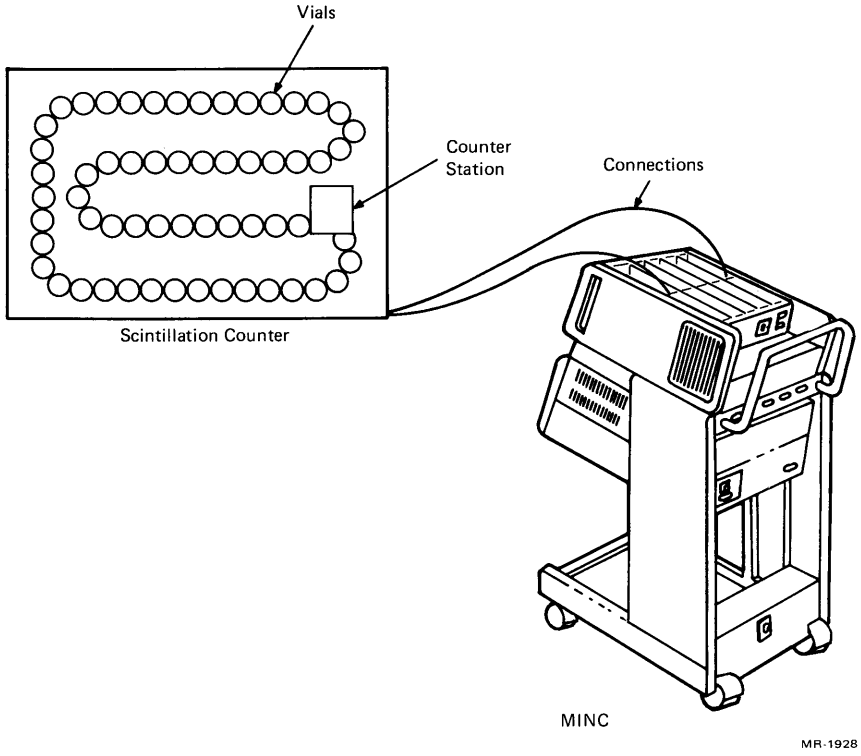


Figure 1. Diagram of Counter, Vials, and MINC.

Concrete Example Suppose the measurement plan requires reading the values from a scintillation counter. The counter must be connected properly to the MINC system. Someone must load the counter with full scintillation vials, turn on the counter, start the control program, and enter parameters (for example, the number of vials).

One useful way to conceptualize the series of events involved is to draw a "time line" for the process. One possible representation of the process described above appears in Figure 2.

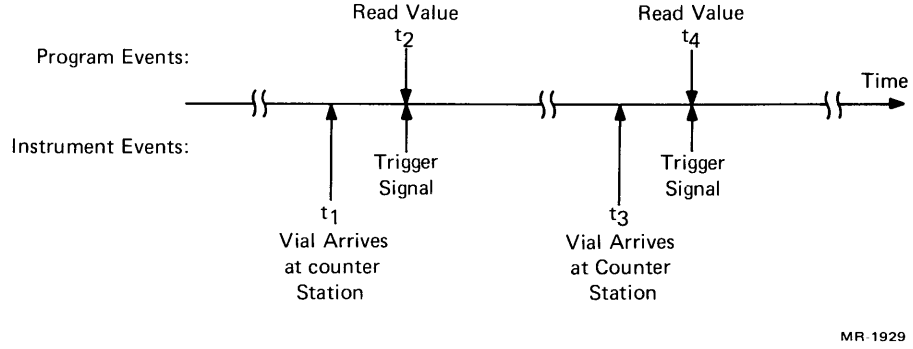


Figure 2. Time Line for Measurement Process.

In this case, the time relationships are simple, and the time line is simple. However, even a simple time line demonstrates some of the control decisions you must make. For example, how long is the interval between t_1 and t_3? If the time interval is known, then the control program can expect to find a valid reading after that time interval has elapsed. However, if the time interval is variable, then some other source, perhaps a signal from the counter itself, must notify the program that a valid value is available. What controls the interval between t_2 and t_3? In some cases, the counter itself automatically advances to the next vial.

Decide how to connect the instrument (in this case, the scintillation counter and the ready signal) to the lab modules. Use both the module descriptions in Book 7 and the manufacturer's literature for the instrument to help you make decisions about connections.

The series of events identified so far constitute the innermost loop of the program:

1. Wait for a "ready" signal from the counter.
2. When it occurs, get the counter value.

3. Return to step 1.

In isolation, these events handle only the measurement process. More decisions are necessary to describe the complete input and output process. For example, is the total number of readings known in advance? If not, what are the criteria for starting and ending data collection? While it is waiting for the next signal, the program could perform calculations on preceding values to decide whether or not termination criteria have been met. Are the instruments designed so that they can be turned off by program signals, or should the program notify humans that data collection is finished?

The measurement problem described here is relatively straightforward. However, the same principles of planning apply to the most complex control situations.

Program Structure

Notice that none of the decisions made in the example were programming decisions—they are measurement process decisions. When you have made these decisions, you can begin writing the program itself. Control programs contain three logical sections: a beginning (preparation or prologue), a middle (the actual measurement or control process), and an end (shutdown or epilogue). The middle part performs the measurement loop, the actual "work" of the program. The beginning and end parts provide context, record-keeping, and termination procedures.

Beginning The beginning of a well-designed control program determines the context in which the measurement takes place, and sets up initial conditions. The beginning section can be very simple or complex, depending on your requirements. For example, the program might request identifying information about the experiment.

```
100 PRINT 'Enter your name please'
110 LINPUT N$
120 PRINT 'Enter identifier for this experiment'
130 LINPUT E$
140 PRINT 'Enter file name for the data'
150 LINPUT F$
160 OPEN F$ FOR OUTPUT AS FILE #1
170 PRINT #1,CLK$,DAT$,E$
180 PRINT #1,N$
190 PRINT 'Enter maximum number of trials'
200 INPUT N
.
.
.
```

This program fragment prompts the person running the program to enter some essential information about the session. This is good programming practice for several reasons. It allows the program to be reused for many sessions, without the danger of losing data. The person running the program (who might not have written it) does not have to remember what information to provide, or in what order to enter it.

Middle The middle of a control program contains the statements for the measurement loop(s), for output transfers, and for any computations required during the process. The actual design of the loops depends on your configuration and application, so it is impossible to state detailed guidelines.

Some general guidelines are possible:

1. The time relationships between events are the most important and difficult aspects of the process to control. Verify that the program in fact meets the timing requirements of the process. Programs containing the wrong timing relationships often execute successfully, providing results misleadingly similar to the correct results.

2. The endpoints of the measurement process are more difficult to control than the middle. Pay close attention to the mechanism for starting measurements or transfers (and the time relationships involved) and to the mechanism for determining the end of the process and terminating it in an orderly way.

3. Unlike computational programs, the control program statements themselves convey little information about what they control or how they do it. Document the time and event relationships carefully and completely, both in program REM statements and in written records (like the time line sketches and instrumentation records in Book 7).

4. The lab module routines report on those erroneous conditions they can detect (see "Error Detection," page 51). However, the routines can detect only well-defined, general error conditions. You must be alert to the specialized error conditions that could occur in your application and monitor the process and results closely throughout to avoid invalid program runs.

End The end of a control program contains the procedures for terminating the session. The program end must halt any data

transfer still in progress, save any required workspace arrays, close open file channels, and compute and display any summary data. For example, the end section of a program might contain statements like these:

```
.
.
.
890 CLOSE #1
900 PRINT N1;' trials completed.'
910 PRINT 'Session ';E$;' completed at ';CLK$;' on ';DAT$
920 PRINT ' - - - - Be sure to reset the equipment - - - - '
930 END
```

Testing

When a control program has been written, it must be tested thoroughly. The most rigorous way to test a program is to simulate the experiment, that is, to run several complete test sessions, using either computed input or known input signals (for example, waveforms from a function generator). Some instruments supply calibration standards you could use in program testing.

By running a complete session, you test several aspects of the program. First, you learn whether the values accumulated during the session stay within expected limits (for example, do sums become too large for integer variables?). You also test error-detection sections of the program by deliberately providing erroneous conditions. Most important, you test the end section of the program. Does the program respond correctly to termination criteria? Does the program collect exactly the number of data points you intended (or one too many or one too few)? Does it save the data correctly? Do the test data correspond properly to the known input?

Check your data analysis procedure to ensure that it works correctly with the test data. Don't believe the numbers just because they were calculated by a computer system. If possible, run your analysis program on a specially constructed test file of numbers for which the correct answers are known. Thorough testing can only increase your confidence, both in your own programs and in MINC itself.

CHAPTER 2
MINC CONVENTIONS AND DEFINITIONS

SYNTAX CONVENTIONS

The reference descriptions in Part 2 of Books 4, 5, and 6 all share a common structural framework. Each reference starts at the top of a new page, headed by the name and title of the routine.

The name of each routine provides information about its function. For example, "AIN_HIST" indicates that the routine deals with analog input (AIN) histograms (HIST). Some general name conventions follow:

1. *A* at the beginning of a name refers to *A*nalog.
2. *C* at the beginning of a name refers to *C*haracter.
3. *D* at the beginning of a name refers to *D*igital.
4. In combination with the above letters, *IN* refers to *IN*put transfer and *OUT* refers to *OUT*put transfers.
5. The suffix *HIST* names routines that generate histogram arrays.
6. The suffix *TIME* names routines that measure elapsed time.

Many routine names contain the underscore character for ease of reading, understanding, debugging, and program maintenance. If MINC rejects a program statement, check to be sure that the routine name contains the underscore character, not a hyphen.

The routine title summarizes the routine's function. For example, the title for the AIN routine is "collect analog data."

The reference for each routine contains the following major sections.

Operation	Short description of the capability of the routine.
Configuration	The lab modules required for the routine and any optional modules. (Not applicable in Books 4 and 5.)
Statement form	The complete syntactic form of the statement for using the routine.
Argument table	Summary table of the argument descriptions, valid data types and values, and default values.
Example/Result	Description of several sample statements that would be valid in the proper program context.
Argument descriptions	Detailed description of each argument for the routine.
Related routines	List of the other lab module routines having similar or complementary function.
Restrictions	Collection of interactions, restrictions, esoteric details, and hints for using the routine.
Errors	List of error conditions, messages, and corrective actions for the routine.
Examples	A reference to the location in this book of a short, complete program example demonstrating use of the routine in a real problem solution.

Each reference uses this framework. If any section is irrelevant to the routine, the book says so explicitly. The following paragraphs discuss both the contents of, and the conventions applying to, each section.

Operation

The operation section is a concise statement of the capability of the routine, which outlines the routine in the simplest possible way. It always refers to the sections in Part 1 where relevant concepts are defined and explained.

Configuration

The configuration section summarizes the lab modules necessary to use the routine. The summary consists of function diagrams of the modules, in the same left-to-right order as they must be positioned in the chassis. Diagrams for required modules appear in black; diagrams of optional modules appear in blue. You must install the required modules to use the routine. Install whatever optional modules you need for the capabilities you want.

For example, the configuration for the AIN_SUM routine follows:

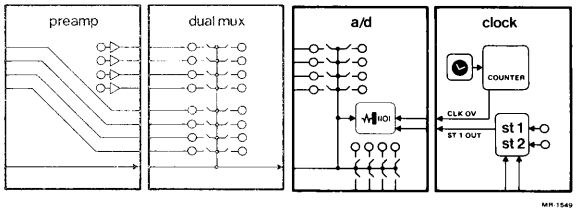


Figure 3. Function Diagram for AIN_SUM.

This means that the A/D converter and Clock 0 (in black) are both required, and the A/D converter must be to the left of the clock. The preamp and dual mux (in blue) are both optional, depending on whether the program requires the amplification and multichannel capabilities they provide. If the program requires them, they must appear in the order shown, to the left of the A/D converter.

Statement Form

The statement form shows the valid statement syntax for the routine. The form consists of the routine name and meaningful descriptive names for the routine arguments. For example, here is the statement form for AIN.

AIN(mode,data-name,data-length,trigger,A/D-channel,no.-of-channels)

required elements The required parts of the form are printed in black and must always appear as shown in the form. The routine name is always required and the parentheses are nearly always required. For some routines, some of the commas are also required.

Thus, in the above example, AIN, the parentheses, and the first comma are all required; data-name is the only required argument. This means that the following short statement is valid.

Many other statements are valid, depending on what the routine is supposed to do. For example, the following statements are all valid.

```
AIN('DISPLAY',R(),100,,3)
AIN('ST2,RANDOM',V(),50,,C%(),4)
AIN(,T,,,3)
```

optional elements All of the arguments printed in blue are optional. That is, if you omit one of these arguments from the statement, the routine assumes a predefined value called the *default value*. The default values and conditions are summarized in the argument tables and are discussed in detail in the argument descriptions.

Argument lists for the MINC routines have been designed so that use of all possible default values provides the most basic function of the routine. For example, the following statement specifies the most basic analog sampling function of immediately reading a single value into variable V from analog channel 0.

```
AIN(,V)
```

This statement is completely equivalent to the following one in which the values for the arguments are specified explicitly instead of by default. (The mode argument does not have an explicit default value, so no mode value appears in the statement.)

```
AIN(,V,1,0,0,1)
```

More complex statements (like those shown earlier) provide more complex capabilities. In addition, the argument lists have been designed so that required arguments appear early in the list and optional arguments appear later.

commas The commas in the argument list serve as argument separators when the arguments are specified and as "placeholders" when the arguments have been omitted. Placeholder commas are necessary only to indicate how many arguments have been omitted to the left of an explicit argument. They are never necessary to the right of an argument when that argument is the last one specified. For example, the following simple AIN statement specifies analog channel 3 in the fifth argument.

```
AIN(,V,,,3)
```

The commas are placeholders for the mode, data-length, and rate arguments, which assume their default values. Notice that no comma is required after 3. That is, the final comma in the following statement is not invalid, but it is also not necessary. (If there were extra final commas, the program would notify you of the error.)

```
AIN(,V,,,3,)
```

It is important to realize that the following statements are *not* equivalent. (In fact, the first one is not even valid.)

```
AIN(,V,,,,3)
AIN(,V,,,3)
AIN(,V,,3)
AIN(,V,3)
```

From this you can see the importance of placeholder commas. In debugging these statements, one of the first steps is to check the commas carefully.

descriptors In the statement form, and in the argument table and descriptions, the arguments are represented by meaningful descriptors. For example, in the AIN statement form, "A/D-channel" is a descriptor for the argument in which you specify the analog channels to sample from. Obviously, a valid program statement looks very different from the statement form, because in the program statement you supply valid variable names or constants in place of the descriptors. (See also the next paragraph on argument tables and the section on "Argument Conventions," page 21.)

Argument Table

The argument table lists the argument descriptors for the routine. The top-to-bottom order in the table is the same as the left-

Figure 4. Argument Table for AIN.

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	CONTINUOUS,DISPLAY, EXTERNAL,FAST, LINE,RANDOM,ST2	standard mode
data-name	numeric variable name or array name	-2048 to +2047; full-scale values	required argument
data-length	numeric expression	≥ 1	1
trigger	numeric expression	0; > 0 to 655.35; 1 to 65,535	0
A/D-channel	numeric expression or array	0 to 63	0
no.-of-channels	numeric expression	1 to 64 or channel array length	1

to-right order in the statement form. (The top argument in the table and the leftmost argument in the statement are both Argument 1.) For each descriptor, the table specifies the type of argument required, the valid values for the argument, and its default value if it is an optional argument. Figure 4 is an argument table for the AIN routine.

type of argument The table specifies the data type required for each argument. The type specifications unambiguously define the set of valid arguments. For example, the type "string expression" means either a string literal, a string variable (or array element), or a string operator expression. (See Book 2 for full explanations of the data type terms.) The following list summarizes all the data types in MINC BASIC.

Type	*Meaning*
Numeric literal	A number, e.g., 5, 3.141, 9%
Numeric variable	An integer or real variable, a numeric array element, e.g., A1, Q%, B(2)
Numeric expression	A numeric literal, numeric variable, or an operator expression with a numeric result, e.g., 6, B(3), A1(2)+5%.
Numeric array	Integer or real array
String literal	A sequence of characters in delimiters, e.g., 'george'
String variable	A variable whose value is a string or an element of a string array, e.g., G$, D$(4)
String expression	A string literal, string variable, or an operator expression with a string result, e.g., 'Date', M$, T$+'ING'
String array	Array containing a set of strings

The routines accept numeric values for many arguments, without requiring the values to be either integer or real. If the routine requires a whole number argument, it discards any fractional part of a real value before using it as an argument.

For some arguments, the table contains an entry like "numeric variable name." The notation "name" does not appear in the data

type table. "Name" is used to indicate that, during execution, the routine *assigns a value* to the argument. That is, in most cases, you do not supply an argument value for the routine to use; the routine assigns the argument value for your program to use.

The range of values that the argument could contain (after the routine has executed) appears in the Valid Values column of the table.

valid values The table summarizes the valid range of values for each argument. The ranges are always inclusive. The argument descriptions explain fully the meanings of the different possible values.

The valid values summary contains several kinds of information.

When the valid values have an open-ended range, the routine accepts any value in that range. For example, a numeric value greater than 0 means *any* numeric value greater than 0, up to the limit for that data type (32,767 for an integer value, and 1.7×10^{38} for a real value). The routine cannot check whether or not the value you specify is meaningful in the program context.

If valid values have a specific range, the routine always checks to make sure that the actual argument value is within that range. See also "Error Detection," page 51.

The valid values shown for the mode argument (see the AIN table, Figure 4) are a special case: the commas separating the values mean that any one, and possibly more than one, of the values is valid. Refer to the section "Operating Modes and Mode Designators" for more information (page 23).

When several values appear separated by semicolons, the argument has several distinct meanings, and each range of values applies to a separate meaning. This situation occurs for the trigger argument in the AIN table, where the valid values are as follows.

0; > 0 to 655.35; 1 to 65,535

This means that the trigger argument has three separate meanings, with a different range of values for each meaning. The meanings are selected by the values of other arguments.

1. For the first meaning (external trigger control), 0 is the only valid value.
2. For the second meaning (internal clock module control), any value greater than 0, up to 655.35 is valid.
3. For the third meaning (external or line mode), any value in the range 1 to 65,535 is valid.

Example/Result

Each reference section contains a set of example and result pairs.

example The examples are single program statements that would be valid in the proper program context. The following example is for the AIN routine.

```
AIN(,V(),50)
```

In order to run properly, this statement requires (at least) that the data input array V be described in a DIM statement with array length of at least 50 elements.

The examples vary in complexity to show the range of capability of the routine.

result For each example statement, the manual describes what the statement would do, given the proper program context. The result describes the effects of most of the arguments as concisely as possible. The result description tries to capture the spirit of the operation rather than to provide complete detail.

Argument Descriptions

The descriptions for all arguments have a common organization which includes meaning, values, default, and discussion.

A single phrase defines the meaning of the argument and in most instances is simply an expansion of the argument descriptor. This phrase is often followed by a reference to an explanatory section in Part 1. Next, the section defines the full set of valid values for the argument and summarizes the meanings of the different values. The default value for the argument follows the set of values. The paragraphs following the definitions explain complex argument meanings and relationships between arguments.

Related Routines

The related routines section lists the related lab module routines. It does not mention any other kinds of MINC routines (for example, graphics). It lists the operation of each related routine, and describes the nature of its relationship to the routine.

Restrictions

The restrictions section is a collection of usage hints, performance information, and restrictions on the applicability of the routine. It sometimes reiterates explanations of interactions between arguments.

Errors

The errors section lists all of the error conditions possible for the routine and the text of the error messages. Where necessary, it discusses how to correct errors, and how to recover from errors that stop the program. (See also "Error Detection," page 51, and the complete discussion of error messages in Book 8.)

Examples

The examples section refers to the location of the example program relevant to the routine.

ARGUMENT CONVENTIONS

Arrays and Array Elements

Throughout this manual, the term "array" refers only to arrays in the workspace. You cannot use a virtual array file as an argument for any of the lab module routines. A virtual array file *element* is a permissible argument only if the routine does not attempt to assign a value to it.

If the data type for an argument is an array, designate the array by its name, followed by empty parentheses, for example, A(). The empty parentheses show that we are talking about the array A, starting with array element 0.

Suppose the array F has 10 elements.

```
50 DIM F(9)
```

You want to specify F as the data-name argument for the AIN routine. The AIN statement would look like the following.

```
100 AIN(,F(),10)
```

In this situation, the parentheses distinguish the array F from the variable F. Notice that the following three statements are all valid and all have different meanings.

```
100 AIN(,F)
100 AIN(,F(0))
100 AIN(,F(),10)
```

The first statement reads one value into variable F, the second statement reads one value into array element F(0), and the third statement reads ten values into the array F (elements 0 through 9).

In addition, you can specify arrays which are part of a whole array. To do this, you specify an array element which is to be treated as if it were the first element in the array. For example, you can specify F(5) as an array name. The following statement reads three values into the array starting with F(5) (that is, F(5), F(6), and F(7)).

```
100 AIN(,F(5),3)
```

(In fact, this means that as array names F() and F(0) are equivalent.)

Mode Strings

Many routines contain a mode argument which requires a string value. The section on "Operating Modes and Mode Designators," (page 23) and the individual routine sections explain the meaning and use of the modes. The conventions for specifying mode strings are as follows.

1. The string can contain more than one mode designator. (Valid combinations of modes appear in the table with each routine.)

2. Mode designators must be unique; that is, a mode designator cannot appear more than once in a mode string.

3. Multiple mode designators must be separated by commas.

4. Multiple mode designators can appear in any order.

5. Either uppercase or lowercase characters are valid.

6. The routine interprets only the first character of each mode designator. In this book, designators are shown as full word mnemonics for ease of reading and understanding.

The following three examples are all equivalent.

1. MAKE_TIME('ST2,CHz',I%,R)

2. MAKE_TIME('S,C',I%,R)

3. P$ = 'ST2,CHz'
   MAKE_TIME(P$,I%,R)

It is good programming practice to use a full word form for mode designators so that you (and others) can read, understand, and modify your programs easily.

The mode designators for the lab module routines are all "positive." You cannot use a minus sign to cancel or reverse the mode designator as you can with the graphic routines (see Book 4). The minus sign does not cause an error but it has no effect.

OPERATING MODES AND MODE DESIGNATORS

For many of the lab module routines, the string argument called "mode" designates an operating mode for the routine. For example, the mode designator EXTERNAL designates *external mode*, in which timebase signals on ST1 provide a time base for a transfer.

The meaning of the absence of a mode string depends on the nature of the routine. Some routines have a *standard mode* of operating which cannot be specified by any of the mode designators. Other routines have a *default mode* of operating which assumes one of the possible mode designators. (The reference section for each routine specifies how the routine operates in the absence of a mode string.)

Standard Mode

For the standard mode case, the reference section for the routine describes its standard operating mode. The mode designators specify additions or changes to the normal operating mode for the routine.

The mode designators specify modes of operation that are either expansions of standard mode or fundamentally different from standard mode. For example, AIN_HIST in standard mode generates a histogram using values collected from an analog channel. One of the optional mode designators for AIN_HIST is DISPLAY (for display mode operation) which results in standard mode operation *plus* the play of the histogram. In this case, the optional mode designator augments the function of standard mode to provide display mode.

In other cases, the optional mode designator changes standard mode rather than adding to it. For example, AIN_SUM in standard mode accumulates data under internal clock control from successive sweeps of an analog channel (or channels). The mode designator EXTERNAL changes AIN_SUM so that it operates in external mode, accumulating data under external timebase control from successive sweeps of an analog channel (or channels).

It is important to realize that mode designators like EXTERNAL are incompatible with standard mode because you cannot have both internal clock control and external timebase control at the same time. Therefore, the EXTERNAL mode designator does not simply add external timebase control to standard mode but instead fundamentally changes how AIN_SUM interacts with the outside environment.

With some routines, you can specify several optional mode designators together. For example, with AIN_SUM you could combine DISPLAY, ZERO, and EXTERNAL to add display and zeroing to external mode. However, you could not combine EXTERNAL and LINE because these modes request mutually incompatible changes to standard mode.

The reference sections contain two-way contingency tables showing the valid pairs of mode combinations. The mode designators are independent. Therefore, you can determine the validity of any multiple combination by examining the validity of its component two-way combinations.

The routines with standard mode are AIN, AIN_HIST, AIN_SUM, AOUT, CIN, COUT, DIN, DOUT, PST_HIST, SET_GAIN, and TIME_HIST.

Default Mode

In the default mode case, default mode is the mode assumed by the routine when no optional mode has been specified. That is, leaving out the mode designator is exactly the same as specifying a particular mode designator. (See the discussion of default values on page 16.)

For example, the SCHEDULE routine has designators for interval mode (INTERVAL) and absolute mode (ABSOLUTE). If you do not specify a mode designator, SCHEDULE assumes interval mode by default. That is, the following statements have the same effect.

```
SCHEDULE('INTERVAL',60,L)
SCHEDULE(,60,L)
```

These routines do not have a standard mode that is different from that obtained with the mode designators. In addition, the designators for these routines are mutually exclusive, that is, only one designator can be specified at a time.

The routines with default mode are FFT, MAKE_TIME, SCHEDULE, START_TIME, and TERMINATE.

DATA TYPES AND NUMBER SYSTEMS

Number Systems

Internally, MINC represents all information using the binary number system. In the binary number system there are two states: 0 (for *off* or *clear*) and 1 (for *on* or *set*). MINC contains a large number of elements that can assume only the values 0 and 1. These elements are called *bits* (meaning *binary digits*).

Everything in MINC—all numeric values, string values, and programs—is represented by codes consisting of bit strings of 1's and 0's. For example, the number *29* is stored as a sequence of bits (representing 0's and 1's), not as the decimal digits *2* and *9*. The command *DEL* is stored as a sequence of bits (representing 0's and 1's), not as the characters *D*, *E*, and *L*.

Integers, real numbers, and characters each have their own standard representation format. That is, the integer 3, the real number 3, and the character 3 are represented internally by different patterns of bits.

The value of an integer must lie within the range -32,768 to +32,767 (inclusive). The value of a real number must lie within the range 0.29×10^{-38} to 1.7×10^{38} (inclusive). The value of a character must be one of 128 ASCII characters (see Book 2 and Book 3). These limits are imposed by the number of bits and the coding scheme used to represent the values.

In most cases, the details of the internal representation of numbers and strings are invisible, and irrelevant, to MINC programmers.

Format Conversion

In some cases, you do need to be aware that the internal differences exist, although knowing details of the internal representation is not necessary. For reasons of speed and efficiency, several MINC routines represent values in nonstandard data formats. In such cases, the program must use a *conversion routine* to convert from the particular nonstandard format used to a standard representation format.

BCD

One common nonstandard format is Binary Coded Decimal (BCD). Many instruments transmit and receive information represented in BCD format. If a BCD instrument is connected to a digital input unit, the values read by the DIN routine are in BCD format. The program must then call the MAKE_NUMBER routine to convert the BCD format values to standard numeric format. If a BCD instrument is connected to a digital output unit, the values sent by the DOUT routine must be in BCD format. The program must first call the MAKE_BCD routine to convert standard numeric values to BCD format values.

In BCD, each decimal digit in a number is represented by four binary digits. For example, the number 29 is represented by two groups of four bits, one group representing the digit 2 and the next group representing the digit 9:

decimal digit	2	9
BCD format	0010	1001

Table 1 shows the BCD code and decimal equivalents for each of the digits.

Table 1. BCD Codes for Decimal Digits.

Decimal digit	*BCD code*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

MINC can transmit and receive BCD values in the range 0 to 9999 (inclusive).

Bits and Words

It is usually unnecessary to consider bit sequences when programming MINC. However, for digital input and digital output, it can be useful or necessary to consider the condition of a single bit.

The following conceptualization is simplified, but it is valid for the situations in which you might have to encounter bits. You cannot refer directly to any individual bit in the MINC workspace, but you can refer to it indirectly as a component of some named value. For example, you can refer to a single bit "within" variable A. We will call each named group of bits a *word*. A word is like an array of 16 bits. Any time you need to refer to a bit, you must specify both the word containing the bit, and the position of the bit within the word (much like referring to an array element).

The 16 bit positions within a word are numbered 0 to 15, from right to left.

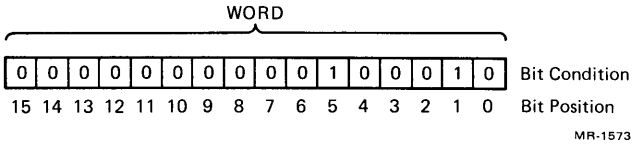


Figure 5. Defining Bits in a Word.

The routines which access bits within a word are TEST_BIT, SCAN_BIT, and SET_BIT.

Bits and Lines

You must understand the concepts of bits and words in order to understand how to use the digital input and digital output units and the routines DIN, DIN_EVENT, DIN_MASK, DOUT, DOUT_MASK, and the bit manipulation routines.

Each digital input unit or digital output unit contains 16 physical lines. You connect instruments to the lines at the connector block terminals, which are labeled D00 through D15, and which stand for lines 0 to 15. As with bits, the conditions of the lines can be clear (0), or set (1). For the correspondence between set, clear, and voltage levels, see Book 7.

For the digital transfer routines DIN and DOUT, lines within a physical unit correspond directly to the bits within a word.

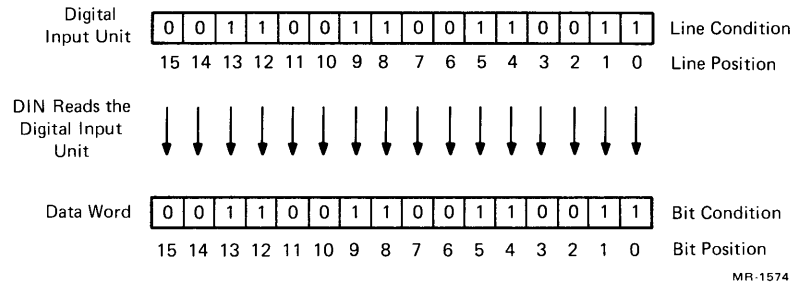


Figure 6. Correspondence of Lines and Bits.

When DIN reads a digital input unit into a data input word, the data input word contains the same value as the digital input unit. (The value of the unit changes whenever any of the signals from the external instruments changes, but the value of the word remains the same unless the program changes it.)

Masking

Masking is the name of a technique that allows you to disregard the conditions of certain lines in a digital input or output unit. Without masking, it would be possible for the program to disregard line values by clearing the bit values in the variable (using SET_BIT, for example). However, to change all the variables in

a sweep of data, this technique would become tedious and time-consuming. The masking technique produces the same effect automatically.

A mask is logically like a filter or screen, imposed between the unit and the corresponding word, which allows the conditions in certain positions to pass through, but stops the conditions in other positions.

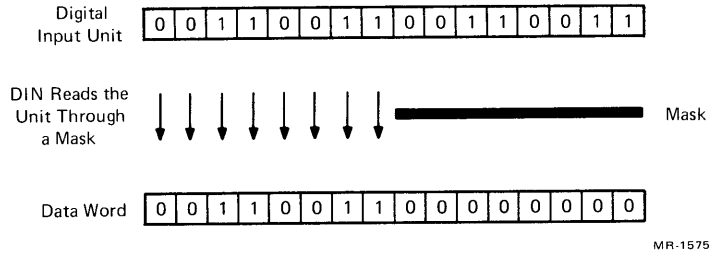


Figure 7. How a Mask Works.

Specifically, the mask itself is a word. The bit conditions of the mask word define whether or not the corresponding line conditions can pass. A set bit allows the condition to pass and a clear bit stops the condition from passing.

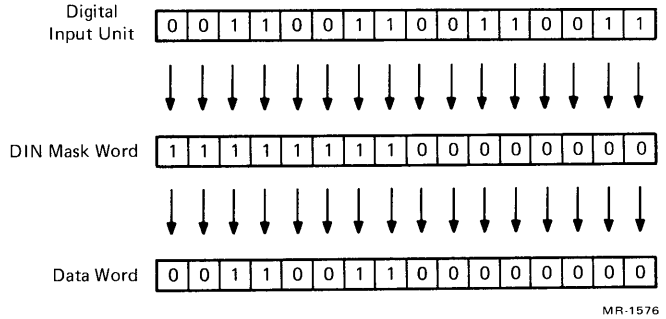


Figure 8. Reading Input Through a Mask Word.

Notice that all the bits in the data word corresponding to the masked lines (masked bit clear) in the unit are clear. The masked bits are always clear, regardless of their condition before the data transfer. Also, the bits in the data word corresponding to unmasked lines (mask bit set) have the same conditions as their equivalent lines.

The examples so far have shown masking digital input units. The concepts apply similarly to digital output units.

The DIN_MASK and DOUT_MASK routines define mask words for digital input and output.

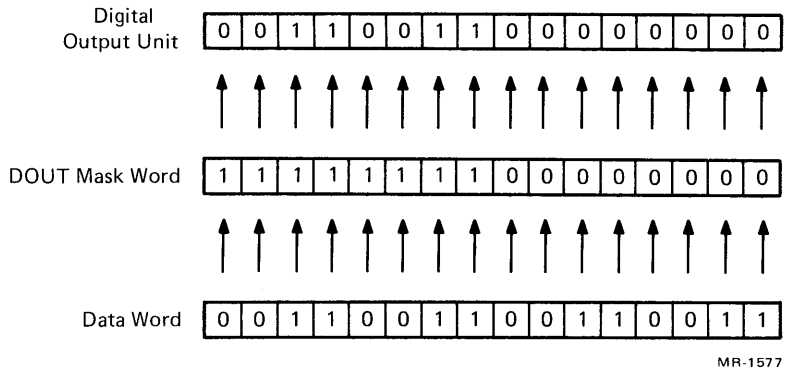


Figure 9. Sending Output Through a Mask Word.

SERVICE SUBROUTINES

A *service subroutine* is a subroutine in your program which is invoked by one of the lab module routines. With normal subroutines, a GOSUB statement transfers control to the subroutine. With service subroutines, one of the lab module routines sets up the conditions which result in transferring control to the service subroutine.

Service subroutines provide a mechanism for your program to respond to the occurrence of external events and time events. The lab module routines which invoke service subroutines are CONTINUE, SCHEDULE, and SCHMITT.

A service subroutine has the same form as a normal subroutine. The last statement executed in a service subroutine is a RETURN statement. With normal subroutines, the RETURN statement transfers control back to the statement following the GOSUB statement. With service subroutines, the RETURN statement transfers control back to the statement following the one which was executing when the external event or time event occurred.

In most programs, a service subroutine executes immediately in response to the external or time event causing it. However, service subroutine execution can be delayed if the event occurs while a long duration statement or another service subroutine is executing. See "Program Dynamics for Control Programs," below, and Table 3 (page 32) for details.

PROGRAM DYNAMICS FOR CONTROL PROGRAMS

In MINC BASIC, the current statement has control of the program for as long as it requires to execute. While it is executing, no other statements, signals, or service subroutines can get the attention of the program. When the current statement finishes, the program can check whether any requests for service have arrived during the time while the statement was executing.

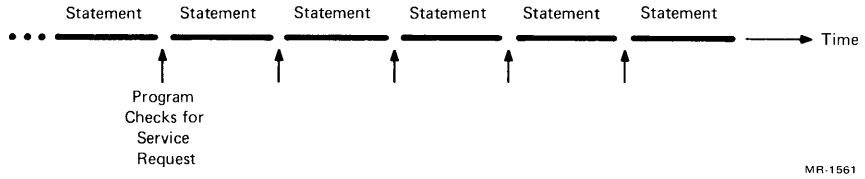


Figure 10. Checking for Service Requests During Program Execution.

If no requests have arrived, the program passes control to the next statement (physically next, or logically next).

If a service request has arrived, the program does not move on to the next statement but instead transfers control to the appropriate service subroutine or data transfer routine.

Most program statements execute very quickly, so that you aren't really aware that external signals can get the program's attention only between statements. For example, consider the sequence of execution for statements in the following program. This program schedules a service request for one second later and then begins executing a FOR/NEXT loop that requires longer than one second to finish.

```
10 S = 0 \ J = -1
20 SCHEDULE('Interval',1,100)
30 FOR I = 0 TO 499                    100 J = I
40 S = S + I                           110 RETURN
50 NEXT I                              120 END
60 PRINT 'Interrupted at index';J
70 STOP
```

The program can respond almost immediately to the SCHEDULE service request that occurs while one of the statements in the FOR/NEXT loop is executing. Suppose the current statement is statement 40 and the end of the scheduled interval occurs. As soon as statement 40 finishes (which is almost immediately), the program transfers control to the service subroutine for SCHEDULE.

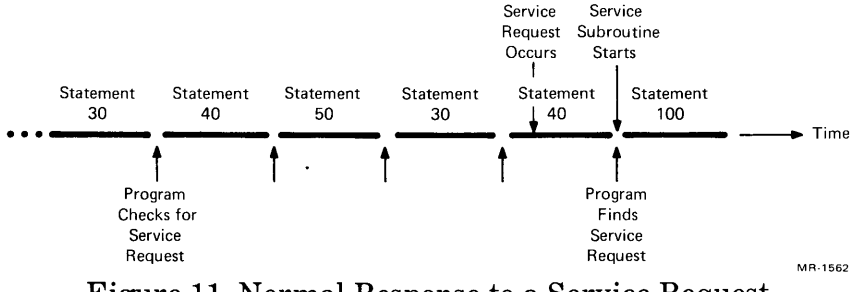


Figure 11. Normal Response to a Service Request.

Some statements for the lab module routines can require a long time to finish executing. For example, if you collect a sweep of analog data using AIN, the AIN statement remains the current statement until all of the points in the sweep have been collected. The program cannot honor any service requests until the AIN statement finishes. The time relationship could look something like the following.

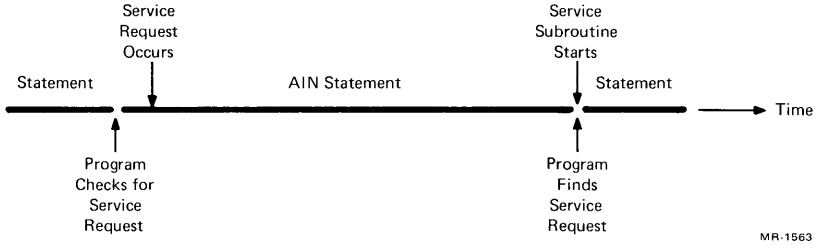


Figure 12. Delayed Response to a Service Request.

The following example program contains time relationships like those shown in Figure 12.

```
10 PRINT 'Time: ';CLK$
20 DIM A(9)
30 SCHEDULE('Interval',1,100)
40 AIN(,A(),10,1)
50 FOR I =  0 TO 9 \ PRINT A(I) \ NEXT I
60 STOP
100 PRINT 'Time: ';CLK$
110 RETURN
120 END
```

In statement 30, the program schedules a service subroutine for one second later and then, in statement 40, starts an analog input sweep of 10 data points, one point per second. The time event requested by SCHEDULE occurs during execution of the AIN statement, but the service subroutine (at statement 100) cannot begin executing until statement 40 finishes. When you run the program, you see two clock values, followed by the converted analog values. This indicates that the service subroutine (which prints the second clock value) has executed between statements 40 and 50.

Most program statements involving terminal input and output or data transfer require a significant amount of time to execute. It is important for you to be aware of these time constraints when you are writing programs to sample at high rates, or programs with very precise timing requirements. Try to avoid slow statements (like file channel transfers) during portions of the program where the time relationships are critical.

Tables 2 and 3 list program operations and routines that can require significant amounts of time. The tables do not indicate how much time these require because the operating speed varies, depending on the nature of the operation and on a complex interaction of factors.

Table 2. Program Operations Requiring Significant Execution Time.

Statement Verb
PRINT
PRINT USING
INPUT
INPUT #
INPUT # with double buffering
Virtual array file operations

Table 3. Routines Requiring Significant Execution Time.

Routine Name	*Limiting Factor*
AIN	Applies to point and sweep transfers. The time relationships differ for continuous mode; see page 33.
AIN_HIST	Finishes when all required points have been collected.
AIN_SUM	Finishes when all sweeps are complete.
AOUT	Applies to point and sweep transfers. The time relationships differ for continuous mode; see page 33.
CIN	Finishes when all required characters have been received or when termination character arrives.
COUT	Finishes when the last character has been sent (in wait mode).
DIN	Applies to point and sweep transfers. The time relationships differ for continuous mode; see page 33.
DOUT	Applies to point and sweep transfers. The time relationships differ for continuous mode; see page 33.
FFT	Computation time depends on length of arrays.
PAUSE	Waits until a time interval is over.

POWER	Computation time depends on length of arrays.
PST_HIST	Finishes when all sweeps are complete.
TIME_HIST	Finishes when all required intervals are complete.
WAIT_FOR_DATA	Finishes when current array partition is ready for processing.

This list is not exhaustive. The same concept applies to routines in Books 4 and 5 as well.

In control programs, you should remain aware of the dynamics of program execution so that you don't design programs with impossible or impractical time relationships (like the example in Figure 12). It is equally important to test the programs carefully so that you can be confident of collecting valid data.

Immediate Mode

Like the other MINC statements and routines, many of the lab module routines work in immediate mode. In general, the only routines that do not work in immediate mode are those which involve service requests, elapsed-time measurement, and continuous mode transfers. (All standard mode transfers do operate in immediate mode.) The Restrictions sections note those routines which do not operate in immediate mode.

CONTINUOUS DATA TRANSFER

Analog and digital data transfers can operate in a mode called *continuous mode* where they can transfer an unspecified number of values. In continuous mode, data transfers begin and continue until the program stops the transfer by executing a TERMINATE statement. The following discussion applies to the routines AIN, AOUT, DIN, and DOUT with the CONTINUOUS mode designator.

Continuous data transfers are more complex than point or sweep transfers. The major differences between a sweep transfer and a continuous transfer involve program dynamics and the workspace.

Transfer Dynamics

In standard mode, a data transfer statement (for example, AOUT) remains in control until all of the data values specified have been transferred. No other statement can execute until the data transfer statement finishes. However, in continuous mode, the data transfer statement passes control to other statements which execute while the transfer is in progress.

Unlike the standard mode case, AIN, AOUT, DIN, DOUT in continuous mode do not actually perform the transfers. The continuous transfer statement prepares the program and the workspace for a continuous transfer, and then finishes, passing control to the next program statement. The data transfer process actually begins when one of two transfer management routines (CONTINUE or WAIT_FOR_DATA) executes. The management routine monitors the transfer and provides the mechanism for stopping the transfer. The process is explained in more detail in the following pages.

Array Partitions

Continuous transfers give the program the capability to start a data transfer without specifying in advance how many values to transfer. However, as you know, the amount of workspace available for data arrays is limited. What if the experiment requires that you collect more data points than the workspace can hold?

The continuous transfer process treats its data array as a temporary array containing two array partitions, as shown in the following figure.

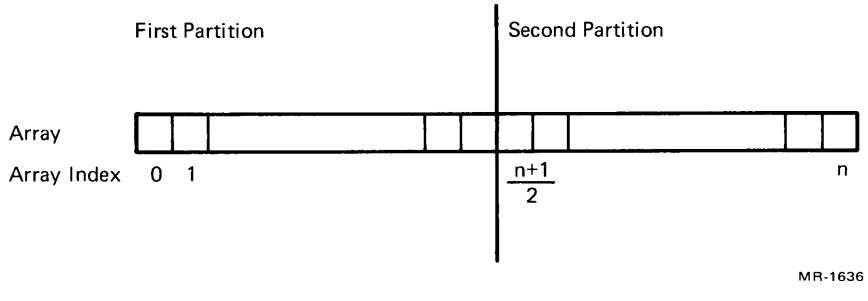


Figure 13. Array Partitions and Array Indexes.

The transfer process continuously reads values into the array (with AIN or DIN) or sends values from the array (with AOUT or DOUT). The transfer management routine transfers control to your program whenever a transfer is complete for either partition. Then your program processes that just-completed partition while the data transfer continues, using the other partition.

For input transfers, your program can either store in a file or process the data in one partition *while the routines continue to fill the other partition with data.* For output transfers, your program fills the partition with the values to be output, *while the routines continue to send the values from the other partition.* That is, the partition being transferred and the partition being processed alternate, as if the data array were circular.

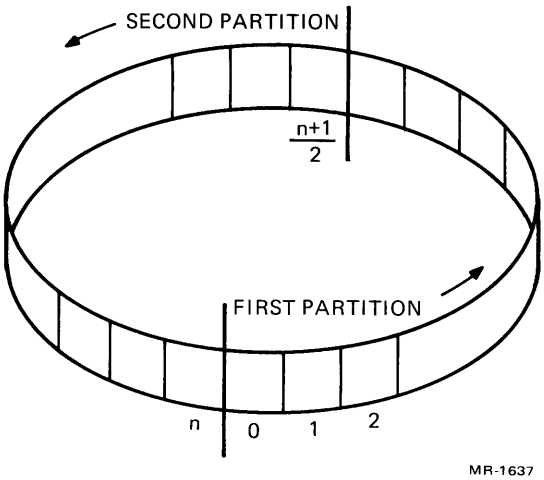


Figure 14. Circular Array Structure.

Array element 0 follows array element n. Therefore, if an output transfer routine has just sent the value from array element n, it sends the next value from array element 0.

Your program treats the data array as if it were two independent arrays of equal length, as shown in Figure 15. (If the array contains an odd number of elements, the transfer routines do not use the final element.)

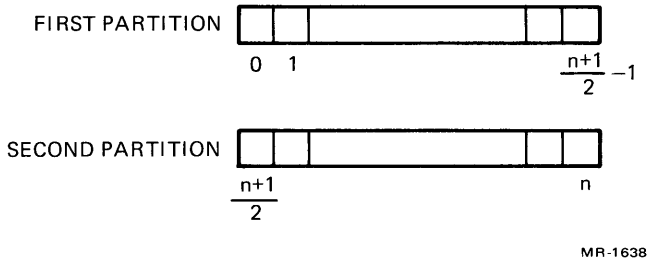


Figure 15. Array Partitioned into Two Linear Arrays.

During the transfer, the management routine passes control to your program at regular intervals while the transfer continues. Your program processes one partition and passes control back to the management routine.

Transfer Management Methods

The transfer management routines (CONTINUE and WAIT_FOR_DATA) provide two conceptually distinct program design options. Basically, with WAIT_FOR_DATA, the program simply waits until an array partition is ready for processing. With CONTINUE, the program continues executing statements unrelated to the data transfer until an array partition is ready for processing.

With either method, the program processes an array partition as soon as it is ready for processing. With either method, data transfer continues regardless of what the program is doing. The important conceptual difference lies in whether the program waits or continues executing in the interval between processing array partitions.

The technical difference between the two methods lies in the location of the statements which process the array partitions. With WAIT_FOR_DATA, the statements processing the partition immediately follow the WAIT_FOR_DATA statement. With CONTINUE, the statements processing the partition are in a service subroutine. (See "Service Subroutines," page 29.)

WAIT_FOR_DATA

With WAIT_FOR_DATA, the program dynamics are very similar to normal program dynamics. (See "Program Dynamics for Control Programs," page 29.) Figure 16 diagrams how continuous input should be managed using WAIT_FOR_DATA. Figure 17 diagrams how continuous output should be managed using WAIT_FOR_DATA.

The following short program shows the statement sequence.

The WAIT_FOR_DATA statement (statement 50) finishes executing and passes control to the statement following it when the current array partition is ready to be processed by your program. The statements following WAIT_FOR_DATA process the current partition (statement 60 through 100). The program then must execute a WAIT_FOR_DATA statement again to wait until the next array partition is ready (statement 110).

```
10 REM Collect data from A/D channel 5 and
20 REM display the average of every 50 points
30 DIM A(99)
40 AIN ('CONTINUOUS',A(),100,1/10,5)
50 WAIT_FOR_DATA (A(),I)
60 S=0
70 FOR J=I TO I+49
80 S=A(J)+S
90 NEXT J
100 PRINT 'AVERAGE=';S/50
110 GO TO 50
```

The program example shown above is designed to keep running until someone enters CTRL/C on the keyboard. For many applications, it is preferable to have the program itself stop the transfer. In that case, the statements processing the partition

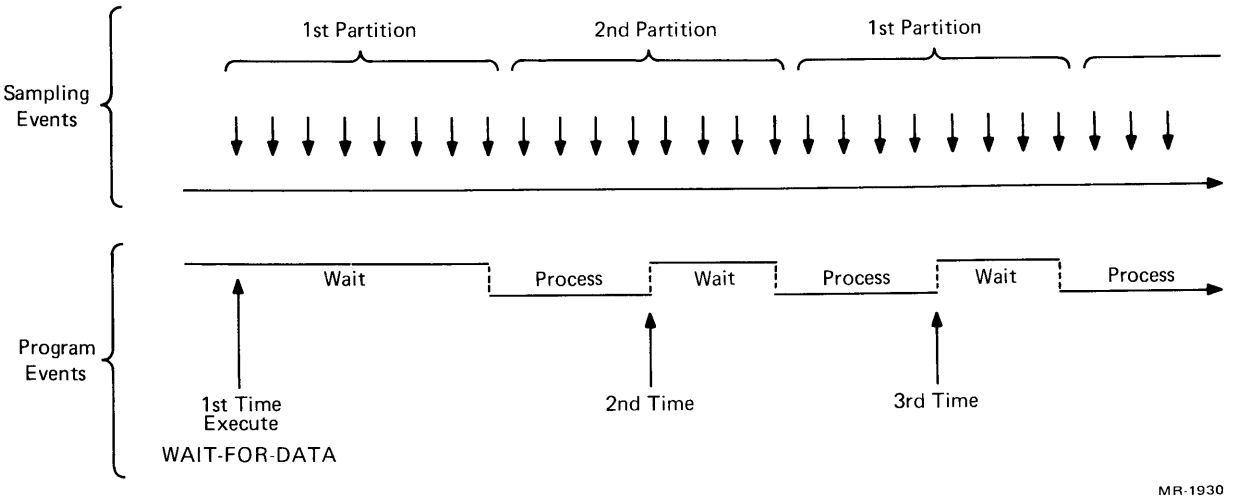


Figure 16. Using WAIT_FOR_DATA to Manage Input Transfers.

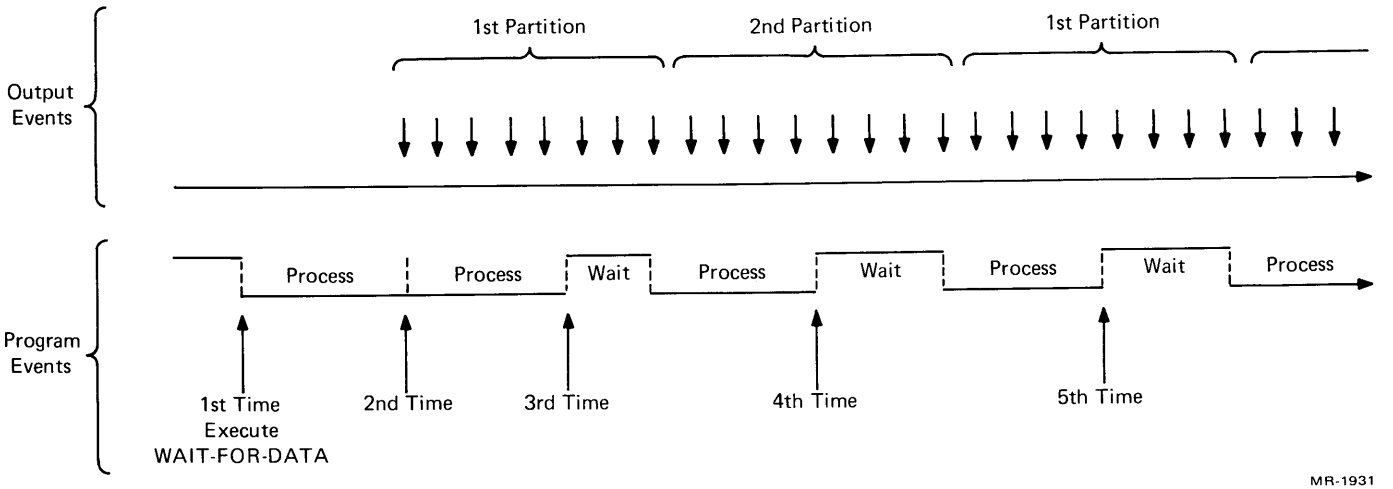


Figure 17. Using WAIT_FOR_DATA to Manage Output Transfers.

would include statements which determine when and how to terminate the transfer.

CONTINUE

With CONTINUE, the program flow is more complex, but again the program dynamics are very similar to normal program dynamics. Figure 18 diagrams how continuous input should be managed using CONTINUE. Figure 19 diagrams how continuous output should be managed using CONTINUE.

The following short program shows statement sequence with CONTINUE.

The CONTINUE statement (statement 70) prepares for the transfer to the service subroutine and then passes control to the program statements which follow (statement 80). The example has a single dummy statement here. Normally, you would do something here, like starting another transfer with CONTINUE and a different device.

The program executes normally until an array partition is ready. It then transfers control to the service subroutine (statements 100 through 200) as soon as the current statement finishes executing.

The service subroutine processes the current partition, and must, in addition, execute another CONTINUE statement (statement 190) to prepare for the response to the next array partition. In returning from the service subroutine, the program transfers control to the program statement following the one that was executing when the array partition became ready (statement 80).

```
10 REM Generate an output function F=(T*T) mod 1024
20 REM on D/A channel 0, where T increases
30 REM by 1 every 0.1 seconds
40 DIM A(100)
50 AOUT ('CONTINUOUS',A(),100,1/10)
60 T=0
70 CONTINUE(A(),I,100)
80 GO TO 80 \ REM other program statements go here
100 REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
110 REM Service subroutine for continuous mode AOUT
120 REM
130 FOR J=I TO I + 49
140 F=T*T
150 T=T+1 \ IF F < 1024 THEN 170
160 F=0\ T=0
170 A(J) =  F
```

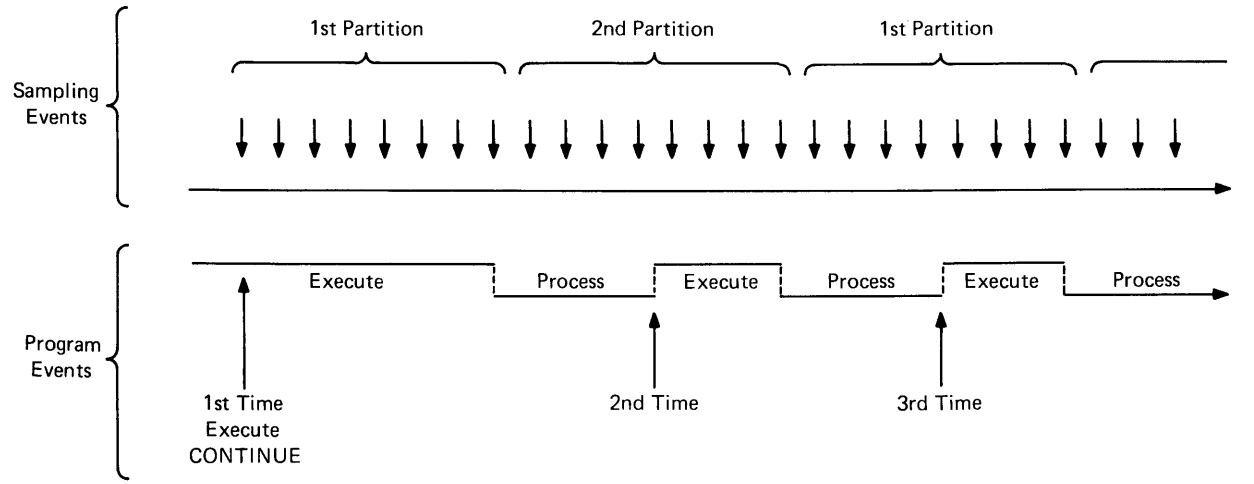


Figure 18. Using CONTINUE to Manage Input Transfers.

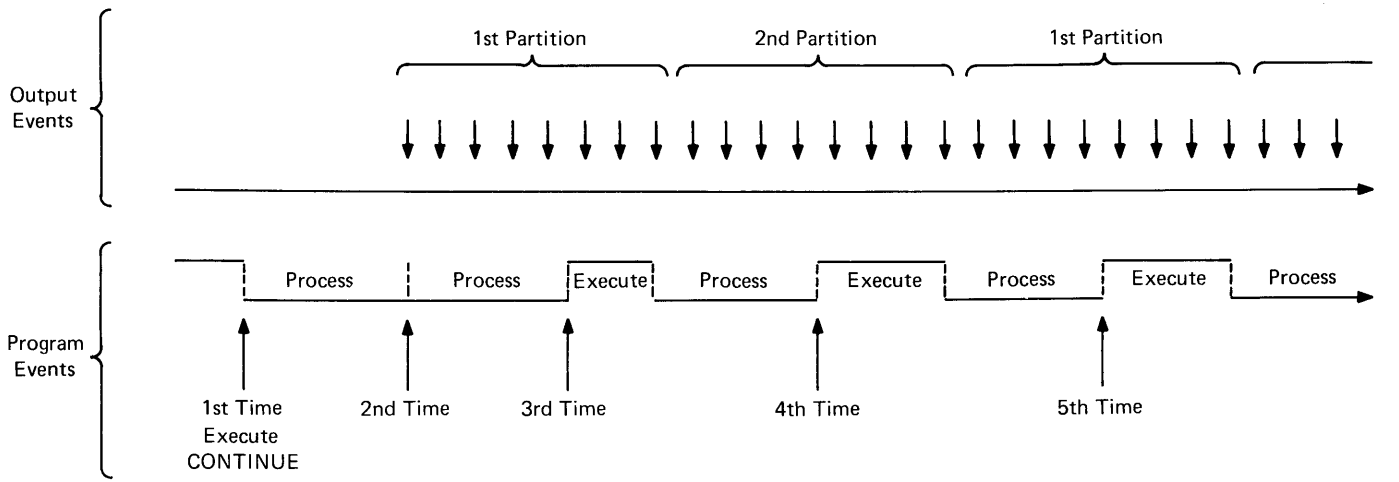


Figure 19. Using CONTINUE to Manage Output Transfers.

```
180 NEXT J
190 CONTINUE (A(),I,100)
200 RETURN
210 END
```

The first time the CONTINUE routine executes for an output transfer, it passes control to the service subroutine. The service subroutine then prepares the first partition for transfer and executes CONTINUE again. The output transfer starts at this point (see Figure 19).

The program example shown above is designed to keep running until someone presses CTRL/C. For many applications, it is preferable to have the program stop the transfer. In that case, the service subroutine would include statements which specify when and how to terminate the transfer. (Example C shows an input transfer using CONTINUE.)

ANALOG CHANNEL SPECIFICATION

The analog processing routines require you to specify the channels for analog transfer.

The AIN_HIST routine samples from one channel only. You specify the channel number for the channel carrying the input signal.

The other analog transfer routines (AIN, AIN_SUM, AND AOUT) permit multichannel analog transfers. If a transfer requires more than one channel, you specify a *conversion sequence* of channels. Whenever the transfer routine detects a trigger event and starts a conversion, it performs the transfer for the full conversion sequence as quickly as possible. Thus, with multichannel transfers, the transfers occur in groups, each group consisting of one conversion sequence, and each conversion sequence initiated by a trigger event (specified in the transfer routine arguments).

Two arguments in the transfer routines define the conversion sequence, the channel argument and the no.-of-channels argument. In standard mode, the conversion sequence consists of sequential channels. In random mode (using the RANDOM mode designator), the conversion sequence consists of nonsequential channels.

Sequential Channels

For sequential conversion sequences, you specify the number of the first channel, and the number of channels to use. For example, suppose you are collecting analog input on a system with 16

channels (numbered 0 to 15) and need to sample channels 4, 5, and 6. Define the starting channel number (4) and the number of channels (3) as shown in the following program fragment.

```
130 DIM A(11)
140 C = 4
150 N1 = 3
160 AIN(,A(),12,,C,N1)
```

This program fragment defines a conversion sequence which starts with channel 4 and continues with the next two channels in the sequence, channels 5 and 6. When statement 160 executes, AIN collects one conversion sequence on each trigger event. Figure 20 shows the sequential sampling order.

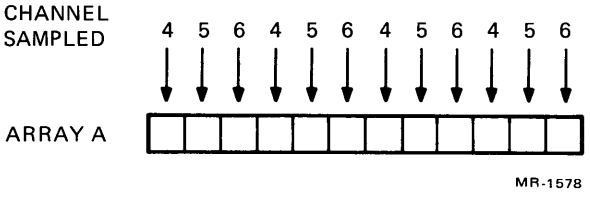


Figure 20. Sampling Sequential Channels.

The starting channel argument can be an array element instead of a simple variable. Suppose there is an array of channel numbers, C%, filled with values 0, 2, 4, through 14.

```
130 DIM C%(7),A(11)
140 FOR I = 0 TO 7 \ C%(I)=I*2 \ NEXT I
150 N1 = 3
160 AIN(,A(),12,,C%(2),N1)
```

This program fragment defines a conversion sequence that starts with the channel specified in array element C%(2) (that is, channel 4) and continues with the next two channels in sequence, channels 5 and 6. When statement 160 executes, AIN collects one conversion sequence on each trigger event. Figure 21 shows the sequential sampling order.

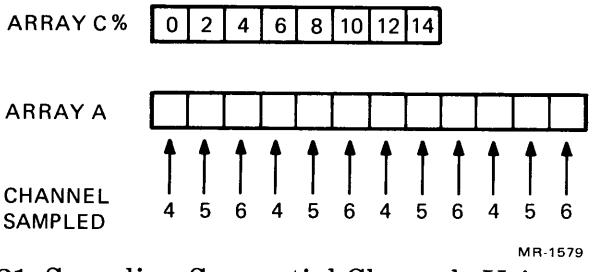


Figure 21. Sampling Sequential Channels Using an Array Element.

Notice that both of these program fragments specify the same conversion sequences.

Random Channels

The random method for specifying a conversion sequence allows you to specify nonsequential channel order. (The SET_GAIN routine, as well as the AIN, AIN_SUM, and AOUT routines, accepts the RANDOM mode designator.) In random mode, the channel order is contained in an integer array. As for sequential conversion sequences, the starting channel number is in an array element, and the no.-of-channels argument specifies the number of channels in the sequence.

For random mode, you include the mode designator RANDOM. The transfer routine then defines the rest of the conversion sequence using successive elements of the channel array.

For example, consider the array C% that we defined above, and the last program fragment. To specify random mode using the channel array C%, add the RANDOM designator to the AIN statement.

```
160 AIN('RANDOM',A(),12,,C%(2),N1)
```

Now the program fragment defines a conversion sequence that starts with the channel number specified in array element C%(2), and continues with the channels in the next two array elements, C%(3) and C%(4). Figure 22 shows the effects of this random mode sampling.

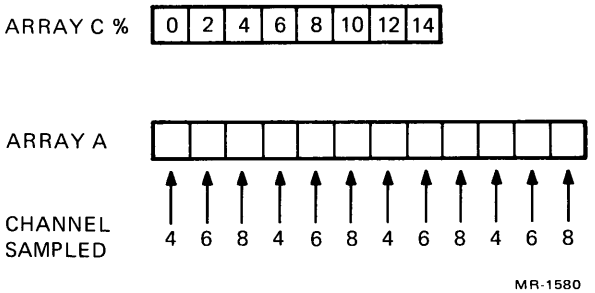


Figure 22. Sampling Order in Channel Array C%.

Array elements C%(2) through C%(4) contain the values 4, 6, and 8 and therefore, the AIN conversion sequence is channels 4, 6, and 8.

It is important to realize that AIN treats the channel array sequentially, and that you control the order of channels in the conversion sequence by changing the values in the array, not by

changing the order in which AIN takes the values from the array. For example, consider the following two channel arrays:

C1%	10	6	2	0	15	1	7	8
C2%	15	13	11	9	7	5	3	1

The same basic AIN statement now results in completely different sampling because the two arrays are different and thus the two conversion sequences are different. Figure 23 diagrams the result of executing the following statement:

```
160 AIN('RANDOM',A(),12,,C1%(2),N1)
```

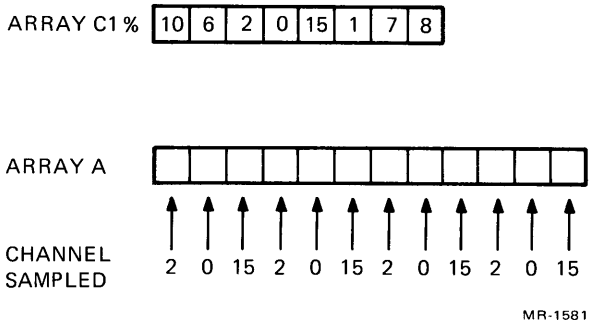


Figure 23. Sampling Order in Channel Array C1%.

Figure 24 diagrams the result of executing the following statement in which only the channel array changes.

```
160 AIN('RANDOM',A(),12,,C2%(2),N1)
```

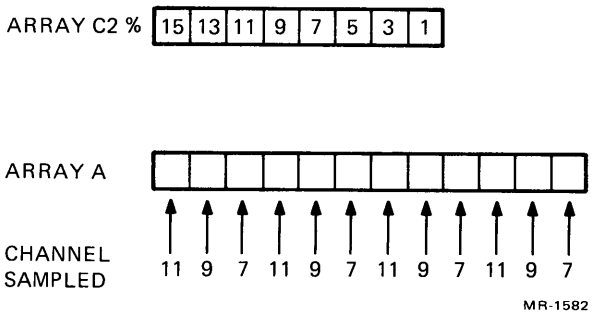


Figure 24. Sampling Order in Channel Array C2%.

TIME BASE

In the MINC lab module routines, you often see references to controlling or specifying the *time base* for some process. The term "time base" refers to periodic signals for controlling events. The time base rate determines the smallest interval detectable between two events (that is, the time resolution). MINC

has several possible sources for time base signals, two *internal* and one *external.*

Internal Time Base

The internal time base can be provided by the MINC system clock, or by the MINC clock module. These different time base sources have different uses.

The MINC system clock is located in the chassis. It is a fixed, unremovable part of the system. The system clock runs at local line frequency, either 50 or 60 Hz, depending on the location. The system clock runs whenever MINC is on, maintaining the time of day. When you use the CLK$ function or the TIME command, you are interacting with the system clock. The PAUSE, SCHEDULE, and CIN routines also use the system clock, so that the time resolution of these routines is limited by the line frequency.

The MINC clock module provides the time base for most of the lab module routines. Its potential resolution is much higher than that of the system clock (up to 1 microsecond). The lab module routines adjust the time base of the clock module according to your program's needs. Therefore, the programs (and the programmer) need not be aware of the time base chosen by the routine.

Figure 25 shows which of five time base frequencies is chosen by

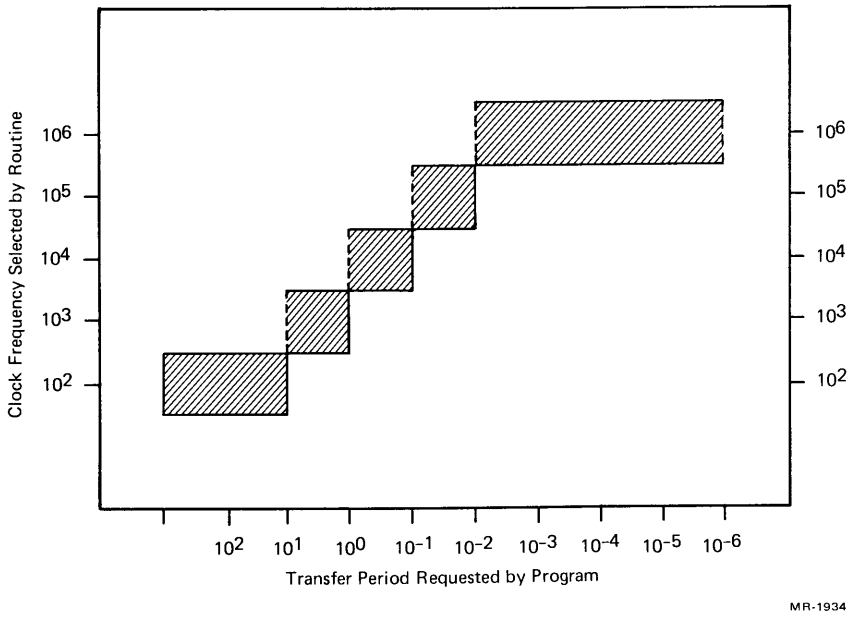


Figure 25. How a Routine Selects the Clock Frequency.

the routines for any clock trigger interval you specify in a program. You could use this information in evaluating the precision of your measurements.

For example, the following statement specifies 50 samples per second:

```
DIN(,A(),100,1/50)
```

The DIN routine would select a clock frequency of 10^5.

You can specify line-frequency time base with the LINE designator in several of the transfer routines (AIN, AIN_SUM, AOUT, DIN, DOUT, MAKE_TIME, and START_TIME). The clock module itself then runs at line frequency (50 Hz or 60 Hz) independently of the system clock.

There are only two situations in which you can specify a time base explicitly. The START_TIME and MAKE_TIME routines provide the mode designators CHZ (centiHertz) and KHZ (kiloHertz) for 100 Hz and 1000 Hz (respectively). In all other internal time base situations, you specify the transfer interval and let the routines select the time base.

External Time Base

Several of the transfer routines permit you to specify an external source for the time base signals. You connect the external timebase source to Schmitt trigger 1 of the clock module and specify the EXTERNAL designator. As a result, the transfer routine uses the external signals, rather than an internal time base, to determine intervals between events. The external time base can run at any frequency required, up to a maximum of 1 MHz.

The routines which permit an external time base are AIN, AIN_SUM, AOUT, DIN, DOUT, MAKE_TIME, and START_TIME.

FREQUENCY HISTOGRAMS

Three MINC routines (AIN_HIST, PST_HIST, TIME_HIST) generate frequency histograms. A *frequency histogram* is a bar graph that summarizes a sample frequency distribution. For example, Figure 26 is a histogram summarizing the ages of employees in a department.

The vertical axis represents *frequency*, that is, the number of occurrences of the measured characteristic, age. The horizontal axis represents the measured characteristic itself. The continuum of age is partitioned into nine age groups (called *bins*). The

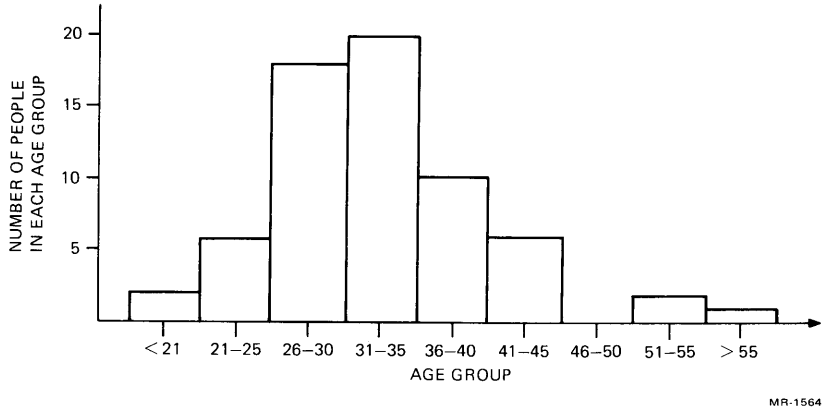


Figure 26. Age of Employee Histogram Example.

number of bins in a histogram is arbitrary. If there were fewer bins, the histogram would lose detail:

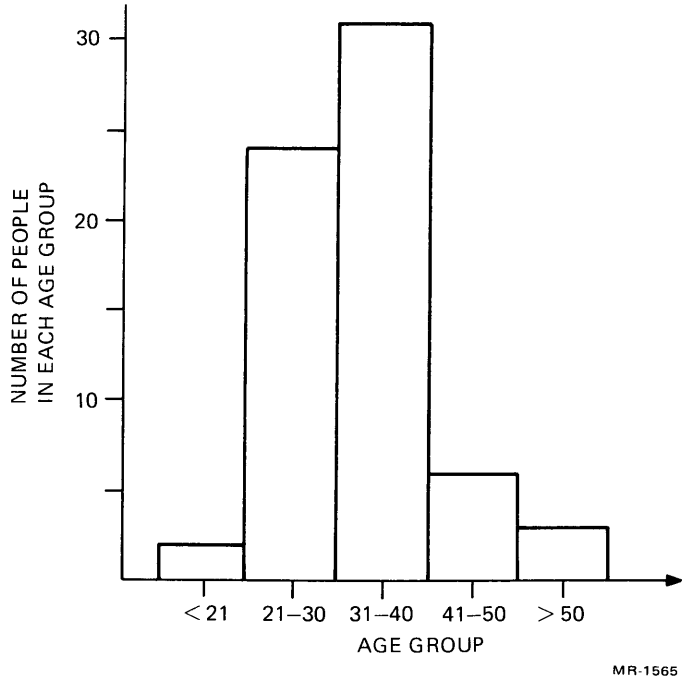


Figure 27. Age of Employee Example with Fewer Bins.

Alternatively, if there were many more bins, the histogram would lose its effectiveness as a summary (Figure 28):

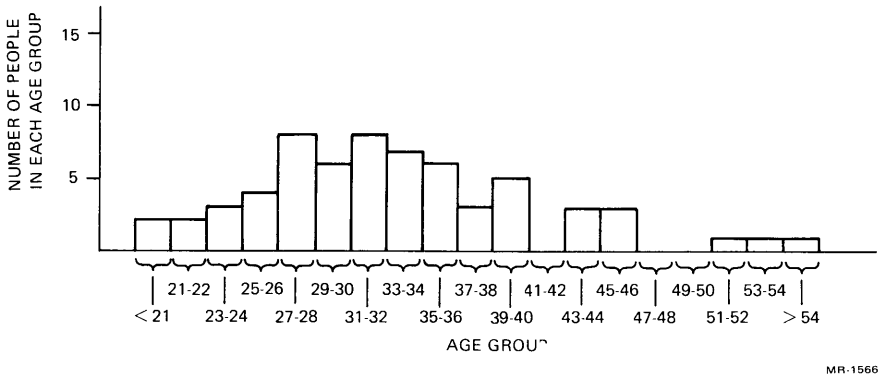


Figure 28. Age of Employee Histogram with More Bins.

One convention holds that for statistical summaries 10 to 20 bins provide a good balance between summary and detail. However, the final choice depends upon the application, the overall range of values possible, the range of interest, and the number of observations summarized by the histogram.

Definitions

The *overall range* is the set of values between the highest possible value and the lowest possible value (inclusive). In Figure 26, the overall range consists of the ages between the minimum employment age and retirement age (neither of which appears on the figure). The *range of interest* is the set of values between the highest data value of interest and the lowest data value of interest (inclusive). In the age of employee example, the highest age of interest was 55; the lowest age of interest was 20.

In many cases, the *endpoints* of the overall range of values and the range of interest are the same (Figure 29).

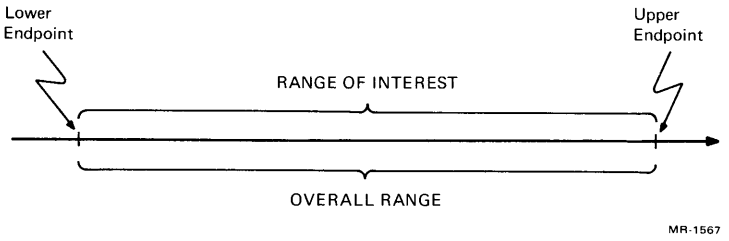


Figure 29. The Same Range of Interest and Overall Range.

In other cases, the endpoints of the range of interest lie within the overall range (Figure 30).

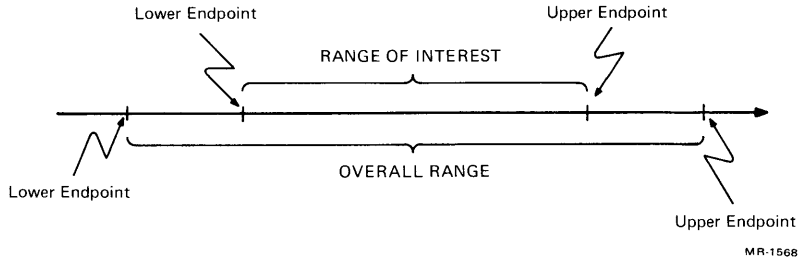


Figure 30. Different Endpoints for Range of Interest and Overall Range.

The histogram routines in MINC partition the range of interest into bins of equal width.

$$\text{bin width} = \frac{\text{range of interest}}{\text{number of bins}}$$

In the program, you specify both the number of bins and the endpoints of the range of interest. Figure 31 shows how the range is partitioned for a hypothetical histogram.

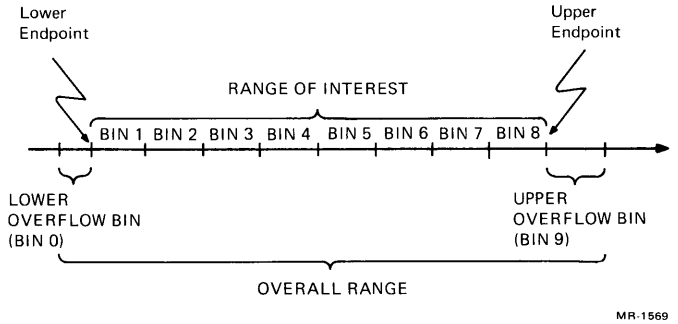


Figure 31. Partitioning the Histogram Ranges into Bins.

In Figure 31, the region of interest contains eight bins. The histogram routines count the number of data values occurring within each bin range.

In Figure 31, there are two extra bins. Bin 0 contains the number of values between the lower limit of the overall range and the lower limit of the range of interest. This bin is called the *lower overflow bin*. Similarly, bin 9 contains the number of values occurring between the upper limit of the range of interest and the upper limit of the overall range. This bin is called the *upper overflow bin*.

Discussion

Consider generating a histogram by sampling analog values. Suppose that the input signals are voltages, and that the program statement defining the histogram specified 10 bins in the

range of interest. If the range of interest were 10.24 volts, then the voltage range would be divided into 10 bins of width 1.024 volts. The following table shows how a histogram routine would define the bin ranges for this case.

Table 4. Bins and Bin Ranges.

Bin	*Endpoints of Bin Range*		
0	< -5.12		
1	-5.12	to	-4.0975
2	-4.095	to	-3.0725
3	-3.07	to	-2.05
4	-2.0475	to	-1.025
5	-1.0225	to	-0.0025
6	0	to	1.0225
7	1.025	to	2.0475
8	2.05	to	3.07
9	3.0725	to	4.095
10	4.0975	to	5.1175
11	⩾ 5.12		

Figure 32 shows a short segment of a sawtooth wave and the histogram generated by sampling 30 evenly-spaced points from the waveform. The statistical variation apparent in the figure

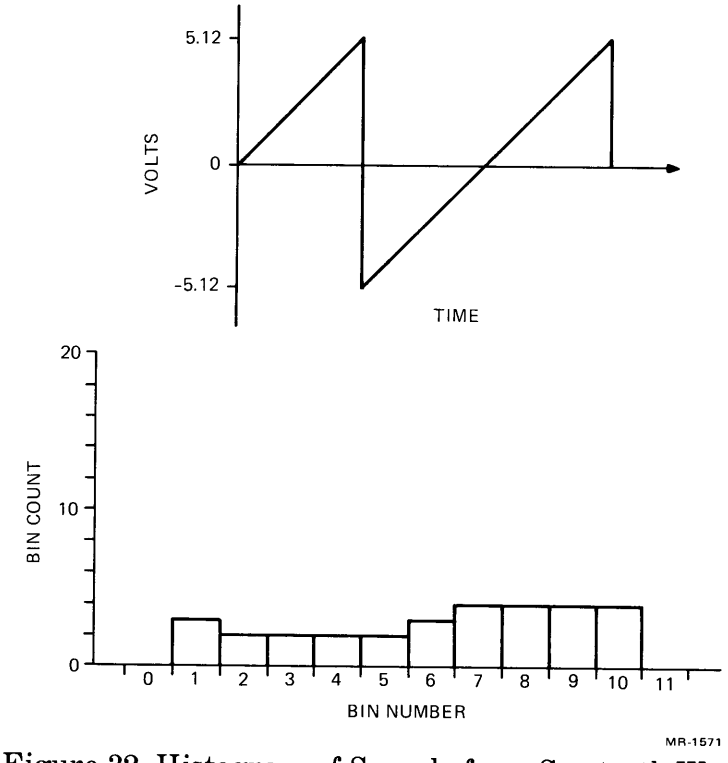


Figure 32. Histogram of Sample from Sawtooth Wave.

would disappear with prolonged sampling from the same waveform and the histogram would approach being a uniform distribution.

The example demonstrates one important point about histograms: the shape of the histogram is not the same as the waveform of the data values. The histogram preserves only the frequency of occurrence of values within specified limits; the histogram does not preserve actual data values or their sequence.

Stored Histogram Arrays

The histogram routines accumulate frequency counts in a one-dimensional array, with one array element corresponding to one histogram bin (Figure 33).

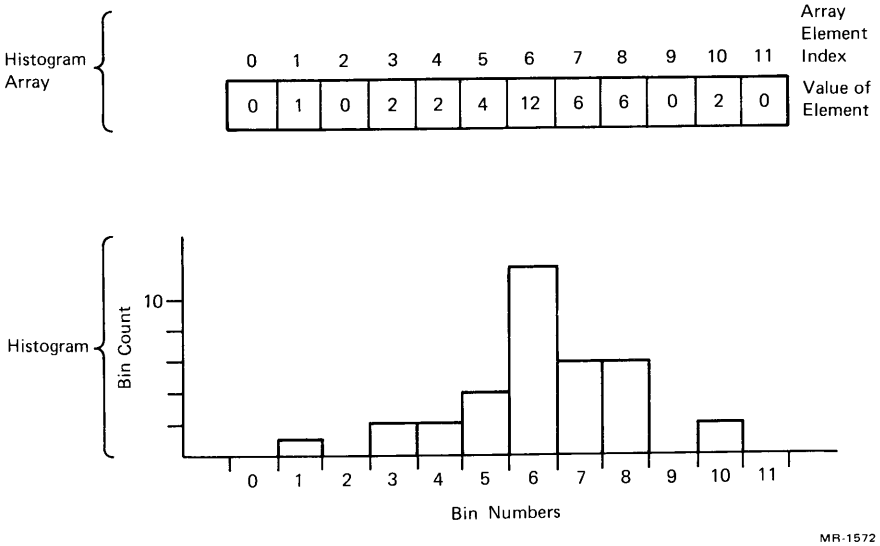


Figure 33. Correspondence of Histogram Array and Bar Graph.

The array must contain enough elements for all bins in the range of interest, and two extra elements for the lower and upper overflow bins. These extra elements are required even if the overall range and the range of interest have the same endpoints. For example, if there are 15 bins in the range of interest, the array must contain 17 elements.

```
DIM A(16)
```

The histogram routines treat element A(0) as the lower overflow bin, and A(16) as the upper overflow bin. If the overall range and the range of interest had the same endpoints, then both A(0) and A(16) would contain the value 0.

ERROR DETECTION

MINC checks all statements and commands for errors both before and during execution. Whenever it detects a problem, MINC displays an error message on the screen. The error messages have been designed to help you locate and correct errors.

Book 6 explains in detail the error messages issued by the lab module routines. All MINC error messages are summarized in Book 8.

The error messages fall roughly into three categories, depending on what the error condition is or on when the error condition is detected. *Syntax errors* are those errors detected during a preliminary validity check of the arguments in a statement. *Interaction errors* are those that arise as a result of conflicts in interactions among routines and processes, rather than as a result of any particular routine. *Execution errors* are those occurring as a result of problems in attempting to execute a particular statement.

The syntax error messages and interaction error messages appear in this section accompanied by a short explanation. The execution error messages specific to each routine appear in the error section for the routine in Part 2. Where applicable, the messages are accompanied by further explanation and suggestions for correcting the problem.

Whenever a message appears while a program is executing, MINC appends the phrase "at line nnn" to the message. The number nnn is the number of the statement which was executing when the error condition occurred. The problem causing the error is usually in the indicated statement for syntax and execution errors. However, the statement number in interaction errors bears no necessary relationship to the location of the condition causing the error.

In immediate mode, the statement executing does not have a statement number, and hence, the phrase "at line nnn" does not appear for error messages in immediate mode.

Syntax Errors

?MINC-F-Invalid character or duplicate modes requested

This message applies to the mode argument. It can occur for any routine which has a mode string. This message appears when the first character in a mode designator is invalid. The valid characters are the letters A through Z (upper or lower case). It can also occur if the same initial character appears twice in the string.

?MINC-F-Invalid data type for argument #

This message appears whenever a statement contains an invalid data type for an argument. Only the PAUSE routine (for which all data types are valid) cannot cause this message. The syntax check replaces the # shown above with the list position of the argument causing the message.

?MINC-F-Missing argument # is required

This message appears whenever the syntax check discovers that a required argument is missing from the argument list. The syntax check replaces the # shown above with the list position of the argument causing the message. This message cannot arise from SCHMITT, START_TIME, and TERMINATE, for which there are no required arguments.

?MINC-F-Previous routine is already using the module requested

This message appears whenever the syntax check discovers that the statement requires a lab module that is already busy. Most such conflicts occur with the clock module during continuous mode transfers. The routines which do not require any modules cannot cause this message, for example, MAKE_BCD, TEST_BIT, TERMINATE, PAUSE, and FFT.

?MINC-F-System does not contain the module requested

This message appears when a statement requires a module that is not installed in the system. The routines which do not require any modules cannot cause this message.

?MINC-F-Too many arguments in the statement

This message appears when the syntax check finds more arguments in the argument list than the routine can interpret. This condition can arise as a result of extra commas in the argument list.

?MINC-F-Variable name required for argument #

The message appears when an argument is a literal instead of a variable or array name. The argument must be a name rather than a literal when the routine being called assigns a value to the argument. The syntax check replaces the # shown above with the list position of the argument causing the message. This condition can arise as a result of missing or extra placeholder com-

mas in the argument list.

Interaction Errors

The following messages arise from problems in interactions between routines and processes, rather than from argument errors in a particular statement. Some of these error messages never appear when MINC is functioning properly.

?MINC-F-Data transfer or pending service request terminated

This message appears whenever you could not resume executing because some ongoing process has been terminated. If a program finishes without executing a TERMINATE statement to terminate all continuous transfers, the message appears. If you press CTRL/C to abort a program, the message appears if there is a pause in progress, if a Schmitt trigger is active, if a scheduled time event has not occurred, or if a continuous transfer is still in progress.

?MINC-F-Notify DIGITAL: Internal error trap

This message does not appear when MINC is functioning properly. It indicates a serious problem in the MINC routines.

?MINC-F-Notify DIGITAL: Mark time failure

This message does not appear when MINC is functioning properly. It indicates a serious problem in the MINC routines.

?MINC-F-Notify DIGITAL: Memory pool exhausted

This message does not appear when MINC is functioning properly. It indicates a serious problem in the MINC routines.

?MINC-F-Notify DIGITAL: Protection failure

This message does not appear when MINC is functioning properly. It indicates a serious problem in the MINC routines.

?MINC-F-Too many response requests pending

The message appears when more response requests have occurred than MINC can respond to. MINC can delay responding to response requests until previous requests have been responded to, but at most eight response requests can be pending at one time. SCHMITT, SCHEDULE, and CONTINUE all cause response requests.

CHAPTER 3
LAB MODULE PROGRAMMING EXAMPLES

Chapter 3 contains example programs for several classes of common laboratory applications. In each case, the example describes the objective for the experiment and the measurement plan that forms the basis for the program design. It outlines the instrumentation assumed by the program and the design of the program.

The examples are designed to serve as models for valid program design using the lab module routines. However, if you have an instrument whose characteristics match those of the hypothetical instruments in the examples, you can connect the instrument and run the example program. All example programs are stored on the demonstration diskette.

The examples illustrate the major classes of function provided by the lab module routines. The example section for each routine in Part 2 contains the reference to the relevant example for the routine. The following table contains the names of the routines used in each example.

Table 5. Routines Illustrated by Each Example.

Example	*Example Title*	*Routines Illustrated*
A	Analog Sweep Integration	AIN, TEST_GAIN, SET_GAIN
B	Signal Averaging	AIN_SUM, TEST_LINE, SET_LINE, PAUSE

C	Saving Continuous Input	AIN, CONTINUE, TERMINATE
D	Digital Analyzer Control	DIN, DOUT, SET_LINE, TEST_LINE, MAKE_NUMBER, MAKE_BCD, PAUSE
E	Arithmetic Quiz	START_TIME, GET_TIME
F	Animal Tracking	DIN, DIN_EVENT, WAIT_FOR_DATA, SCHEDULE
G	Time Interval Measuring	TIME_HIST
H	Oscilloscope Control	AOUT

EXAMPLE A. ANALOG SWEEP INTEGRATION

Example A illustrates AIN, TEST_GAIN, and SET_GAIN. The program name is A6INT.BAS on the demonstration diskette.

Experimental Model We have a source of analog data which we want to collect and integrate. The sampling rate is slow (50Hz) and 513 points will be sufficient. The conditions of the experiment itself determine when sampling should begin, so the experiment provides a signal indicating when the input sweep be-

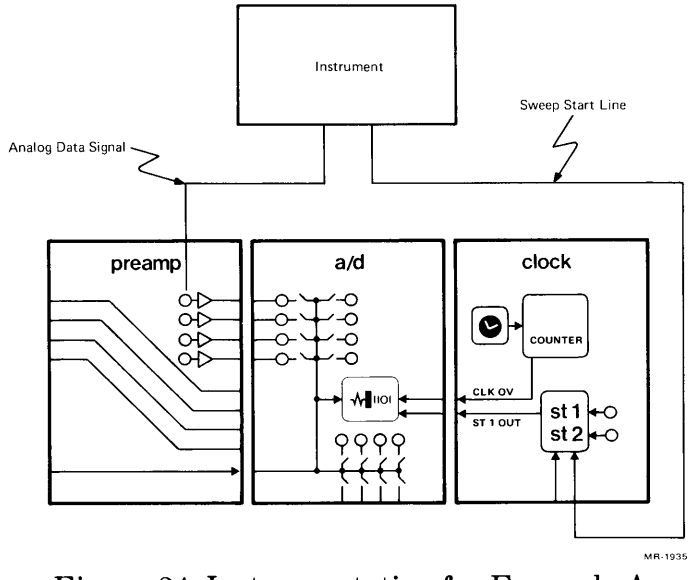


Figure 34. Instrumentation for Example A.

gins. We shall use this signal to initiate the collection of analog data.

The source of the analog data provides a voltage whose full scale range is considerably less than the full scale of the A/D converter. Therefore we need to connect the data source to a preamp module. Figure 34 shows a diagram of the instrumentation.

Program Description The program first asks for an input channel and a gain code. After obtaining these parameters, the program collects the analog data (with the beginning of the sweep triggered by a signal on ST2) and displays the values on the screen. After the data have been collected, the integral is calculated using Simpson's Rule. The result is printed on the screen and then the whole process repeats for another acquisition sweep. The program continues indefinitely until we press CTRL/C twice.

```
10 REM EXAMPLE A. AIN, TEST_GAIN, SET_GAIN
20 REM Identify the program to the user.
30 PRINT "Analog sweep integration example." \ PRINT
40 REM Define the error message format strings.
50 DIM M$(3)
60 DATA "Channel ## must be programmable."
70 DATA "Channel ## must have a preamp."
80 DATA "Channel ## must be installed in system."
90 FOR I=1 TO 3 \ READ M$(I) \ NEXT I
100 REM Prompt for the channel number and gain.
110 PRINT "Enter the channel number of the input:"; \ INPUT C
120 PRINT "Enter the gain code (0-4):              "; \ INPUT G
130 REM Verify that the preamp is properly set [P mode and volts]
140 TEST_GAIN(C,A,A$)
150 IF A<0 THEN 200
160 IF A$<>'V' THEN 210
170 SET_GAIN(,G,C)
180 GO TO 250
190 REM The channel parameters are wrong. Print a message.
200 PRINT USING M$(-A),C \ GO TO 220
210 PRINT USING "The mode on channel ## must be volt.",C
220 GOSUB 20000 \ REM Execute the ready dialog.
230 IF A=-1 THEN GO TO 130 \ REM If channel has preamp, recheck until correct.
240 GO TO 100 \ REM Otherwise ask for another channel.
250 REM Collect 513 point sweep, displaying data as they are acquired.
260 PRINT
270 PRINT "513 points will be collected. Connect the start signal to ST2."
280 GOSUB 20000 \ REM Execute the ready dialog.
290 REM Allocate the data array.
300 DIM R%(512)
310 REM Collect the data.
320 AIN('Display,ST2',R%(),513,1/50,C)
330 REM Simpsons rule for the numerical integration of a 513 point array:
340 REM
350 REM
```

```
360 REM S = ---------*R%[0]+4*R%[1]+2*R%[2]+...4*R%[511]+R%[512]
                 512
370 REM      6 * 256
380 REM
390 S=R%(0)+R%(512)
400 FOR I=1 TO 511 STEP 2 \ S=S+4*R%(I) \ NEXT I
410 FOR I=2 TO 510 STEP 2 \ S=S+2*R%(I) \ NEXT I
420 S=(S*512)/(6*256)
430 REM Print the integral value under the displayed data.
440 PRINT "Calculated integral:",S
450 REM Repeat the program.
460 GO TO 100
20000 REM Wait for Y instruction
20010 PRINT "Enter Y when you are ready to start"; \ LINPUT T$
20020 IF T$='y' THEN RETURN
20030 IF T$='Y' THEN RETURN
20040 GO TO 20010
```

EXAMPLE B. SIGNAL AVERAGING

Example B illustrates AIN_SUM, TEST_LINE, SET_LINE, and PAUSE. The program name is A6AVER.BAS on the demonstration diskette.

Experimental Model We have an instrument which provides a somewhat noisy signal of the form Y = C*EXP(-K*T), that is, an exponentially decaying signal from time (T) zero. We wish to determine the value of the two constants C and K. In addition, we wish to reduce the effects of the signal noise on our results. To do this we shall repeat the experiment 100 times, averaging the results, and then we shall fit the data to the equation using the method of least squares.

Assume that the instrument supplies the five connections shown in Figure 35.

1. The exponentially decaying analog data signal.

2. An input control signal from the instrument (trigger signal) to trigger the analog data sampling.

3. An input control signal from the instrument (sweep start) indicating that the instrument has begun a sweep and that data are available on the first connection.

4. An input control signal from the instrument (ready signal) indicating that the instrument is ready to begin.

5. An output control signal to the instrument (start signal) indicating that it should begin its operation. If the instrument is ready then it will repeatedly provide the required data on

the analog connector until this signal is cleared.

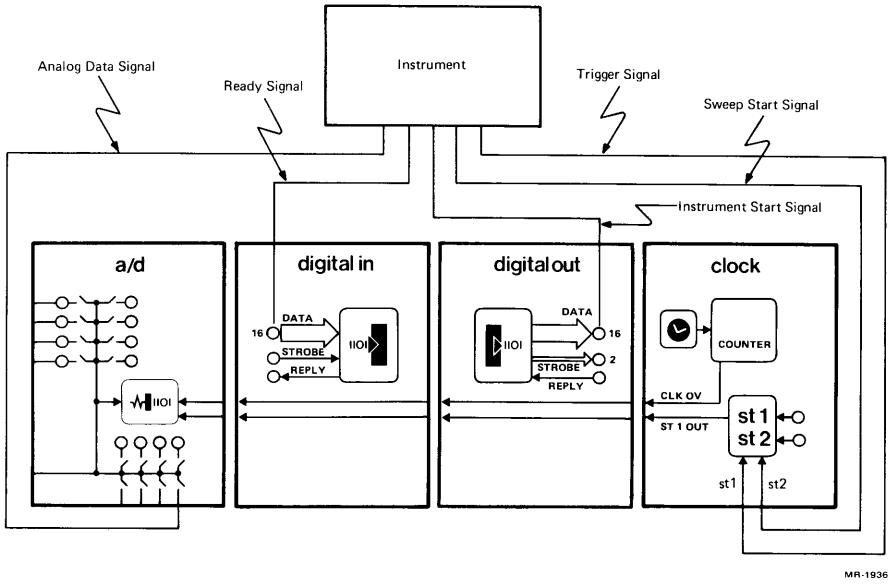


Figure 35. Instrumentation for Example B.

We connect the analog data signal to channel 3 of the A/D converter. We connect the trigger signal to ST1 of the clock module to trigger each A/D conversion. The sweep start signal is connected to ST2 of the clock module and marks the beginning of each sweep for the AIN_SUM routine. The ready signal indicating that the instrument is ready is connected to line 0 of digital input unit 0. By testing this line, we can determine when the instrument is ready and can proceed automatically. The instrument start signal is connected to line 0 of digital output unit 0. By setting this line, we start the experiment.

Program Description The program first prompts the operator to prepare the experiment and then waits for keyboard verification from the operator to proceed. Next, it waits for the instrument ready signal and sends the start signal. The program then collects 100 sweeps of 512 points each from the instrument and stops the instrument.

Using the method of least squares, the program fits the data collected (divided by 100) to the exponential decay equation and prints out the two values computed. The program repeats the acquisition sequence indefinitely until the operator enters CTRL/C.

```
10 REM EXAMPLE B. AIN_SUM, TEST_LINE, SET_LINE, PAUSE
20 REM Identify the program to the user.
30 PRINT "Signal averaging example." / PRINT
40 REM Initialize the channel, lines, and number of sweeps.
50 C=17 \ N=100 \ REM Channel, C =  17; Number of sweeps, N =  100.
60 U=0 \ S0=0 \ R0=0 \ REM Unit, U =0; Start line, S0, and ready line, R0 =  0
70 P=512 \ DIM D(511) \ REM Allocate space for 512 points.
80 PRINT "Prepare the instrument for computer control."
90 GOSUB 20000 \ REM Execute the ready dialogue.
100 REM Now wait for the instrument ready signal.
110 TEST_LINE(R0,I,U)
120 IF I>0 THEN 160
130 PRINT CHR$(7); "The instrument is not yet ready."
140 PAUSE(15)
150 GO TO 110
160 REM Start the instrument.
170 SET_LINE(S0,1,U)
180 PRINT "Acquisition now beginning."
190 AIN_SUM('Display',D(),P,,C,,N) \ REM Display, 512 points, Chan. C, N sweeps
200 PRINT "Acquisition complete." / PRINT
210 SET_LINE(S0,0,U) \ REM Stop the instrument.
220 REM Average the data.
230 FOR I=0 TO P-1 \ D(I)=D(I)/N \ NEXT I
240 REM
250 REM Assume data fits D(T)=C*EXP(-K*T)
260 REM or the relation LOG(D(I))=LOG(C)-K*T
270 REM Which is equivalent to Y =  A + B*T
280 REM A least squares fit to the linear form follows.
290 REM T0 is the average of T. T2 is the average of T^2.
300 REM Y0 is the average of Y. Z is the average of Y*T.
310 REM B = (T0*Y0-Z)/(T0^2-X2)
320 REM A = Y0 - B*T0
330 REM
340 T0=0 \ T2=0 \ Z=0 \ Y0=0 \ REM Initialize the accumulators.
350 FOR T=0 TO 511
360 Y=LOG(D(T))
370 T0=T0+T \ T2=T2+T^2
380 Y0=Y0+Y \ Z=Z+Y*T
390 NEXT T
400 REM Calculate the averages.
410 T0=T0/P \ T2=T2/P
420 Y0=Y0/P \ Z=Z/P
430 REM Calculate the slope and intercept.
440 B=(T0*Y0-Z)/(T0^2-T2)
450 A=Y0-B*T0
460 REM Convert the intercept to the exponential form.
470 A=EXP(A)
480 REM Print the results
490 PRINT "Y = C * EXP(-K * T)";
500 PRINT USING "C = #####.#### K = #####.####",A,-B
510 PRINT
520 GO TO 80 \ REM Repeat the program
20000 REM Wait for Y instruction
20010 PRINT "Enter Y when you are ready to start"; / LINPUT T$
20020 IF T$='y' THEN RETURN
20030 IF T$='Y' THEN RETURN
20040 GO TO 20010
```

EXAMPLE C. SAVING CONTINUOUS INPUT

Example C illustrates AIN, CONTINUOUS, and TERMINATE. The program name is A6SAVE.BAS on the demonstration diskette.

Experimental Model We have an instrument which provides a signal consisting of peaks and valleys. In a later analysis program, we want to detect the locations of the peaks and valleys in the waveform. To do this, we will require 20,000 points. Therefore, we know we cannot collect all the points in the workspace but must save them in a virtual array file.

Figure 36 diagrams the instrumentation.

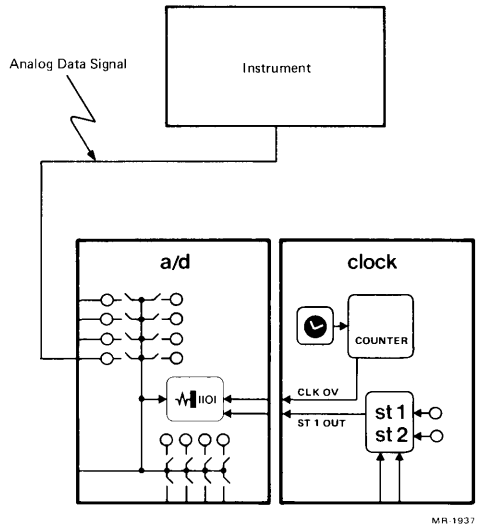


Figure 36. Instrumentation for Example C.

Program Description The program prompts the operator for the name of the file for the data points. The program then collects 25 data points per second and terminates when 20,000 points have been collected. During the data collection, the program monitors the progress of the acquisition by printing the number of samples collected.

When the stream is complete, the program saves the file and repeats the acquisition sequence indefinitely until the operator enters CTRL/C.

```
10 REM EXAMPLE C. AIN, CONTINUE, TERMINATE
20 REM Identify the program to the user.
30 PRINT "Saving continuous input example."
40 REM Initialize the channel [C], trigger interval [R] and counter [J]
50 C=5 \ R=25 \ J=0 \ K=0
60 REM Accept the file specifications.
```

```
70 PRINT "Enter the file name of the data file:                ";  / LINPUT D$
80 REM Allocate the virtual array file and open its channel.
90 DIM #1,D(19999) \ S=19999
100 OPEN D$ FOR OUTPUT AS FILE 1
110 REM Allocate the array used for the continuous acquisition.
120 DIM R(199) \ B=99
130 REM Setup the continuous acquisition process.
140 AIN('Display,Continuous',R(),200,1/R,C)
150 REM Start the acquisition and specify R0 as the array index.
160 CONTINUE(R(),R0,2000)
170 PRINT
180 PRINT "Number of data points requested:";S+1
190 PRINT "Number of data points collected:";
200 GET_CURSOR(C0,C1)
210 PRINT
220 HTEXT(,C0,C1+1,STR$(J)) \ REM print the number of points collected.
230 IF J<S THEN 220
240 HTEXT(,C0,C1+1,STR$(J)) \ REM print the number of points collected.
250 CLOSE #1
260 PRINT
270 GO TO 70
2000 REM Transfer the data to the virtual array.
2010 FOR I9=R0 TO R0+B
2020 D(J)=R(I9)
2030 NEXT I9
2040 J=J+B
2050 IF J<S THEN 2080
2060 TERMINATE('Immediate',R()) \ PRINT "TRANSFER HALTED."
2070 RETURN
2080 CONTINUE(R(),R0,2000)
2090 RETURN
```

EXAMPLE D. DIGITAL ANALYZER CONTROL

Example D illustrates DIN, DOUT, SET_LINE, TEST_LINE, MAKE_NUMBER, MAKE_BCD, and PAUSE. The program name is D6MCA.BAS on the demonstration diskette.

Experimental Model We need to control a multichannel analyzer, an instrument which performs its own data acquisition at very high speed and often with greater precision than most computers. Essentially it is an instrument which performs the same function as the AIN_HIST call except that it is very fast. We shall assume that the analyzer has 512 channels. The analyzer can be started and stopped by the computer, can be directed to take a particular number of samples, can tell the computer how many samples it has taken so far, and can be directed to dump its results out to the computer. Because all numeric quantities used to communicate with the analyzer are expressed in BCD format, the digital input and output routines can control the analyzer.

The analyzer has a set of digital control lines, one set for accepting commands and the other for sending status values. We connect the analyzer's status lines to digital input unit 0 and the analyzer's command lines to digital output unit 0.

Two other sets of lines carry data values to and from the analyzer. The analyzer expects 16-bit values to specify its sweep length. We connect the sweep length lines to digital output unit 1. The analyzer stores its data internally temporarily and can send all the stored data at once when it receives a dump command. We connect the analyzer's 16-bit data lines to digital input unit 1.

Figure 37 diagrams the instrumentation.

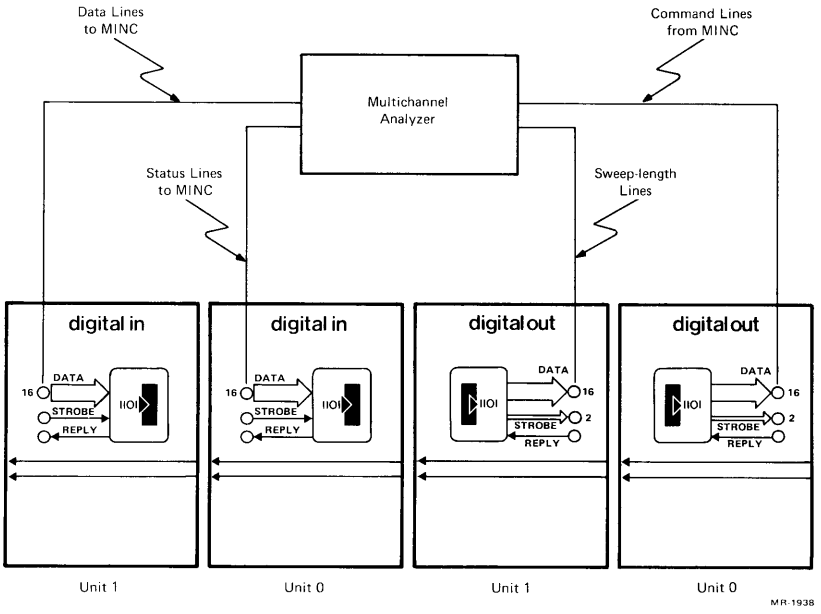


Figure 37. Instrumentation for Example D.

This analyzer can acquire up to 99,999,999 samples. The maximum BCD values that the sweep length lines can transmit is 9,999. Therefore, the analyzer can accept the sweep length specification in two parts, the four least significant digits and the four most significant digits. Similarly, the analyzer can present values up to 99,999,999 on the data lines. Therefore, it sends data values in two parts, first the four least significant digits and then the four most significant digits.

The analyzer status lines are connected to terminals on the digital input unit 0 connector block as follows:

Terminal D00. Ready line. Set when the analyzer is ready.

Terminal D01. Busy line. Set while the analyzer is acquiring data.

The analyzer control lines are connected to terminals D00, D01, and D02 of the connector block for digital output unit 0. The analyzer performs whatever function is specified by the condition of the lines. It performs the function as soon as the condition of any of the control lines changes. The following table defines the functions performed as a result of the various line conditions.

000 Initialize the analyzer.
001 Load the four least significant digits of the sweep length.
010 Load the four most significant digits of the sweep length.
011 Present the four least significant digits of the count on the data lines.
100 Present the four most significant digits of the count on the data lines.
101 Start data acquisition.
110 Stop data acquisition.
111 Dump the data acquired on the data lines.

Program Description The program prompts the operator for the number of samples to acquire. It then waits for the analyzer to set its ready line, sends the number of samples to acquire, and starts the analyzer. The program monitors the progress of the acquisition by checking the data lines every 10 seconds to see how many samples have been acquired since the beginning of the sweep and by checking the condition of the busy line. When the busy line is clear, the program commands the analyzer to dump the data values to an array and stores them in a file.

```
10 REM EXAMPLE D. DIN, DOUT, SET_LINE, TEST_LINE, MAKE_NUMBER, MAKE_BCD, PAUSE
20 REM Identify the program to the user.
30 PRINT "Digital analyzer control example." \ PRINT
40 PRINT "How many data samples do you want to take:"; \ INPUT N \ N=INT(N)
50 TEST_LINE(0,R) \ REM Test the analyzer's ready line.
60 IF R>0 THEN 100
70 PRINT "Analyzer not ready."
80 PAUSE(15)
90 GO TO 50
100 DOUT(,BIN('000')) \ REM Initialize the analyzer.
110 T0=N/10000 \ T1=T-(T0*10000) \ REM Split the count into two 4 digit halves.
120 MAKE_BCD(T0,T0) \ MAKE_BCD(T1,T1) \ REM Convert the halves to BCD.
130 DOUT(,T1,,,1) \ DOUT(,BIN('001')) \ REM Set the lower half.
140 DOUT(,T0,,,1) \ DOUT(,BIN('010')) \ REM Set the upper half.
```

```
150 PRINT "Acquisition begun.",TAB(50);CLK$
160 DOUT(,BIN('101')) \ REM Start the analyzer.
170 ERASE_TEXT('TEXT',1) \ REM Erase the first line for the sample counter.
180 REM Loop until the acquisition is done.
190 GOSUB 1000 \ REM Go get the counts done so far.
200 TEST_LINE(1,R) \ REM Check the analyzer acquiring bit.
210 IF R=0 THEN 240 \ REM Exit this loop if done.
220 PAUSE(10)
230 GO TO 180
240 REM Data has been acquired but print the sample count once more.
250 GOSUB 1000
260 PRINT "Acquisition complete.",TAB(50),CLK$
270 REM Allocate a working array and save the data on the disk.
280 DIM D%(1023)
290 DOUT(,BIN('111')) \ DIN(,D%(),1024,,1)
300 REM Accept the file name from the user.
310 PRINT "Enter the filename for the data storage:"; \ LINPUT F$
320 OPEN F$ FOR OUTPUT AS FILE #1
330 REM Store the data in the file.
340 FOR I=0 TO 1022 STEP 2
350 REM Convert from BCD and store in file F$.
360 MAKE_NUMBER(D%(I),T0) \ MAKE_NUMBER(D%(I+1),T1)
370 PRINT #1,T1*10000+T0
380 NEXT I
390 REM Close the output file, print the end message and repeat program.
400 CLOSE #1
410 PRINT \ PRINT "Data saved in file ";F$ \ PRINT
420 GO TO 40
1000 REM Get the number of samples taken and display them.
1010 DOUT(,BIN('011')) \ DIN(,T0,,,1) \ MAKE_NUMBER(T0,T0)
1020 DOUT(,BIN('100')) \ DIN(,T1,,,1) \ MAKE_NUMBER(T1,T1)
1030 C=T1*10000+T0
1040 T$=CLK$+" Samples requested: "+STR$(N)+" Samples taken: "+STR$(C)
1050 HTEXT(,1,1,T$)
1060 RETURN
```

EXAMPLE E. ARITHMETIC QUIZ

Example E illustrates START_TIME and GET_TIME. The program name is T6QUIZ.BAS on the demonstration diskette.

Experimental Model We wish to present simple arithmetic problems to a subject and measure solution latencies. We need to record both the number of correct and incorrect responses and the latencies for correct and incorrect responses.

The problems appear on the terminal screen. The subjects enter their responses on the keyboard. To measure the response latencies, we need the clock module. No connections are needed to the clock module.

Program Description The program creates arithmetic problem items consisting of two numbers and an operator, for example, 7 + 2. It chooses all three elements of the item using the RND

function to generate a random number. It chooses the operator using a random number in the range 0 to 2 for the operators +, -, and * and chooses the two digits from the set of digits 0 to 9.

The program presents the item on the screen, starts the elapsed-time clock and waits for the subject to enter the answer on the keyboard (ended by either the RETURN key or the ENTER key). When the response arrives, the program collects the elapsed time and evaluates the answer. It provides feedback to the subject concerning latency and correctness.

The program presents a sequence of 20 problems to the subject and then prints a summary of proportion correct and mean latency.

```
10 REM EXAMPLE E. START_TIME, GET_TIME, PAUSE
20 REM Identify the program to the users.
30 PRINT "Arithmetic quiz example." \ PRINT
40 PRINT "This program presents you with simple arithmetic problems to solve."
50 PRINT "The program records your score and how long you took to enter the"
60 PRINT "answer for each problem."
70 PRINT "There are 20 problems in each set."
80 PRINT
90 REM Define a MOD [or REM] function.
100 DEF FNR(X,Y)=X-(INT(X/Y)*Y)
110 GOSUB 20000 \ REM Execute the ready dialogue.
120 REM Initialize the result counters and then repeat 20 times.
130 C=0 \ W=0 \ G=0 \ B=0 \ REM Score counters, correct/wrong elapsed times.
140 FOR I=1 TO 20
150 REM Get three random numbers
160 X=FNR(INT(10*RND),3) \ REM X is in range 0 to 2.
170 Y=FNR(INT(10*RND),10) \ REM Y, Z are in range 0 to 9.
180 Z=FNR(INT(10*RND),10)
190 REM Create the problem to present.
200 ON X+1 GO TO 210,230,250
210 R=Y+Z
220 GO TO 270
230 R=Y-Z
240 GO TO 270
250 R=Y*Z
260 REM Display the problem.
270 PRINT
280 PRINT USING "## ' #=",Y,SEG$('+-*',X+1,X+1),Z;
290 START_TIME('KHZ') \ REM Start the elapsed-time counter in msec
300 INPUT A \ REM Wait for the answer.
310 GET_TIME(T) \ REM Determine the latency of the answer.
320 START_TIME('Halt') \ REM Stop the elapsed-time counter.
330 IF R<>A THEN 380 \ REM Score the answer.
340 PAUSE(.1)
350 PRINT TAB(40);"CORRECT."
360 C=C+1 \ G=G+T \ REM Increment the number correct and accumulate the times.
370 GO TO 400
380 PRINT TAB(40);"WRONG! The correct answer is ";R
390 W=W+1 \ B=B+T \ REM Increment the number wrong and accumulate the times.
```

```
400 PRINT USING "Your response took ###.## seconds.",T \ PRINT
410 PAUSE(2) \ REM Give person time to read the results
420 NEXT I
430 REM Display the results of the quiz.
440 PRINT "***...***"
450 PRINT "There were 20 problems."
460 PRINT USING "You answered ## problems correctly",C
470 PRINT USING "          and ## incorrectly.",W
480 PRINT
490 PRINT "Your average response times were:"
500 PRINT USING "                ###.## seconds for correct answers",G/C
510 IF B>0 THEN 540
520 PRINT USING "          and you had no incorrect answers."
530 GO TO 550
540 PRINT USING "          and ###.## seconds for incorrect answers.",B/W
550 PRINT
560 GOSUB 20000
570 PRINT
580 GO TO 120 \ REM Repeat with another problem set.
20000 REM Wait for Y instruction
20010 PRINT "Enter Y when you are ready to start"; \ LINPUT T$
20020 IF T$='y' THEN RETURN
20030 IF T$='Y' THEN RETURN
20040 STOP
```

EXAMPLE F. ANIMAL TRACKING

Example F illustrates DIN, DIN_EVENT, WAIT_FOR_DATA, and SCHEDULE. The program name is D6TRAK.BAS on the demonstration diskette.

Experimental Model We have an activity-monitoring experiment in which we need to know how much time an animal spends in different areas of a region over a five hour period. The region is divided into 16 zones and we have sensors in each zone which detect the presence of the animal. For the sake of simplicity we assume that the sensors detect the arrival of an animal immediately and that the sensors themselves resolve the problems arising when the animal straddles zone boundaries.

Each sensor is wired to one line of digital input unit 0. We acquire data from the sensors continuously for five hours using the timestamp mode. Therefore, we can determine the time the animal stays in each zone by the difference in times between successive arrival times in zones. The timestamp clock is driven by an external timebase with a period of 1 second. Figure 38 diagrams the instrumentation.

During the acquisition, a continuous plot of the zones is maintained on the screen, including the current position of the animal and how much time it has spent so far in each zone. The zones are

displayed as a four-by-four square with no attempt to preserve a one-to-one spatial relationship between the screen zones and the physical zones.

After five hours, the results of the experiment are saved in a file.

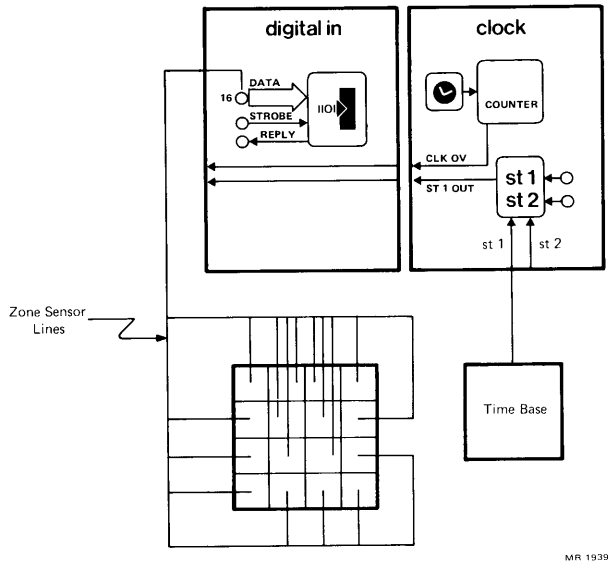


Figure 38. Instrumentation for Example F.

Program Description The program prompts the operator for the name of a file in which to save the results of the experiment. It draws the zones on the screen and initializes all of the times to zero. It then waits for an operator signal to proceed and begins collecting digital input with independent event control. Because the program runs for five hours, it repeatedly prints out a message informing any onlookers that it is busy. When the data collection period expires, the program stores the time data in the file and stops executing.

```
10 REM EXAMPLE F. DIN, DIN_EVENT, WAIT_FOR_DATA, SCHEDULE
20 REM Identify program to user.
30 PRINT "Animal tracking example." \ PRINT
40 REM Request filename from user.
50 PRINT "Enter name of data file:"; \ LINPUT F$
60 REM Draw the boxes on the screen using the following constants:
70 X9=18 \ Y9=5 \ X0=1 \ Y0=1 \ X1=73 \ Y1=21
80 REM Initialize display strings A$ and B$.
90 A$="*****" \ B$=" "
100 REM Initialize the occupancy times, T(I)
110 DIM T(15)
120 FOR I=0 TO 15 \ T(I)=0 \ NEXT I
130 REM Initialize the screen display
```

```
140 DISPLAY_CLEAR
150 ROLL_AREA(22,24)
160 PRINT TAB(31);"ZONE SUMMARY"
170 FOR I=X0 TO X1 STEP X9 \ TEXT_LINE('I,R',Y0,I,Y1,I) \ NEXT I
180 FOR I=Y0 TO Y1 STEP Y9 \ TEXT_LINE('I,R',I,X0,I,X1) \ NEXT I
190 REM Number the boxes and set the starting accumulation to 0.
200 GET_CURSOR(A,B)
210 FOR I=1 TO 4 \ FOR J=1 TO 4
220 X=(I-1)*X9+2 \ Y=(J-1)*Y9+2
230 MOVE_CURSOR(Y,X) \ PRINT USING "##",(J-1)*4+(I-1)
240 MOVE_CURSOR(Y+3,X+9) \ PRINT USING "########",0
250 NEXT J \ NEXT I
260 MOVE_CURSOR(A,B)
270 REM Track an animal using DIN and DIN_EVENT
280 REM Set the experiment timer.
290 K9=0 \ SCHEDULE('Interval','5:00:00',2000)
300 REM Set the event-enable word.
310 DIN_EVENT(BIN('1111111111111111'))
320 REM Allocate an array for acquisition of the events and timestamp values.
330 DIM S%(3) \ REM Allocate space for two events.
340 REM Start the elapsed-time counter.
350 START_TIME('External')
360 REM Start the acquisition.
370 DIN('Time,Continuous',S%(),4)
380 WAIT_FOR_DATA(S%(),Q)
390 SCAN_BIT(R,S%(Q+1)) \ REM Find the line that caused the event.
400 IF R<0 THEN 440
410 T=S%(Q) \ REM Get the time.
420 MAKE_TIME(,T,T)
430 GOSUB 1000 \ REM Update the display.
440 IF K9=0 THEN 380 \ REM If K9=1, the time is up.
450 REM Terminate process, 5 hours are up.
460 TERMINATE('Immediate',S%())
470 REM Save the results.
480 OPEN F$ FOR OUTPUT AS FILE #1
490 FOR I=0 TO 15 \ PRINT #1,T(I) \ NEXT I
500 REM Close the file and stop with the final data on the screen.
510 CLOSE #1
520 STOP
1000 REM Plot the animal position and update the occupancy time.
1010 REM Old zone is in R0, new zone is in R.
1020 P=R0 \ GOSUB 1200
1030 REM clear old mark
1040 HTEXT(,Y+1,X+6,B$) \ HTEXT(,Y+2,X+6,B$)
1050 REM Update the occupancy time of the zone just exited.
1060 T(R0)=T(R0)+T-T0
1070 T$=STR$(T(R0))
1080 HTEXT(,Y+3,X+9,SEG$("          ",1,8-LEN(T$))+T$)
1090 REM Display new mark.
1100 P=R \ GOSUB 1200
```

```
1110 HTEXT(,Y+1,X+6,A$) \ HTEXT(,Y+2,X+6,A$)
1120 T0=T \ R0=R
1130 RETURN
1200 REM Convert P into X and Y coordinates.
1210 Y0=INT(P/4) \ Y=Y0*Y9+2
1220 X0=P-(Y0*4) \ X=X0*X9+2
1230 RETURN
2000 REM Set the timeout flag.
2010 K9=1
2020 RETURN
```

EXAMPLE G. TIME INTERVAL MEASURING

Example G illustrates TIME_HIST. The program name is T6SPIN.BAS on the demonstration diskette.

Experimental Model We have a machine with a spinning rotor and we wish to monitor its speed over a long period of time. We expect some variations in its rotation rate but wish to verify that variation is within our tolerance limits over long periods of time.

We have a sensor on the machine which tells us each time the rotor has turned exactly once. Therefore, we can monitor the time interval between the signals from the sensor. We would like to observe the behavior of the the machine for about 24 hours. Figure 39 diagrams the instrumentation.

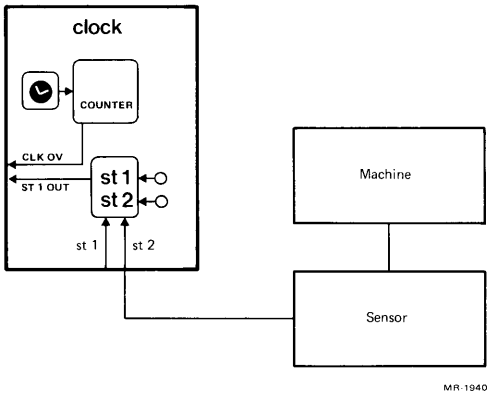


Figure 39. Instrumentation for Example G.

The basic rotation rate is 360 rpm and we expect no variations greater than 10 percent in either direction.

Program Description The program prompts the operator for the name of a file in which to save the data. The program then begins the acquisition of the data. It cannot execute TIME_HIST for 24 hours because the counts in the histogram

bins would overflow. Instead, it executes TIME_HIST 26 times with a 20,000-point sweep each time. After each sweep, it adds the counts in the histogram bins to a separate real array. After the final sweep is complete, it saves the histogram counts in a file for later processing.

```
10 REM EXAMPLE G - TIME_HIST
20 REM Identify the program to the user.
30 PRINT "Time interval measurement example." \ PRINT
40 REM Request a filename.
50 PRINT "Enter a data filename in which to save the results:"; \ LINPUT F$
60 REM Open the file.
70 OPEN F$ FOR OUTPUT AS FILE #1
80 REM Allocate the histogram and storage arrays.
90 DIM H%(511),S(511)
100 REM Begin the acquisition
110 FOR I=1 TO 26
120 TIME_HIST('Zero,Display',H%(),5,20000,15152,18520)
130 FOR J=0 TO 511 \ S(J)=H%(J)+S(J) \ NEXT J
140 NEXT I
150 FOR I=0 TO 511
160 PRINT #1,S(I)
170 NEXT I
180 CLOSE #1
```

EXAMPLE H. OSCILLOSCOPE CONTROL

Example H illustrates AOUT. The program name is A6SCOP.BAS on the demonstration diskette.

Experimental Model We have previously calculated a set of X-Y coordinate pairs and stored them in a file. We need to plot them on a storage oscilloscope because the coordinates represent a complex, nonwaveform function which could not be plotted on the MINC screen.

We connect the oscilloscope intensify line to Bit 0 of D/A channel 3. We connect the erase line to Bit 1 of D/A channel 3. In addition, we connect the X axis to D/A channel 0 and the Y axis to D/A channel 1. Figure 40 diagrams the instrumentation.

The coordinate file contains pairs of numbers in the range -5000 to +5000. The oscilloscope expects values in the range -5 volts to +5 volts.

Program Description The program prompts the operator for the name of the data file and issues instructions for setting the D/A voltage ranges of channels 0 and 1 to the ±5 volt range. It waits for an operator signal to proceed with the display. The program first erases the screen. It then retrieves the coordinate

pairs from the file, converts them to the range required by the D/A converter, and sends them to the D/A converter. When it reaches the end of the file, the program closes the file and prompts the operator for a new file name. It repeats indefinitely until the operator enters CTRL/C.

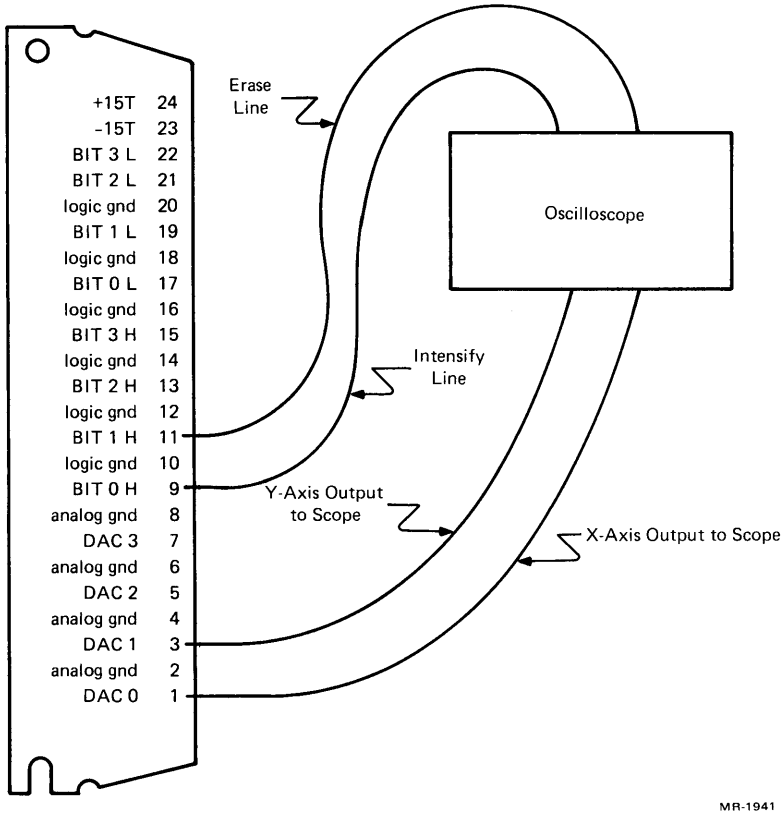


Figure 40. Instrumentation for Example H.

```
10 REM EXAMPLE H. AOUT
20 REM Identify program to user
30 PRINT "Analog output example." \ PRINT
40 REM Request a data filename.
50 PRINT "Enter data filename:"; \ LINPUT F$
60 REM Tell user to set the channels.
70 PRINT \ PRINT "Set channels 0 and 1 to bipolar mode, 5 volt range." \ PRINT
80 GOSUB 20000 \ REM Execute the ready dialogue.
90 REM Open the user specified file.
100 OPEN F$ FOR INPUT AS FILE #1
110 REM Erase the oscilloscope screen by sending control bit 1 of channel 3.
120 AOUT(,BIN('10'),,,3)
130 REM Repeat until end of file on #1
140 IF END #1 THEN 220
150 INPUT #1,V(0),V(1)
```

```
160 IF ABS(V(0))>5000 THEN 250
170 IF ABS(V(1))>5000 THEN 250
180 REM 5000/1000 =  VOLTS. VOLTS/2.5E-3 =  DAC output value.
190 V(0)=V(0)/2.5 \ V(1)=V(1)/2.5
200 AOUT(,V(),2,,0,2)
210 GO TO 140
220 PRINT "Data have been plotted."
230 PRINT
240 GO TO 50
250 PRINT "Value exceeds range +/-5000. X: ";X;" Y: ";Y
260 GO TO 140
20000 REM Wait for Y instruction.
20010 PRINT "Enter Y when you are ready to start"; \ LINPUT T$
20020 IF T$='y' THEN RETURN
20030 IF T$='Y' THEN RETURN
20040 GO TO 20010
```

PART 2
ROUTINES

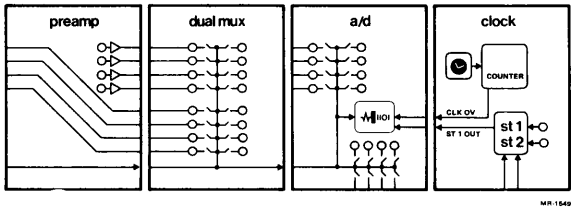
Part 2 contains the reference descriptions for the lab module routines. The reference sections appear alphabetically for convenient reference use. The structure within each reference section is explained in "Syntax Conventions," page 13, and "Argument Conventions," page 21.

AIN
Collect Analog Input

Operation

AIN collects analog data using the A/D converter. AIN can collect a single sample point, a sweep of data, or a continuous stream of data. (See "Analog Signal Processing," page 1.)

Configuration



Statement Form

AIN(mode,**data-name**,data-length,trigger,A/D-channel,no.-of-channels)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	CONTINUOUS,DISPLAY, EXTERNAL,FAST,LINE, RANDOM,ST2	standard mode
data-name	numeric variable or array name	-2048 to 2047; full-scale values	required argument
data-length	numeric expression	⩾1	1
trigger	numeric expression	0; > 0 to 655.35; 1 to 65,535	0
A/D-channel	numeric expression or integer array	0 to 63	0
no.-of-channels	numeric expression	1 to 64 or channel array length	1

Example AIN(,V)
Result Collect one value from channel 0 immediately. Put the value in variable V.

Example AIN(,V(),4,,,4)
Result Collect four values from the conversion sequence channels 0, 1, 2, and 3, starting the conversion sequence immediately. Put the values in array V.

Example AIN(,V(),100,1/100,8,4)
Result Collect 100 values from the conversion sequence channels 8, 9, 10, 11, one sequence every one-hundredth of a second. Put the values in array V.

Argument Descriptions

mode The character string selecting optional modes for AIN. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
CONTINUOUS	Use continuous mode. (See "Continuous Data Transfers," page 33.)
DISPLAY	Use display mode to monitor the data collected with a stripchart display on the screen.
EXTERNAL	Enable external source connected to ST1 to provide the time base (see "Time Base," page 43.)
FAST	Use fast mode. No autogain. Collects bipolar integer data arrays only.
LINE	Use line frequency time base (50 or 60 Hz).
RANDOM	Use random mode for sampling nonsequential channels. (See "Random Channels," page 42.)
ST2	Start data collection process with a signal on Schmitt trigger 2.
Default:	Standard mode (point or sweep transfer with either instrument trigger control of sampling, trigger = 0, or clock module control of sampling, trigger > 0 to 655.35).

Some combinations of modes are valid and some are not. In the following table, invalid mode combinations are marked with an "X."

(mode = DISPLAY) In display mode, AIN produces a plot of the data which appears to move across the screen. (The display is similar to the stripchart displays produced by the GRAPH routine with the MOVE option. See Book 4.) The screen holds 512 data points at a time. New data points are added from the right, and old data points are lost from the left of the display. (Note: the data array contains all of the points collected, not just the points currently shown on the screen.) The maximum rate of movement of the trend display is lower than the maximum data collection rate. In order to keep pace with the data being collected,

	CONTINUOUS	DISPLAY	EXTERNAL	FAST	LINE	RANDOM	ST2
CONTINUOUS	X			X			
DISPLAY		X		X			
EXTERNAL			X		X		
FAST	X	X		X		X	
LINE			X		X		
RANDOM				X		X	
ST2							X

MR 1925

AIN drops points from the display. Therefore, with rapid sampling, not all points collected appear on the screen display.

The first AIN statement with display mode clears the screen before beginning the display. Subsequent display mode use of AIN appends points to the existing display. To clear the screen, use DISPLAY_CLEAR (see DISPLAY_CLEAR, Book 4).

The display includes a horizontal axis but no vertical axis or units. The data are displayed in bipolar mode without any gain conversion. Full vertical scale on the display corresponds to full scale of the A/D converter. For autogain conversions, full scale on the display corresponds to a gain of 0.5. Therefore, with autogain, the display shows the relative magnitude of the signal before conversion, not the result of the conversion.

Display mode produces a display suitable for monitoring purposes. Use the specialized graphic routines for labelled displays. (See Book 4.)

(mode = FAST) In fast mode, AIN collects the points within a conversion sequence faster than normal. Therefore, collecting each sequence is somewhat faster and the trigger intervals can

be shorter. The actual increase depends on the values of other arguments. The fastest case is single-channel sampling with external signal control (trigger = 0). The slowest case is multichannel sampling with clocked control (internal or external, trigger > 0). Fast mode is not valid in combination with many mode options (see the table above). Fast mode is further limited because the data array must be integer and the A/D channels must be set for fixed gain. AIN terminates execution if it encounters an autogain channel in fast mode. (See also the Restrictions section.)

data-name The numeric variable or array to contain the data collected.

Values: -2048 to 2047 for integer data-name argument

Actual value of the input signal for real data-name argument (appropriate to gain and mode selected)

Default: Required argument. AIN assigns data values to the argument.

If the data-name argument is a variable name, then AIN collects a single sample point and stops sampling. If the data-name argument is an array name, AIN collects a sweep of data (containing the number of points in data-length) in standard mode, and a stream of data (with no length specified) in continuous mode.

If the data-name argument is a real variable or array, AIN collects the data in units appropriate to the gain and mode selected by the preamp controls. If there is no preamp, the data values are in the range ±5.12 volts. Refer to Book 7 for description of the ranges and units obtainable with the preamp controls.

If the data-name argument is an integer variable or array, AIN collects the data as bipolar integers. For integer data names, AIN does not permit autogain because the range of values possible (the dynamic range) is too great to be represented by integers.

data-length The length of the data-name argument.

Values	*Meaning*
1	The data-name argument is a numeric expression.

> 1	The data-name argument is an array name and the data-length argument specifies the number of array elements to be filled with data.
Default:	1

If the data-length argument is greater than 1, the data-name argument must be an array name. For a sweep of data, the data-length argument is the total number of data points to be collected.

For continuous mode transfer, the data-length argument contains the number of elements in the data-name array. (See "Array Partitions," page 34.)

The data-length argument need not be an even multiple of the number of channels in a conversion sequence. If it is not, AIN stops when the array is full regardless of whether or not the conversion sequence is complete.

trigger The code which specifies how to control the data transfer.

Values	*Meaning*
0	External signal control. (Signal line connected either to the external start terminal of the A/D connector block or to ST1 of the clock module.)
> 0 to 655.35	Internal clock control (without mode designators for external control).
1 to 65,535	External time base (with EXTERNAL or LINE designators).
Default	0

(trigger = 0) When the trigger argument is zero, A/D conversion sequences are triggered by external signals. If the data-length argument is greater than 1, AIN collects a conversion sequence whenever a signal occurs on an external signal line. AIN collects the first conversion sequence when the first external signal occurs. The signal line can be connected either to the external start terminal of the A/D converter or to ST1. (See Book 7.)

An exception to this occurs when only one conversion sequence is to be collected. If the data-length argument is less than or equal to the no.-of-channels argument, AIN collects the sequence immediately (no external signal required). If you need external

signal control for a single conversion sequence, connect the trigger line to ST1 of the clock and use external mode with trigger argument value 1.

(trigger > 0 to 655.35) When the trigger argument is greater than zero, and none of the mode designators specifies an alternative time base, the clock module triggers the conversion sequence. In this case, the trigger argument specifies the trigger interval in seconds. The trigger interval is the time from one trigger event to the next. The longest trigger interval possible is 655.35 seconds. The shortest trigger interval possible depends on the operating mode, the length of the conversion sequence, and on values of the other arguments.

(trigger = 1 to 65,535) In external mode, an external time base connected to ST1 provides the trigger events for AIN. In line mode, the line frequency mode of the clock module provides the trigger events for AIN. The time base can have a regular rate (for example, line mode) or a varying rate. In either case, the trigger argument specifies the number of external signals required before a conversion sequence occurs. The maximum number possible is 65,535. For example, if the trigger argument is 1, every external signal triggers a conversion sequence. If the trigger argument is 15, every fifteenth signal triggers a conversion sequence.

A/D-channel The first channel in the conversion sequence.

Values: 0 to the highest-numbered channel installed in the system

Default: 0

The valid channel numbers depend on which analog modules are installed in the system, and on how the instruments are connected to the A/D-channels (Book 7). The program halts with an error if you specify a channel that is not present.

The A/D channel argument can be either a numeric expression or an integer array name. If it is a numeric expression, the conversion sequence consists of that channel and as many higher-numbered channels as are specified by the no.-of-channels argument. If it is an integer array in random mode, the conversion sequence consists of as many channels in the array as specified by the no.-of-channels argument. (see "Random Channels," page 42).

no.-of-channels The number of channels defining a conversion sequence for AIN.

Values: 1 to the maximum number of channels on the system (for sequential channels).

1 to the length of the channel array (in random mode).

Default: 1

The no.-of-channels argument defines the number of points in a conversion sequence. AIN collects the number of points in a conversion sequence each time a trigger event occurs.

If the A/D channel argument is a numeric expression, then the conversion sequence is defined by that channel and the number of sequential channels specified by no.-of-channels. For example, if the channel number is 2 and the number of channels is 6, then the conversion sequence is channels 2 through 7.

If the A/D channel argument is an integer array and random mode is specified, the conversion sequence is defined by the channel numbers contained in the array. For example, suppose the channel array contains the values 6, 0, 1, 4, 7, 5, and 3, the A/D-channel argument is C(2), and the number of channels is 4. Then the AIN conversion sequence is channels 1, 4, 7, and 5.

Related Routines

CONTINUE CONTINUE allows the program to resume execution during a continuous mode transfer and to continue executing until the current array partition fills. When an array partition fills, CONTINUE transfers control to its service subroutine which processes the full array partition while the other partition continues to fill. (See "Continuous Data Transfers," page 33, and CONTINUE.)

TEST_GAIN, SET_GAIN TEST_GAIN and SET_GAIN test and set the condition of the preamp connected to any A/D channel.

TERMINATE TERMINATE terminates continuous mode data collection.

WAIT_FOR_DATA WAIT_FOR_DATA stops the program during a continuous mode transfer and the program waits until the current array partition fills. When the array partition fills, the program resumes executing to process the full array partition while the other one continues to fill. See "Continuous Data Transfers," page 33 and WAIT_FOR_DATA.

Restrictions

Autogain No autogain in fast-sweep mode.

No autogain if the data-name argument is integer.

In autogain conversion, the A/D converter does two conversions for each data point collected (see SET_GAIN). (The first conversion determines the gain range to use; the second conversion produces the data point). If the amplitude of the data signal is varying more rapidly than the conversion rate, the signal could have changed to a value outside the gain range selected by the first conversion.

CONTINUE Only one continuous transfer using the A/D converter can be in progress.

If the array partition length is not an even multiple of the conversion sequence length, then the conversion sequence points cross the array partition boundaries. AIN takes longer to acquire those conversion sequences which cross partition boundaries than those conversion sequences which do not.

Data type and gain For an integer data-name argument, the values collected are always in the range -2048 to +2047, regardless of the fixed gain specified. For example, with a gain of 5, the full input range is -1.024 volts to +1.0235 volts. The data value -2048 corresponds to input signal -1.024 volts; the data value +2047 corresponds to input signal +1.0235 volts; the data value 0 corresponds to input signal 0 volts. The program must convert the values to restore the original signal scale.

If the data are collecting in a real array, the values always reflect the actual input signal range. That is, with a gain of 5, the full input signal range is -1.024 volts to +1.024 volts. The data collected have values in the range -1.024 to +1.0235 with resolution of 0.5 millivolts. With autogain, the resolution ranges from 5 microvolts to 5 millivolts (depending on the absolute value of the signal).

In either case, the values in the data-name argument have the units specified by the front panel switch on the preamp. If there is no preamp connected to the channel, the units for that channel are volts. If the channel is connected to a preamp, then the units for the data can be volts, milliamps, or kiloohms, depending on the front panel setting (see also TEST_GAIN). That is, if the data value is 1, that value could represent 1 mA, 1 volt, or 1 kohm, depending on the front panel setting of the preamp module.

External Do not confuse external *signal control* (trigger = 0) with external *mode* (mode designator = EXTERNAL and trigger $\geqslant$ 1). They are similar methods of controlling transfer but not the same. With a trigger argument of 0, every external signal triggers a conversion sequence. In external mode, the trigger argument specifies how many time base signals are necessary to trigger a conversion sequence.

It is easy to confuse these cases because both can use ST1 to connect the trigger line. In fact, external signal control and external mode have exactly the same effect in the case where every signal on ST1 triggers a conversion sequence. For example, the following two statement have exactly the same effect (when the signal line is connected to ST1, not to the external start terminal of the A/D connector block).

```
AIN (,V(),10)
AIN ('EXTERNAL',V(),10,1)
```

Fast mode In some cases, fast mode attains its higher-than-normal speeds because AIN can use all of the system's computing resources. In these cases, all of the system time-keeping functions are suspended until AIN is finished. This means that the system clock loses time if AIN runs often or for long in fast mode.

During a fast mode transfer, entering a single CTRL/C stops program execution.

Immediate mode Point and sweep AIN transfers operate in immediate mode; continuous mode transfers do not.

Resolution The positive full scale and negative full scale of the A/D converter are not symmetric around 0. Although the book refers to voltage ranges like "±5.12 volts," the actual range of values is -5.12 volts to +5.1175 volts. See Book 7 for further details.

Sampling rates The arguments for AIN allow you to specify the trigger intervals, that is, the interval from the beginning of one conversion sequence to the beginning of the next. However, you have no direct control over the sampling rate within a conversion sequence. (The rate of conversion within a conversion sequence is always higher than the rate you could obtain with the shortest trigger interval possible for single-channel sampling.)

The conversions within a sequence always occur as fast as possi-

ble given the length of the conversion sequence, the number of autogain channels, the data type of the data-name argument, and the operating modes specified. The CONTINUOUS, RANDOM, and DISPLAY mode designators all slow the conversion rate possible within a conversion sequence and their effects are additive.

Given the number of variables involved, you cannot predict easily whether or not any particular sampling rate would execute without error. The feasibility of particular desired rates must be determined empirically.

ST1 conflict The A/D converter and the clock module interact during most forms of analog sampling. The elapsed-time routines (START_TIME and GET_TIME) use the clock module so that all clock-controlled A/D sampling is incompatible with elapsed-time measurement in systems with only one clock module.

Most externally-controlled A/D sampling is also incompatible with elapsed-time measurement. One class of externally controlled A/D sampling is compatible with elapsed-time measurement, that is, direct triggering of the A/D converter (trigger =0). The external signal line must be connected to the external start terminal of the A/D converter (not to ST1 of the clock module).

Errors

?MINC-F-Another transfer is in progress for the array specified

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Data lost—transfer rate too high

The conversion sequence takes longer than the trigger interval.

?MINC-F-Data-name array is shorter than sweep length requested

The number of elements available for data is less than the length of the sweep.

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for AIN.

The AIN statement specifies incompatible mode designators. Refer to the table of valid mode designator combinations.

One of the mode designators specified is incompatible with another argument value.

?MINC-F-No autogain channels permitted

?MINC-F-No. of channels exceeds length of integer channel array

The number of elements in the channel array must be greater than or equal to the no.-of-channels argument.

?MINC-F-Too many transfers in progress simultaneously

Examples

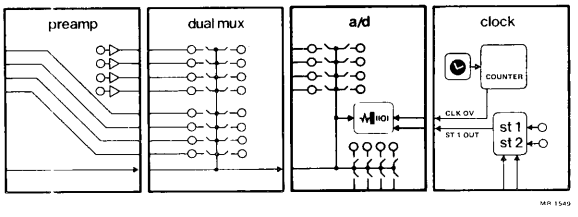
See Example A, page 56 and Example C, page 61.

AIN_HIST
Generate Analog Input Histogram

Operation

AIN_HIST collects a sweep of data from a single A/D channel and generates a frequency histogram array using the data values. The beginning of the sweep and each point in the sweep are triggered by signals on a trigger line connected to ST1 or to the external start terminal of the A/D converter. (See "Frequency Histograms," page 45.)

Configuration



Statement Form

AIN_HIST(mode,histogram-name,A/D-channel,sweep-length,lower-endpoint,upper-endpoint)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	DISPLAY,ZERO	standard mode
histogram-name	integer array name	0 to 32,767	required argument
A/D-channel	numeric expression	0 to 63	required argument
sweep-length	numeric expression	1 to 32,767	required argument
lower-endpoint	numeric expression	within channel range	negative full scale
upper-endpoint	numeric expression	within channel range	positive full scale

Example AIN_HIST(,V%(),0,100)
Result Collect a 100-point sweep from channel 0. Generate the histogram using the full channel range, and store the histogram in array V%.

Example AIN_HIST('DISPLAY',H%(),1,256)
Result Collect a 256-point sweep from channel 1. Generate the histogram using the full channel range, display the histogram and store it in array H%.

Example AIN_HIST(,A%(),4,512,-5.0,5.0)
Result Collect a 512-point sweep from channel 4. Generate the histogram using the range -5.0 volts to +5.0 volts, and store the histogram in array A%.

mode The character string selecting an optional mode for AIN_HIST. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
DISPLAY	Display the generated histogram on the screen.
ZERO	Set all elements of the histogram array to zero before starting.
Default:	Standard mode (no display; use existing array)

The combination of mode designators is valid.

(mode = DISPLAY) In display mode, AIN_HIST displays the contents of up to 512 elements of the histogram array on the screen.

AIN_HIST clears the screen before beginning the display. If the screen already contains a histogram display, AIN_HIST clears only the graph region and leaves the scrolling area unchanged. The display includes a horizontal axis. The count data are continuously redisplayed with the data scaled so that the full vertical scale of the screen represents the largest current count value. Therefore, the display indicates the relative counts in the histogram bins, but not the absolute counts. The count values are shaded.

histogram-name The name of the integer histogram array to be generated. (See "Frequency Histograms," page 45.)

Values:	0 to 32,767
Default:	Required argument. AIN_HIST assigns values to the histogram.

AIN_HIST uses the full length of the array to store the histogram. There is no argument for specifying the number of bins in the histogram. The number of elements in the array is the number of bins in the histogram. The histogram array must contain at least four elements.

AIN_HIST reserves the first and last elements in the array as the upper and lower overflow bins. Therefore, the array must contain two elements more than required for the range of interest.

A/D-channel The A/D channel carrying input data.

Values: 0 to the highest-numbered channel installed in the system.

Default: Required argument

The valid channel numbers depend on which analog modules are installed in the system, and on how the instrument is connected to the A/D channel. (See Book 7.)

sweep-length The number of data points to be collected to generate the histogram.

Values: 1 to 32,767

Default: Required argument

lower-endpoint The minimum signal value expected (lower endpoint of the range of interest).

Values: The minimum value must be greater than or equal to the negative full scale value, and within the range of values for the specified channel.

Default: Negative full scale at the current channel gain and mode setting.

The minimum value specified must be within the current channel range. The current channel range depends on the gain and mode settings for the channel. See the full discussion under "Restrictions" in this section.

upper-endpoint The maximum signal value expected (upper endpoint of the range of interest).

Values: The maximum value must be less than or equal to the positive full scale value, and within the overall range for the specified channel.

Default: Positive full scale at the current channel gain and mode setting.

The maximum value specified must be within the current channel range. The current channel range depends on the gain and mode settings for the channel. See the full discussion under "Restrictions" in this section.

Related Routines

AIN AIN is the general analog data collection routine. It performs operations not available with AIN_HIST, for example, multichannel sampling, continuous sampling, autogain, and clock-controlled sampling. If these more powerful analog collection operations are necesary, then you can collect the data with AIN and generate the histogram with program statements after the data have been collected. (See "Frequency Histograms" for general histogram principles, page 45.)

TEST_GAIN, SET_GAIN AIN_HIST permits fixed-gain data conversions, but not autogain. Use TEST_GAIN to test the current gain and mode settings on the required channel. Use SET_GAIN to select the required fixed gain. (See SET_GAIN, TEST_GAIN, and Book 7.)

Restrictions

Autogain AIN_HIST does not permit autogain sampling. Fixed-gain sampling is valid.

External control AIN_HIST expects a signal on a line connected to the external start terminal of the A/D converter or on ST1. That is, AIN_HIST sampling is controlled only by external signals, not by time base signals.

Maximum counts When an element of the histogram array reaches 32,767, AIN_HIST locks the bin. No further counts can be recorded for that bin. Thus, the bin count cannot cause an error by exceeding the integer range.

Range of values The valid minimum and maximum values depend on the current gain and mode settings for the specified A/D channel. The front panel controls (or SET_GAIN) define the overall range. Set the gain with SET_GAIN (or with the front panel control) and set the mode with the front panel control. For example, if the front panel controls specify volts at range 10, then the A/D channel range is -10.24 volts to +10.235 volts. (See Book 7.) The specified minimum and maximum values must fall within this range.

AIN_HIST tests only whether the minimum and maximum values are valid, not whether they are sensible. That is, with the current channel range set to ±5.12 volts, the following AIN_HIST statement is still valid:

```
AIN_HIST('DISPLAY',H%(),C,200,-.9,.9)
```

In this situation, the equipment configuration defines the overall range as -5.12 volts to +5.12 volts but the statement defines

the range of interest as -0.9 to +0.9 volts.

Suppose the signal shown in Figure 41 is being converted. (This is the triangle wave signal supplied by A/D channel 3 when the front panel knob is set to TEST.)

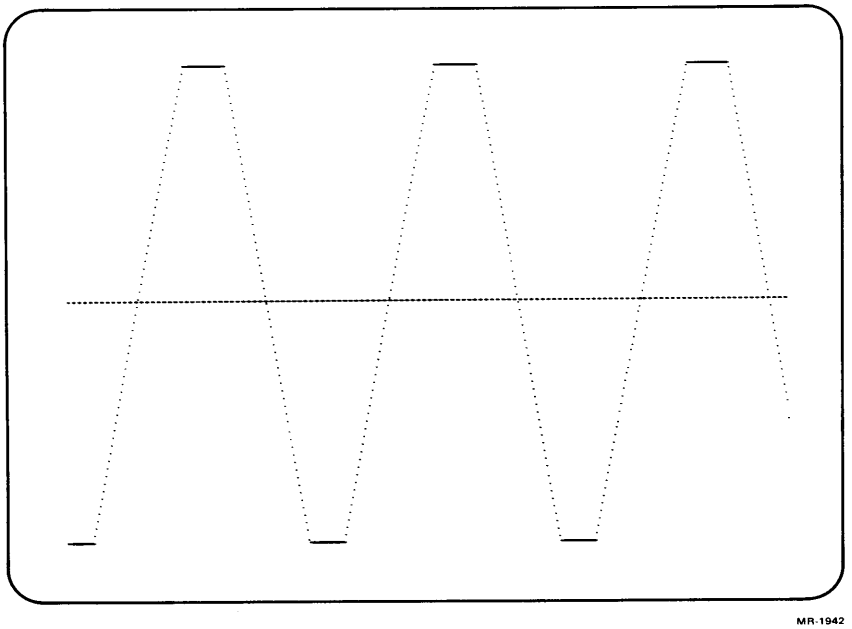


Figure 41. Display of Signal Being Converted.

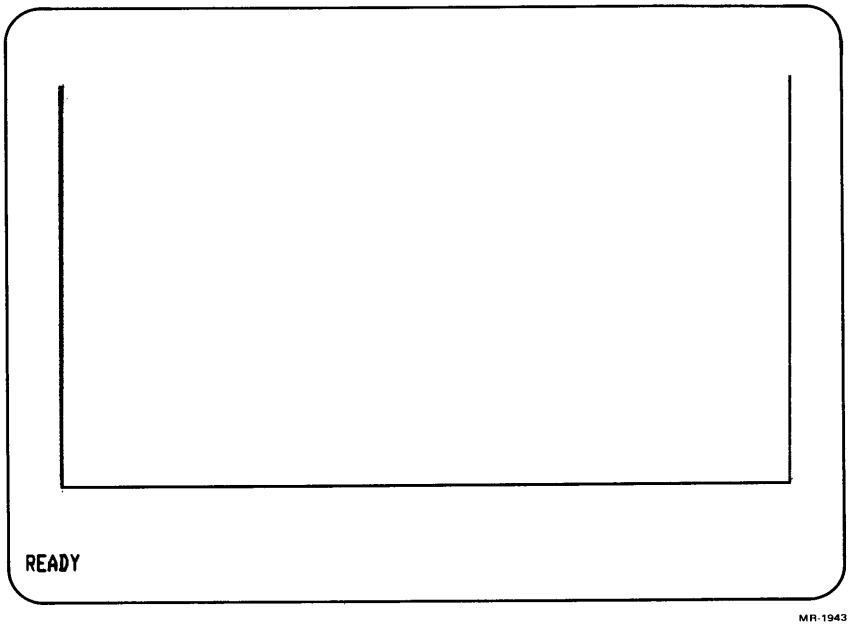


Figure 42. Histogram Generated with Too Narrow a Range of Interest.

The AIN_HIST statement specifies a histogram range of -.9 to +.9 volts. Therefore, almost all of the points being sampled lie outside of the range of interest and their occurrences increase the counts in the overflow bins. The histogram obtained from this statement looks like the one in Figure 42.

A histogram in which the range of interest corresponds to the full scale voltage looks like the one in Figure 43.

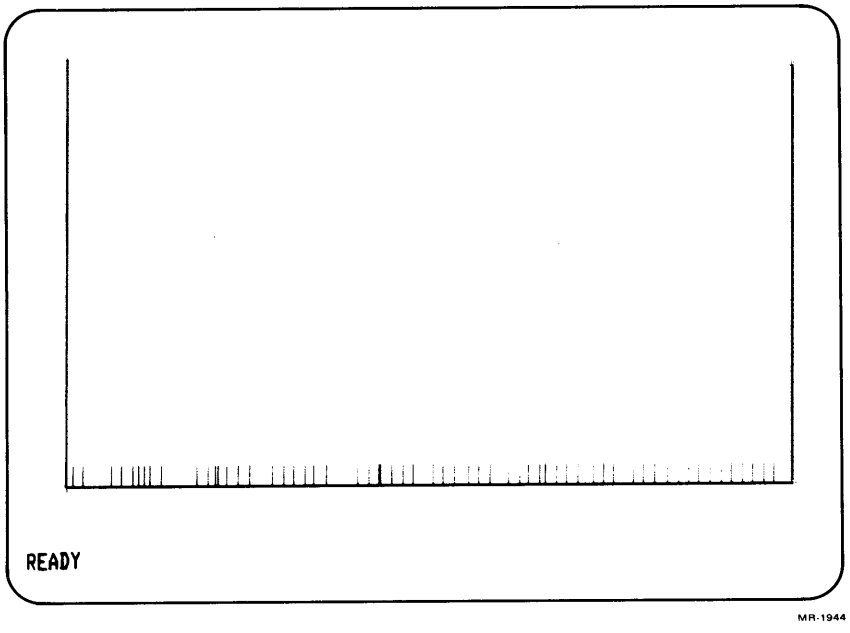


Figure 43. Histogram with Appropriate Range of Interest.

It is good programming practice to check the A/D channel with TEST_GAIN before starting the data collection sweep with AIN_HIST.

Errors

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Data lost—transfer rate too high

?MINC-F-Existing display conflicts with display requested

A prior AIN statement specified continuous display mode. AIN_HIST cannot erase this display.

?MINC-F-Histogram arrays must contain at least 4 elements

?MINC-F-Invalid or conflicting options requested

One of the mode designators is invalid for AIN_HIST.

?MINC-F-No autogain channels permitted

?MINC-F-Value of argument # exceeds valid range

The range of interest specified exceeds the maximum range for the A/D converter. The endpoints must be within the range -10.24 to +10.24 volts.

The lower-endpoint must be less than the upper-endpoint.

Examples

No example included.

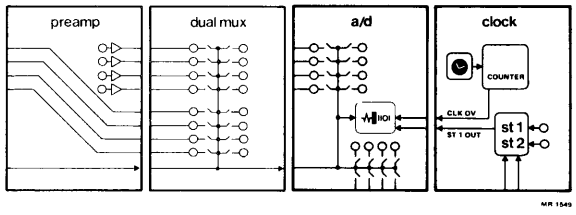
AIN_SUM

Accumulate Sums of Analog Input

Operation

AIN_SUM accumulates the data from a series of A/D sweeps. On each sweep, AIN_SUM adds the newly acquired digitized values to the previous sums. This process is commonly known as "signal-averaging," although no averaging actually occurs. AIN_SUM expects a signal on ST2 to start each new sweep. (See "Analog Signal Processing," page 1.)

Configuration



Statement Form

AIN_SUM(mode,**data-name**,sweep-length,trigger,A/D-channel,no.-of-channels,no.-of-sweeps,sweep-delay)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	DISPLAY,EXTERNAL, FAST,LINE, RANDOM,ZERO	standard mode
data-name	numeric array name	numeric range	required argument
sweep-length	numeric expression	≥1	1
trigger	numeric expression	0; > 0 to 655.35; 1 to 65,535	0
A/D-channel	numeric expression or integer array	0 to 63	0
no.-of-channels	numeric expression	1 to 64 or channel array length	1
no.-of-sweeps	numeric expression	1 to 32,767	1
sweep-delay	numeric expression	≥0 sec	0 sec

Example AIN_SUM('DISPLAY',V(),50,0.1,1,,4)

Result AIN_SUM collects 4 sweeps of data, each sweep 50 points long, from channel 1. On each sweep, AIN_SUM adds the data points to the appropriate elements of the array V. The array V is continuously redisplayed on the screen. AIN_SUM collects one point every 0.1 seconds (for a total sampling duration of 5 seconds).

Example AIN_SUM('EXTERNAL',D(),N,10,C(2),N1,N2)

Result	In external mode, AIN_SUM collects N2 sweeps of data, using the sequential conversion sequence starting with channel C(2). AIN_SUM adds the data points to the appropriate elements of array D. AIN_SUM collects one conversion sequence on every tenth ST1 signal.
Example	AIN_SUM('EXTERNAL,RANDOM',D(),N, 100,C%(2),N1,N2)
Result	In external mode, AIN_SUM collects N2 sweeps of data, using the conversion sequence defined by N1 channel numbers from array C%, starting with C%(2). AIN_SUM collects one conversion sequence on every hundredth ST1 signal.

Argument Descriptions

mode The character string selecting optional modes for AIN_SUM. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
DISPLAY	Monitor the data collected with a screen display.
EXTERNAL	Enable an external source connected to ST1 to provide the time base.
FAST	Use fast mode. No autogain. Collects bipolar integer data arrays only.
LINE	Line mode. Use line-frequency time base (50 or 60 Hz). (See "Time Base," page 43.)
RANDOM	Use random mode. (See "Random Channels," page 42.)
ZERO	Zero all elements in the data array before starting.
Default:	Standard mode (accumulates values from the specified number of sweeps, with either instrument trigger control of sampling, trigger = 0, or clock module control of sampling, trigger > 0 to 655.35).

Some combinations of modes are valid and some are not. In the following table, invalid mode combinations are marked with an 'X.'

	DISPLAY	EXTERNAL	FAST	LINE	RANDOM	ZERO
DISPLAY	X		X			
EXTERNAL		X		X		
FAST	X		X		X	
LINE		X		X		
RANDOM			X		X	
ZERO						X

MR-1924

(mode = DISPLAY) In display mode, AIN_SUM displays the contents of the first 512 elements of the data array on the screen. If the sweep length is less than 512 points, you see a short display with a full-length horizontal axis. AIN_SUM clears the screen before beginning the display.

After each display, AIN_SUM calculates a scale factor approximately 25% greater than the maximum for that display and uses the scale factor to make the next display fill the screen. (Note: the values in the data array itself are not scaled.)

The data are continuously redisplayed. Values that exceed the maximum scaled value do not appear in the display. The maximum display rate possible is lower than the maximum data collection rate. If data are being acquired faster than the screen is being updated, the data being acquired are displayed with the scale factor intended for the previous values.

The DISPLAY mode produces a display suitable for monitoring purposes. Use the specialized graphic routines for labelled displays and other purposes. (See Book 4.)

(mode = FAST) In fast mode, AIN_SUM collects the points within a conversion sequence faster than normal. Therefore, col-

lecting each sequence is somewhat faster and the trigger intervals can be shorter.

The actual increase depends on the values of other arguments. The fastest case is single-channel sampling with external triggering (trigger = 0). The slowest case is multi-channel sampling with clocked control (internal or external, trigger > 0).

Fast mode is further limited because the data array must be integer and the A/D channels must be set for fixed gain. AIN_SUM terminates execution if it encounters an autogain channel. (See also "Restrictions" in this section.)

data-name The numeric data array in which the data from multiple sweeps are summed.

Values: -32,768 to 32,767 for integer data-name argument.

Real number range for real data-name argument.

Default: Required argument. AIN_SUM assigns data values to the argument.

The data array can be either real or integer except in fast mode. Fast mode requires an integer array. (See also "Restrictions" in this section.)

The range of conversion values collected on each sweep depends on the data type of the data-name array. If the data-name array is integer, the range of values possible on each sweep is -2048 to +2047. If the data-name array is real, the range of values possible on each sweep depends on the gain and mode selected for each channel. For example, without a preamp, the range of values possible for each channel is -5.12 to +5.1175 volts.

sweep-length The number of data points in each sweep.

Values: 1 to the maximum length of the data-name array.

Default: 1

The sweep length need not be an even multiple of the number of channels in the conversion sequence. If it is not, AIN_SUM stops the sweep after the specified sweep-length regardless of whether the conversion sequence is complete.

trigger The code which specifies how to control data transfer. (See also "Time Base," page 113.)

Values	*Meaning*
0	External signal control (signal line connected either to the external start terminal of the A/D converter or to ST1)
> 0 to 655.35	Clock module control (without mode designators for external control)
1 to 65,535	External time base (with EXTERNAL or LINE designators)
Default:	0 (external signal control)

The beginning of each sweep is triggered by a signal on ST2.

(trigger = 0) When the trigger argument is zero, A/D conversion sequences are triggered by signals connected to the external start line of the A/D converter or to ST1 of the clock module.

(trigger = >0 to 655.35) When the trigger argument is greater than zero and none of the mode designators specifies an alternate time base, the internal clock control triggers each conversion sequence within the sweep. A signal on ST2 starts each sweep.

In this case, the trigger argument specifies the *trigger interval* in seconds. The trigger interval is the time from one trigger event to the next.

The longest trigger interval possible is 655.35 seconds. The shortest trigger interval possible depends on the operating mode, the length of the conversion sequence, and on the values of other arguments.

(trigger = 1 to 65,535) In external mode, an external time base connected to ST1 provides the trigger events for AIN_SUM. In line mode, the line frequency mode of the clock module provides the trigger events for AIN_SUM.

The time base can have a regular rate (for example, line mode) or a varying rate. In either case, the trigger argument specifies the number of external signals required before a conversion sequence occurs. The maximum number possible is 65,535. If the trigger argument is 1, every external signal triggers a conversion sequence. If the trigger argument is 10, every tenth external signal triggers a conversion sequence.

A/D-channel The first channel in the conversion sequence. (See also the next paragraph on no.-of-channels.)

Values: 0 to the highest-numbered channel installed in the system.

Default: 0

The valid channel numbers depend on which analog modules are installed in the system, and on how the instruments are connected to the A/D-channels (see Book 7). The program halts with an error if you specify a channel which is not present.

The channel number can be either a numeric expression or an integer array name. If it is a numeric expression, the conversion sequence consists of that channel and as many higher-numbered channels as are specified by the no.-of-channels argument. If it is an integer array in random mode, the conversion sequence consists of as many channels from the array as specified by the no.-of-channels argument.

AIN_SUM reuses the conversion sequence defined by A/D-channel and no.-of-channels until it has collected all the points in the sweep.

no.-of-channels The number of channels defining a conversion sequence for AIN_SUM.

Values: 1 to the maximum number of channels on the system (for sequential channels)

1 to the length of the channel array (in random mode)

Default: 1

The no.-of-channels argument defines the number of points in a conversion sequence. AIN_SUM collects that number of points each time a trigger event occurs. AIN_SUM reuses the channel sequence defined by A/D-channel and no.-of-channels until it has collected all the points in the sweep.

If the A/D-channel argument is a numeric expression, the conversion sequence is defined by that channel and the number of sequential channels defined by no.-of-channels. For example, if the channel number is 16, and the number of channels is 8, then the conversion sequence is channels 16 through 23.

If the A/D-channel argument is an integer array and the state-

ment specifies random mode, then the contents of the channel array define the conversion sequence. For example, suppose the channel array contains the elements 21, 1, 13, 2, and 9, and the number of channels requested is 3. If the A/D-channel argument is the second element of the array, the conversion sequence is channels 1, 13, and 2. See also "Random Channels," page 42.

no.-of-sweeps The number of data sweeps performed.

Values: 1 to 32,767

Default: 1

sweep-delay The time interval delay of the beginning of the sweep (after the start signal on ST2).

Values	*Meaning*
0	No delay. Sweep starts on ST2 signal as usual.
> 0	Delay interval in seconds (between ST2 and first conversion).

Default: 0

The first point in the sweep is delayed after the occurrence of the ST2 signal. The sweep-delay argument defines the interval between ST2 and the first point in the first conversion sequence. It affects only when the sweep begins, not any of the time intervals or trigger events within the sweep.

For timing the delay, AIN_SUM uses the clock module. Therefore, the timing resolution is high. However, with very short delays, the variability of the interval increases. For example, delays of 100 microseconds are less likely to be accurate.

Related routines

TEST_GAIN, SET_GAIN TEST_GAIN and SET_GAIN test and set the condition of the preamp connected to any A/D channel. AIN_SUM does not permit any autogain channels, so these routines can be used before the AIN_SUM statement to prevent errors.

Restrictions

CONTINUE If the array partition length is not an even multiple of the conversion sequence length, then the data points in the conversion sequence cross the array partition boundaries. AIN_SUM needs longer to acquire those conversion sequences which cross partition boundaries than those conversion sequences which do not.

DISPLAY Display mode can produce visually peculiar results. AIN_SUM displays the first 512 physical array elements, regardless of whether the array has one or two dimensions. (See discussion of the DIM statement in Books 2 and 3.) The display shows the data in the order in which they were collected, regardless of the channel numbers from which they were collected. For a small number of channels, with distinct values, the display appears reasonable. However, with many channels or similar values, it may require a practised eye to make any sense of these displays.

FAST In some cases, fast mode attains its higher-than-normal speeds because AIN_SUM can use all of the system's computing resources. In these cases, all of the system time-keeping functions are suspended until AIN_SUM is finished. This means that the system clock loses time if AIN_SUM runs often or for long in fast mode.

During a fast mode transfer, entering a single CTRL/C stops program execution.

Gain AIN_SUM does not permit any autogain channels.

Range of values With high data values and many sweeps, it is possible that the sums in integer array elements could exceed the range for integers (-32,768 or +32,767). AIN_SUM "locks" these array elements at the limit values so that further sweeps do not cause errors. You can determine whether any array elements overflowed by inspecting a display of the final array. Values at either limit (-32,768 or 32,767) probably indicate overflow. Overflow is not a problem with real arrays because in practice the values never reach the limits for real numbers.

Sampling rates The arguments for AIN_SUM allow you to specify the trigger intervals, that is, the interval from the beginning of one conversion sequence to the beginning of the next. However, you have no direct control over the sampling rate within a conversion sequence. (The rate of conversion within a conversion sequence is always higher than the rate you could obtain with the shortest trigger interval possible for single-channel sampling.)

The conversions within a sequence always occur as fast as possible given the length of the conversion sequence, the data type of the data-name argument, and the operating modes specified. The CONTINUOUS, RANDOM, and DISPLAY mode designa-

tors all slow the conversion rate possible within a conversion sequence and their effects are additive.

Given the number of variables involved, you cannot predict easily whether or not any particular sampling rate would execute without error. The feasibility of particular desired rates must be determined empirically.

Errors

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Data lost—transfer rate too high

The time required for a conversion sequence is longer than the trigger interval.

?MINC-F-Data-name array is shorter than sweep length requested

The number of elements available for data must be greater than or equal to the length of the sweep.

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for AIN_SUM.

The AIN_SUM statement specifies incompatible mode designators. Refer to the table of valid mode designator combinations.

One of the mode designators is incompatible with one of the other arguments.

?MINC-F-No autogain channels permitted

?MINC-F-No. of channels exceeds length of integer channel array

The number of elements in the channel array must be greater than or equal to the no.-of-channels argument.

Examples

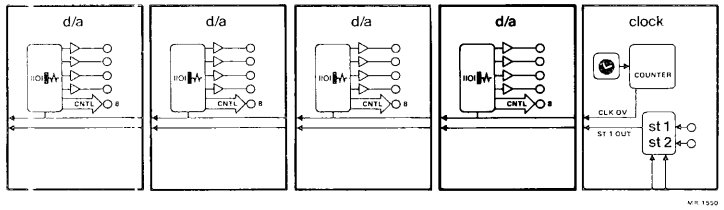
See Example B, page 58.

AOUT
Send Analog Output

Operation

AOUT sends data to analog instruments using the D/A converter. AOUT can send a single data point, a sweep of data, or a continuous stream of data. (See "Analog Signal Processing," page 1.)

Configuration



Statement Form

AOUT(mode,**data-name**,data-length,trigger,D/A-channel,no.-of-channels)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	CONTINUOUS, EXTERNAL,LINE, RANDOM,ST2	standard mode
data-name	numeric expression or numeric array	-2048 to 2047	required argument
data-length	numeric expression	⩾1	1
trigger	numeric expression	0; > 0 to 655.35; 1 to 65,535	0
D/A-channel	numeric expression or integer array	0 to 15	0
no.-of-channels	numeric expression	1 to 16 channel array length	1

Example AOUT(,D,1,0,2,1)
Result Send the contents of variable D to D/A channel 2 immediately.

Example AOUT(,300)
Result Send the value 300 to channel 0 immediately.

Example AOUT(,V(),512,,,2)
Result Send the contents of array V in pairs to channels 0 and 1. Send the conversion sequences as burst output.

Argument Descriptions

mode The character string selecting optional modes for AOUT. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
CONTINUOUS	Continuous mode. (See "Continuous Data Transfers," page 33.)
EXTERNAL	Enable an external source connected to ST1 to provide the time base.
LINE	Use line-frequency time base. (See "Time Base," page 43.)
RANDOM	Random mode. (See "Random Channels," page 42.)
ST2	Start data output process with a signal on ST2.
Default:	Standard mode (point or sweep transfer with either burst output or clock module control of output, trigger > 0 to 655.35).

Some combinations of modes are valid, and some are not. In the following table, invalid mode pairs are marked with an "X."

	CONTINUOUS	EXTERNAL	LINE	RANDOM	ST2
CONTINUOUS	X				
EXTERNAL		X	X		
LINE		X	X		
RANDOM				X	
ST2					X

MR-1923

data-name The values to be sent to the instrument connected to the D/A module.

Values: -2048 to 2047

Default: Required argument

If the data-name argument is a single value, AOUT sends only

the single value to the D/A converter. If the data name is an array name, AOUT sends either a sweep of output values, or, in continuous mode, a continuous stream of output values.

The values can be either integer or real. In either case, the range of output values allowed is -2048 to +2047.

data-length The length of the data-name argument.

Values	*Meaning*
1	The data-name argument is a single value.
> 1	The data-name argument is an array name and the data-length argument is the number of array elements to output.
Default:	1

If the data-length argument is greater than 1, the data-name argument must be an array name. The data-length argument need not be an even multiple of the number of channels in a conversion sequence. In standard mode, AOUT stops sending data after the specified number of points regardless of whether the conversion sequence is complete. In continuous mode, a conversion sequence can straddle an array partition boundary.

trigger The code specifying how to control data transfer.

Values	*Meaning*
0	Burst output. The conversion sequences are sent to the D/A converter as fast as possible.
>0 to 655.35	Clock module control (when there are no mode designators for external control).
1 to 65,535	External time base (with EXTERNAL or LINE mode designators).
Default:	0

(trigger = 0) When the trigger argument is zero, AOUT sends the conversion sequences to the D/A converter as fast as possible. This is called *burst output*. Burst output transfers are not controlled by trigger events.

Burst output is not allowed in continuous mode.

(trigger = > 0 to 655.35) When the trigger argument is greater

than zero and none of the mode designators specifies an alternative time base, the clock module triggers the analog output conversion sequences. In this case, the trigger argument specifies the *trigger interval* in seconds. The trigger interval is the time from one trigger event to the next, that is, the beginning of one conversion sequence to the beginning of the next.

The longest trigger interval possible is 655.35 seconds. The shortest trigger interval possible depends on the operating mode, the length of the conversion sequence, and the values of the other arguments. For example, the minimum trigger interval for single-channel conversion sequences is shorter than for multi-channel conversion sequences.

(trigger = 1 to 65,535) In external mode, an external time base connected to ST1 provides the trigger events for AOUT. In line mode, line frequency mode of the clock module provides the trigger events for AOUT.

The time base can have a regular rate (for example, line mode) or a varying rate. In either case, the trigger argument specifies the number of external signals required before a conversion sequence occurs. The maximum number possible is 65,535. If the trigger argument is 1, every external signal triggers a conversion sequence. If the trigger argument is 20, every twentieth signal triggers a conversion sequence.

D/A-channel The D/A channel or channels to receive output data.

Values: 0 through 15

Default: 0

The D/A-channel argument defines the channels in a conversion sequence. AOUT reuses the conversion sequence defined by D/A channel and no.-of-channels until it has sent all of the output values.

The channel number can be either a numeric expression or an integer array name. If it is a numeric expression, the conversion sequence consists of that channel and as many higher-numbered channels as are specified by the no.-of-channels argument. If it is an integer array in random mode, the conversion sequence consists of the channel numbers in the array. (See also the next paragraph.)

no.-of-channels The number of D/A channels defining a conversion sequence for AOUT.

Values: 1

1 to 16 (for sequential channels)

1 to the length of the channel array (in random mode)

Default: 1

The no.-of-channels argument defines the number of points in a conversion sequence. AOUT sends that many points each time a trigger event occurs. AOUT reuses the conversion sequence defined by D/A-channel and no.-of-channels until it has sent all of the output values.

If the D/A-channel argument is a numeric expression, the conversion sequence consists of that channel and as many sequential channels as are specified by the no.-of-channels argument. For example, if the channel number is 1 and the number of channels is 2, then the conversion sequence is channels 1 and 2.

If the D/A-channel argument is an integer array in random mode, the conversion sequence consists of the channels in the array. For example, suppose the channel array contains the values 2, 1, 0, 3, 2, 1, the D/A-channel argument is the first element in the array, and the no.-of-channels argument is 6. Then, the conversion sequence is channels 2, 1, 0, 3, 2, and 1. If the number requested is 4, the conversion sequence is channels 2, 1, 0, and 3. (See "Random Channels," page 42.)

Related Routines

CONTINUE CONTINUE allows the program to resume executing during a continuous mode transfer and to continue executing until the currrent array partition has been transferred. Whenever an array partition becomes empty, CONTINUE transfers control to the service subroutine which refills that partition with data, while AOUT continues to output from the other partition. On returning from the subroutine, the program continues executing the statements following the CONTINUE statement. (See "Continuous Data Transfers," page 33, and CONTINUE.)

SET_BIT SET_BIT specifies the condition (set or clear) for a single bit in a numeric variable. You can use SET_BIT to specify the control signal values.

TERMINATE TERMINATE terminates continuous mode data output.

WAIT_FOR_DATA WAIT_FOR_DATA stops the program during a continuous mode transfer and the program waits for the current array partition to be output. When the array partition is empty, the program resumes executing to fill the empty array partition while AOUT continues to send values from the other one.

CONTINUE Only one continuous AOUT transfer can be in progress.

Restrictions

If the array partition length is not an even multiple of the conversion sequence length, then the data points in the conversion sequence cross the array partition boundaries. AOUT needs longer to send those conversion sequences which cross partition boundaries than those conversion sequences which do not.

Control signals Four instrument control signals are available from the D/A converter for plotter or scope control. You can view the D/A converter as corresponding to a word composed of 16 bits. (See "Bits and Words," page 26.) For D/A channels 3, 7, 11, and 15, you can connect instrument control lines to the terminals labelled Bits 0 through 3 on the D/A converter connector block. (See Book 7.)

Suggested conventions follow for using the instrument control bits.

Bit 0 — The intensify line.
For channel 3, AOUT automatically sets and then clears Bit 0 about 80 μsec after each complete conversion sequence. The duration of the Bit 0 pulse is about 10 μsec. (AOUT does not supply Bit 0 automatically for channels 7, 11, and 15.)

You can use Bit 0 to supply the intensify strobe signal required by an oscilloscope. You can connect the intensify line of an oscilloscope to one of the Bit 0 terminals on the D/A converter connector block (see Book 7). The output sequence can be two channels (the X and Y values for the scope) or any number of channels required.

Bit 1 — Suggested as the erase line.

You can use Bit 1 to erase an oscilloscope display by setting Bit 1 in the data value going to channel 3 (see SET_BIT). Connect the oscilloscope erase line to one of the Bit 1 terminals on the D/A converter connector block (see Book 7).

Bit 2 — No suggested function. Connect the instrument to one of the Bit 2 terminals on the D/A converter connector block (see Book 7). Set Bit 2 in the data value going to channel 3 (see SET_BIT).

Bit 3 — Suggested as the pen position line for an analog plotter. The use of the control bit depends on the plotter. For some plotters, each transition on the line changes the up/down position of the pen; for others, one logic level represents pen up and the other represents pen down.

Connect the plotter pen position line to one of the Bit 3 terminals on the D/A converter connector block. Set Bit 3 in the data value going to channel 3 (see SET_BIT).

Immediate mode Point and sweep AOUT transfers operate in immediate mode; continuous transfers do not.

Range of values All output data sent to the D/A converters must have values in the range -2048 to +2047. The D/A converter front panel has controls for output range and output mode, unipolar or bipolar (see Book 7). Set range and mode to the desired values with the front panel controls; AOUT cannot set range and mode.

Errors

?MINC-F-Another transfer is in progress for the array specified

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Clock too fast for system to respond

The trigger interval requested is too short.

?MINC-F-Data lost—transfer rate too high

The trigger interval is too short to process the array partitions in continuous mode.

?MINC-F-Data-name array is shorter than sweep length requested

The number of elements available for data must be greater than or equal to the length of the sweep.

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for AOUT.

The AOUT statement specifies incompatible mode designators. Refer to the table of valid mode designator combinations.

One of the mode designators is incompatible with one of the other arguments.

?MINC-F-No. of channels exceeds length of integer channel array

The number of elements in the channel array must be greater than or equal to the no.-of-channels argument.

?MINC-F-Value exceeds valid range for argument

Output values must be in the range -2048 to +2047.

See Example H, page 71.

Examples

CIN

Collect Character String Input

Operation

CIN receives characters transmitted in serial ASCII format via a serial transfer channel. (See "Transferring ASCII Characters," page 4.)

Configuration

The serial transfer channels are an integral part of the MINC physical system. The serial transfer unit, containing four channels, is not one of the removable MINC modules. It is permanently installed in the chassis (see Book 7).

Two of the channels are reserved, one for the terminal, and the other for an optional DECwriter printer. Make connections to the other two channels through the connectors on the MINC back panel.

Statement Form

CIN(mode,string-name,string-length,channel-no.,timeout-interval)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	RETRIEVE	standard mode
string-name	string variable name	ASCII characters	required argument
string-length	numeric expression	0; 1 to 255	0
channel-no.	numeric expression	0 or 1	0
timeout-interval	numeric expression	0; ≥0.1 sec	0

Example CIN(,S$,5)
Result Collect a string of five characters from channel 0, and assign the string to variable S$.

Example CIN(,S$)
Result Collect character input from channel 0 until a carriage return character occurs (or until 255 characters have been received). Assign the string to variable S$.

Example CIN(,S$,20,1,60)
Result Collect up to 20 characters from channel 1. Cancel the request after 60 seconds if 20 characters have not arrived, and assign whatever has arrived to variable S$.

Example CIN('RETRIEVE',S$,,,30)
Result Collect up to 255 characters from channel 0,

starting with the first one after the last CIN statement. Stop receiving characters after 255 characters, or when a carriage return character is found, or when the timeout interval of 30 seconds has elapsed.

Argument Descriptions

mode The character string selecting an optional mode for CIN. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
RETRIEVE	Retrieve any characters arriving since the last CIN statement.
Default:	Standard mode (ignore characters arriving between CIN statements).

In standard mode, the first character assigned to the string is the first character that arrives after the current CIN statement starts executing. Any intervening characters (arriving since the last CIN statement finished executing) are lost.

(mode = RETRIEVE) In retrieve mode, CIN goes back to retrieve any characters that have arrived since the most recent CIN statement finished executing. (That is, retrieve mode cannot retrieve any characters unless CIN has executed previously because if CIN has not executed there cannot be any characters to retrieve.)

The first character assigned to the string is the first character that arrived after the most recent CIN statement finished executing. The contents of the rest of the string are determined by string-length argument, just as in standard mode.

At most 128 characters can be retrieved in retrieve mode. If more than 128 characters have arrived, then CIN can retrieve up to 128 characters before the program halts with a fatal error.

In retrieve mode, CIN collects as many characters as specified by the string-length argument. For example, if the string length is 10, then CIN collects 10 characters starting with the first one after the most recent CIN statement. If more than 10 characters have already arrived, CIN still collects only 10 characters. If fewer than 10 characters have arrived, CIN collects all those characters and waits for the remaining characters necessary to satisfy the count before finishing executing.

For a variable-length string (string-length argument of 0), CIN

collects characters until it finds the first carriage return character (or until the limit of 255 characters occurs or until the timeout interval expires). If a carriage return has already arrived, that carriage return defines the end of the string and CIN completes executing. (The character following that carriage return is the first one retrieved by the next retrieve mode CIN statement.) Otherwise, it collects those characters that have arrived and continues until a carriage-return character arrives.

string-name The variable name for the data collected.

Values: ASCII characters (or any 8-bit codes).

Default: Required argument. CIN assigns the value to the argument.

CIN requires the string-name argument to be a string variable, for example, B$, or string array element, for example, B$(0). You cannot collect an array of strings using one CIN statement.

string-length The length of the character string to be collected.

Values	*Meaning*
0	Collect a variable-length string.
1 to 255	Collect a fixed-length string.
Default:	0 (variable-length string)

(variable-length strings) With variable-length strings, CIN collects characters until a carriage-return character arrives, or until the maximum string length of 255 characters has been reached. The carriage-return character is not included in the string.

(fixed-length strings) If the string-length argument is a number from 1 to 255, CIN collects that number of characters. There are no terminator characters for fixed-length strings. CIN can accept a carriage-return character as one of the characters in a fixed-length string. CIN continues executing until all of the characters specified have arrived. (Thus, the timeout argument can be useful to ensure that CIN eventually stops executing in case not enough characters arrive.)

channel-no. The serial channel carrying input characters.

Values	*Meaning*
0	9600 baud serial channel
1	1200 baud serial channel
Default:	0

The baud rates for the serial channels are fixed. They are not programmable. Under normal warranty conditions, only Digital Field Service can alter the baud rate.

Choose the channel with the baud rate you require. If the baud rate for the channel does not match the baud rate of the instrument connected to it, the characters received by CIN are not the same as the ones sent by the instrument. CIN halts the program with an error.

timeout-interval The time interval after which CIN considers the current string to be complete.

Values	*Meaning*
0	No timeout
$\geqslant 0.1$	Timeout interval in seconds
Default:	0 (No timeout)

CIN begins timing the timeout interval when the CIN statement starts executing. If the timeout interval elapses before the specified number of characters arrives, or before a carriage-return terminator arrives, CIN considers the current string to be complete. The transfer is therefore complete, and the program continues executing with the statement following the CIN statement.

The shortest time interval guaranteed to be precise is 0.1 seconds. The resolution on the timeout interval is one tick of the system clock.

Related Routines

COUT COUT sends character strings to an instrument connected to a serial channel. COUT can send fixed-length or variable-length character strings.

Restrictions

RETRIEVE Unlike other kinds of transfers, character input can continue for up to 128 more characters after the CIN statement finishes executing. In standard mode, CIN ignores any intervening characters, and starts the string with the next charac-

ter that arrives. In retrieve mode, CIN can retrieve characters that have arrived since the most recent CIN statement. Therefore, you can use CIN in retrieve mode to handle essentially continuous character input because you can process each string as it arrives without losing intervening characters.

The 128-character maximum imposes a practical limit on the character input rate. At 9600 baud, 128 characters can arrive in approximately 135 milliseconds. At 1200 baud, 128 characters can arrive in approximately one second.

Response CIN does not echo characters. That is, it does not automatically transmit the received character back to the sender for display or verification.

Errors

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Data lost—transfer rate too high

More than 128 characters have arrived since the last CIN statement finished executing (retrieve mode).

The characters are arriving too quickly for CIN to receive them properly.

Characters are arriving at the wrong baud rate.

?MINC-F-Invalid or conflicting options requested

RETRIEVE is the only valid mode designator.

?MINC-F-No workspace available for the string specified

?MINC-F-Use array element instead of array for argument #

Examples

No example included.

CONTINUE

Manage Continuous Data Transfer

Operation

CONTINUE designates a service subroutine for continuous input or output. The program continues executing until the data transfer routine has processed one array partition. At that point, program control transfers to the service subroutine. (See "Continuous Data Transfers," page 33.)

Configuration

The configuration depends on which data transfer routine is associated with the CONTINUE statement.

Statement Form

CONTINUE(data-name,index,subroutine)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
data-name	numeric array	transfer array	required argument
index	numeric variable name	0,(n+1)/2	required argument
subroutine	numeric expression	1 to 32767	required argument

Example CONTINUE(V(),I1%,1000)
Result Designate the service subroutine beginning at statement number 1000 to execute when one partition of array V has been transferred. The array index for the current partition is assigned to variable I1%.

Example CONTINUE(D%(),I,L)
Result Designate the service subroutine (whose beginning statement number is in variable L) to execute when one partition of array D% has been transferred. The array index for the current partition is assigned to variable I.

Argument Descriptions

data-name The data array being used for the continuous data transfer.

Values: The array appearing in a previous transfer statement.

Default: Required argument

CONTINUE itself allows any numeric array name. The associated data transfer routine might have some restriction on the data type allowed for the array.

The data-name array must already have appeared in a data

transfer statement specifying continuous mode. If the dataname array is not involved in a continuous transfer, CONTINUE halts with an error.

index The variable to contain the array index for the next partition to be processed by the service subroutine.

Values: 0 or (n+1)/2 (where n is the array dimension)

Default: Required argument. CONTINUE assigns the value to the index argument.

CONTINUE assigns the value of the index argument as the index of the first element in the next array partition for the program to process. If continuous output is in progress, the index points to the beginning of the array partition to be filled with output data to be transferred. If continuous input is in progress, the index points to the beginning of the array partition containing valid data to be processed.

subroutine The statement number of the service subroutine for this data transfer.

Values: 1 to 32,767 (any valid program statement number)

Default: Required argument

CONTINUE designates a service subroutine statement number. MINC transfers program control to the statement designated whenever an array partition is ready for processing. With output transfers, the service subroutine must contain the statements to fill the array partition with output data (see also "Restrictions"). For input transfers, the service subroutine must contain the statements to retrieve data from the full array partition. In both cases, the service subroutine must execute CONTINUE again to prepare for the next array partition transfer. (See "Continuous Data Transfers," page 33.)

The subroutine argument can be any numeric expression. CONTINUE truncates any fractional statement numbers. For example, if the value of the subroutine argument were 125.8, CONTINUE would transfer control to statement 125.

Related Routines

AIN, DIN The input transfer routines AIN and DIN can operate in continuous mode. MINC transfers program control to the service subroutine (designated in CONTINUE) whenever an array partition fills and is ready for processing.

AOUT, DOUT The output transfer routines AOUT and DOUT can operate in continuous mode. MINC transfers program control to the service subroutine (designated in CONTINUE) whenever an array partition requires filling with output data.

WAIT_FOR_DATA WAIT_FOR_DATA synchronizes data transfer and program execution. (See "Continuous Data Transfers," page 33, and WAIT_FOR_DATA.) WAIT_FOR_DATA waits until the next array partition is ready for processing before allowing the program to continue past the WAIT_FOR_DATA statement.

TERMINATE TERMINATE stops a specified continuous data transfer. The statements processing the array partition must test for a condition that defines the end of the data transfer.

Restrictions

Index argument The index argument must be a variable name. Do not use that variable name for any other purpose elsewhere in the program. With transfers managed by CONTINUE, erroneous results could result if the program uses the index variable for any other purpose during the data transfer.

Immediate mode CONTINUE does not operate in immediate mode.

Multiple transfers Multiple transfers with CONTINUE management can operate simultaneously. Only one transfer at a time can be controlled by the clock module and all must use different MINC devices.

Output transfers The first time the CONTINUE routine executes for an output transfer, it immediately passes control to the service subroutine. The service subroutine then prepares the first partition for transfer and executes CONTINUE again. The output transfers actually start at this point (see Figure 19).

Starting transfers The program must call CONTINUE before the data transfer routine can start transferring data. Even when the data transfer statement specifies an external start signal, the routine does not start transferring data when the signal occurs unless the CONTINUE statement has executed. Therefore, the CONTINUE statement must closely follow the data transfer statement, either physically or logically.

RESEQ The resequencing command, RESEQ, resequences normal program statement numbers. It does not resequence the

service subroutine statement number in the CONTINUE statement. When you resequence a program, you have to determine the new statement number of the service subroutine and change that number in the CONTINUE statement.

For this reason, you might find it more convenient to use variable names for subroutine arguments. Put the variable assignment statement and an explanatory remark at the beginning of the program where you can locate it quickly. Then, each time you resequence the program, assign new values to the statement number variables, and save searching for all occurrences of the CONTINUE statement.

Errors

?MINC-F-Continuous transfer not in progress for array specified

?MINC-F-Could not find service subroutine ##### requested

?MINC-F-Service subroutine request pending. Cannot use CONTINUE

The array partition transfer being managed by the last CONTINUE statement is still in progress. You cannot execute CONTINUE again until the program has entered the service subroutine requested by the previous CONTINUE statement.

?MINC-F-Subroutine ####; Clock too fast for system to respond

The transfer associated with the subroutine #### requested trigger intervals that are too short.

?MINC-F-Subroutine ####; Data lost—transfer rate too high

The transfer associated with the subroutine #### has failed.

?MINC-F-Subroutine ####; Value exceeds valid range for argument

The transfer associated with the subroutine #### tried to transfer an invalid data value.

Examples

See Example C, page 61.

COUT
Send Character String Output

Operation

COUT sends characters in serial ASCII format via a serial transfer channel. (See "Transferring ASCII characters," page 4.)

Configuration

The serial transfer channels are an integral part of the MINC physical system. The serial transfer unit, containing four channels, is not one of the removable MINC modules. It is permanently installed in the chassis.

Two of the channels are reserved, one for the terminal, and the other for an optional DECwriter printer. Make connections to the other two channels through the connectors on the MINC back panel (see Book 7).

Statement Form

COUT(mode,string-name,string-length,channel-no.)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	WAIT	standard mode
string-name	string expression	ASCII characters	required argument
string-length	numeric expression	0; 1 to 255	0
channel-no.	numeric expression	0 or 1	0

Example COUT(,S$,5)
Result Send a string of five characters to channel 0, and start executing the next statement without waiting for the transfer to be complete.

Example COUT(,S$)
Result Send the whole string S$ to channel 0, adding a carriage-return character to the end of the string. Start executing the next statement without waiting for the transfer to be complete.

Example COUT(,S$(2),10)
Result Send a string of 10 characters to channel 0 (without appending a carriage-return character) and start executing the next statement without waiting for the transfer to complete.

Example COUT('WAIT','Attention',,1)
Result Send the whole string *Attention* to channel 1, adding a carriage-return character to the end

of the string. Start executing the next statement only after the transfer is complete.

Argument Descriptions

mode The character string selecting an optional mode for COUT. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
WAIT	Wait until the transfer is complete before executing the next statement.
Default:	Standard mode (start the transfer and pass control to the next statement when at most 32 characters remain to be transferred).

In standard mode, COUT sets up and starts the transfer and then finishes executing. The transfer then completes by itself while the program continues.

(mode = WAIT) In wait mode, the COUT statement does not finish executing until the transfer is complete, that is, until the final character has been sent over the serial channel.

string-name The variable name for the data sent.

Values:	String of ASCII characters (or any 8-bit code).
Default:	Required argument

COUT requires the string-name argument to be a string expression. You cannot send a complete string array using one COUT statement.

string-length The number of characters to send.

Values	*Meaning*
0	Send as many characters as the string contains (variable-length string) with a carriage return character appended.
1 to 255	Send a fixed number of characters from the string (fixed-length string) without any carriage return character appended.
Default:	0 (variable-length string)

(variable-length string) With variable-length strings, COUT

sends as many characters as the string contains (up to the string maximum of 255 characters). COUT appends a carriage-return character to the string.

(fixed-length string) If the string-length argument is a number from 1 to 255, COUT sends that number of characters beginning with the first one in the string. If the string does not contain enough characters, COUT sends only as many as are in the string. COUT does not append a carriage-return character to a fixed-length string.

channel-no. The serial channel carrying output characters.

Values	*Meaning*
0	9600 baud serial channel
1	1200 baud serial channel
Default:	0

The baud rates for the serial channels are fixed. They are not programmable. Under normal warranty conditions, only Digital Field Service can alter the baud rate.

Choose the channel with the baud rate you require. If the baud rate for the channel does not match the baud rate of the instrument connected to it, the instrument receives characters that are different from the ones sent by COUT. (This causes errors at the receiver end; COUT cannot detect this situation.)

Related Routines

CIN CIN collects character strings from an instrument connected to a serial channel. CIN can receive fixed-length or variable-length character strings.

Restrictions

WAIT In wait mode, COUT operates with normal program dynamics. That is, the COUT statement finishes executing when the character transfer is complete. In standard mode, COUT starts the transfer and then finishes executing before all the characters arrive at the receiver. COUT in standard mode can finish executing when up to 32 characters are awaiting transfer. Another standard mode COUT statement can start executing before the transfer completes. However, it cannot finish executing until at most 32 characters (total) remain to be transferred.

Errors

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Invalid or conflicting options requested

WAIT is the only valid mode designator.

?MINC-F-Use array element instead of array for argument #

Examples

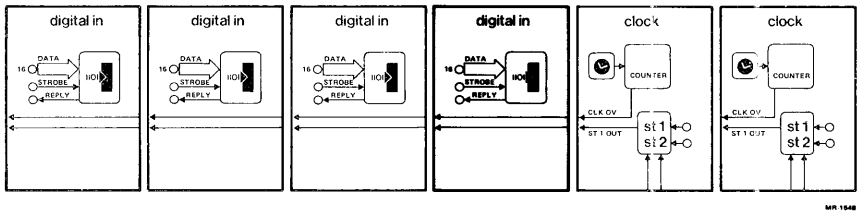
No example included.

DIN
Collect Digital Input

Operation

DIN collects data from the digital input unit specified. DIN can collect a single value, a sweep of data, or a continuous stream of data. (See "Digital Sampling and Control," page 3.)

Configuration



Statement Form

DIN(mode,**data-name**,data-length,trigger,unit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	CONTINUOUS, EXTERNAL LINE,ST2,TIMESTAMP	standard mode
data-name	numeric variable or array name	-32,768 to 32,767	required argument
data-length	numeric expression	⩾1	1
trigger	numeric expression	0; 0 to 655.35 1 to 65,535	0
unit	numeric expression	0 to 3	0

Example DIN(,V,1,0,2)
Result DIN reads the value of digital input unit 2 as soon as the DIN statement executes, and puts the single value in variable V.

Example DIN('ST2',A(),50,1)
Result DIN starts collecting a sweep of 50 values from digital input unit 0 when a signal occurs on ST2. It puts one point into array A every second.

Argument Descriptions

mode The character string selecting optional modes for DIN. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
CONTINUOUS	Continuous mode sampling. (See "Continuous Data Transfers," page 33.)

EXTERNAL	Enable an external source connected to ST1 to provide the time base. (See "Time Base," page 43.)
LINE	Use line-frequency time base (50 or 60 Hz). (See "Time Base," page 43.)
ST2	Start data collection process with a signal on ST2.
TIMESTAMP	Timestamp the input values.
Default:	Standard mode (point or sweep transfer with either instrument triggering, independent event control, trigger = 0, or clock module control, trigger > 0 to 655.35).

Some combinations of modes are valid, and some are not. In the following table, "X" marks invalid mode pairs and "?" marks mode pairs whose validity depends on the configuration. The mode pairs marked with "?" are valid if the system contains two clock modules and invalid otherwise.

	CONTINUOUS	EXTERNAL	LINE	ST2	TIMESTAMP
CONTINUOUS	X				
EXTERNAL		X	X		?
LINE		X	X		?
ST2				X	
TIMESTAMP		?	?		X

MR 1922

(mode = TIMESTAMP) In timestamp mode, DIN records both the time when a value was read from the digital input unit (the time stamp) and the value itself. Timestamp mode is intended for data collection in which the data collection is controlled by an external time base (for example, external mode) or strobe signals (trigger = 0). (Note: In external mode, timestamping requires two clocks.)

Timestamp mode uses the elapsed-time counter. The elapsed-time counter must already be running at the required rate. (Start the elapsed-time counter with START_TIME.)

DIN uses a nonstandard format to record the timestamp values. Convert timestamp values to standard numeric format using MAKE_TIME. (See "Number Systems," page 25.)

In timestamp mode, DIN collects two values for every data point. Therefore, the input data name must always be an array name. DIN stores the value pairs sequentially in the array as they are collected, the timestamp value first, then the digital input value.

The length of the array must always be twice the number of data points that the array will hold. That is, if you want to collect 50 digital input values, the data array length must be 100—50 of the array elements contain timestamp values, and 50 contain digital input values.

data-name The numeric variable to contain the data collected by DIN.

Values: -32,768 to 32,767

Default: Required argument. DIN assigns the value(s) to the data-name argument.

In timestamp mode, the data array must be an integer array.

The data-name values represent the conditions of the digital input lines when the trigger signal occurs. If you use these values as standard numeric values, you find that they have values in the integer range (because MINC interprets these 16-bit values as if they were integers). Normally, you need to interpret the data-name values either as BCD format values (using MAKE_NUMBER) or as individual line conditions (using SCAN_BIT or TEST_BIT). (See "Bits and Lines," page 27.)

data-length The amount of workspace allocated for digital input data.

Values	*Meaning*
1	The data-name argument is a numeric variable or single array element.
> 1	The data-name argument is an array name and

the data-length argument is the number of array elements to be filled with data.

Default: 1

If the data-length argument is greater than 1, the data-name argument must be an array name.

If timestamp mode is specified, data-length must be twice the number of digital input values required, in order to accommodate the timestamp readings in the same array.

trigger The code which specifies how to control the data transfer.

Values	*Meaning*
0	Collect the digital input reading immediately if data length is 1.
	Collect a digital input reading on every signal from the instrument to the strobe terminal on the connector block for the digital input unit.
	Collect a digital input reading whenever a signal occurs on an enabled line (for lines enabled with DIN_EVENT).
> 0 to 655.35	Internal clock control of sampling (without mode designators for external control)
1 to 65,535	External control of sampling (with EXTERNAL or LINE designators)
Default:	0

(trigger = 0) When the trigger argument is zero, individual digital input samples are triggered by signals from the instrument. The signal line is connected to the strobe terminal on the digital input unit connector block or to any of terminals D00 through D15 (for lines enabled with DIN_EVENT) (see Book 7).

(trigger > 0 to 655.35) When the trigger argument is greater than zero and none of the mode designators specifies an alternative time base, the rate of digital input is under internal clock control. DIN chooses the appropriate clock module frequency (see "Time Base," page 43.)

The trigger argument specifies the sample period in seconds. The sample period is the time from reading one input value to reading the next.

The longest sample period possible is 655.35 seconds. The shortest sample period possible depends on the operating mode and the values of other arguments.

The sampling frequency is the inverse of the sample period (trigger argument). For example, if the trigger argument is .015, the sampling frequency is 66 Hz; if the trigger argument is .002, the sampling frequency is 500 Hz. If the sample period is 5, DIN reads the unit every 5 seconds (0.2 Hz).

(trigger = 1 to 65,535) In external mode, DIN collects data under the control of an external time base connected to ST1. In line mode, DIN collects data using the line frequency mode of the clock module as the time base. The time base can have a regular rate (for example, line mode) or a varying rate. In either case, the trigger argument specifies the number of input signals required for DIN to read the digital input unit. The maximum number possible is 65,535. If the trigger argument is 1, DIN reads the digital input unit on every input signal. If the trigger argument is 5, DIN reads the digital input unit on every fifth input signal.

If the input signals have a regular rate, the trigger argument acts as a rate divisor (or, equivalently, as a period multiplier). For example, if input signals are coming from a 60 Hz line frequency clock, then a trigger argument of 6 specifies a rate of 60/6 or 10 Hz, and a period of (1/60)*6 or 0.1 seconds (one sample every 0.1 seconds).

unit The number of the digital input unit carrying input data.

Values: 0 through 3

Default: 0

The digital input units are independent. Use multiple DIN statements to sample from multiple digital input units. (See also Restrictions in this section.) Note: Each DIN statement must contain a different data name. Multiple digital input units cannot collect data into the same array at the same time.

Related Routines

CONTINUE CONTINUE allows the program to resume executing during a continuous mode transfer and to continue executing until the current array partition fills. Whenever an array partition fills, CONTINUE transfers control to a service

subroutine which processes the full array partition while the other one continues to fill. (See "Continuous Data Transfers," page 33, and CONTINUE.)

DIN_EVENT DIN_EVENT enables individual lines in the digital input to control data collection. If the trigger argument in DIN is zero (indicating external control of sampling), then DIN can read the contents of the whole digital input unit into the data array each time the condition of an enabled line changes. (Note: The DIN_EVENT routine itself does *not* read the digital input unit.) If DIN_EVENT has not executed, the event-enable word for each unit has the bits for all lines clear.

DIN_MASK DIN_MASK specifies a mask for a digital input unit. If a line is masked with 0, then the condition of the corresponding bit in the data input word is always 0, regardless of the actual condition of the line. If a line is masked with 1, then the condition of the corresponding bit in the data input word is the actual condition of the line (0 or 1). (See "Masking," page 27 and DIN_MASK.)

MAKE_NUMBER MAKE_NUMBER converts BCD format data to standard numeric format. If the instrument connected to the digital input unit is sending BCD data, your program must use MAKE_NUMBER to convert the BCD data to a numeric format that MINC can use. (See MAKE_NUMBER, "Number Systems," page 25.)

MAKE_TIME MAKE_TIME converts nonstandard format timestamp values collected by DIN to standard numeric format. (See "Number Systems" page 25, and MAKE_TIME.)

START_TIME START_TIME sets the rate of the elapsed-time clock and starts the clock counting from zero. If you have not started the elapsed-time clock, timestamp mode does not work. DIN halts the program with an error.

TEST_LINE TEST_LINE reads the condition of a single specified line in a digital input unit.

TERMINATE TERMINATE terminates continuous mode data collection. (See "Continuous Data Transfers," page 33, and TERMINATE.)

WAIT_FOR_DATA WAIT_FOR_DATA stops the program during a continuous mode transfer and the program waits until

the current array partition fills. When the array partition fills, the program resumes executing to process the full array partition while the other one continues to fill. (See "Continuous Data Transfers," page 33, and WAIT_FOR_DATA.)

Restrictions

Array structure In timestamp mode, DIN reads two values on each trigger—the timestamp value and the digital input unit value. Therefore, the data-name array contains alternating time and digital values. The even-numbered array elements contain times and the odd-numbered array elements contain digital values.

You can use a two-dimensional data-name array to make the subsequent array processing easier. As Book 2 explained, MINC stores arrays in *row-major order*. That is, if you could examine the elements in a two-dimensional array sequentially, you would find them arranged in row order, one row after the other. Therefore, you can dimension the data-name array so that it contains two columns and as many rows as there are input points. Then column 0 contains all the timestamp values and column 1 contains all the digital input values.

DIN_EVENT In sampling from lines enabled by DIN_EVENT, DIN cannot produce the error message: Data lost—transfer rate too high.

Immediate mode Point and sweep DIN transfers operate in immediate mode; continuous transfers do not.

Multiple units The digital input units are independent of each other. Use one DIN statement for each unit being used. Different units must use different data arrays, and different service subroutines.

Multiple continuous mode DIN statements can all use timestamp mode. All of the DIN statements use the same elapsed-time counter.

Multiple continuous mode DIN transfers must be managed by CONTINUE statements. (WAIT_FOR_DATA management is impossible in this case.)

Only one such transfer can use the clock module (ST1 time base, internal clock control, or line mode). The other transfers must be using external strobe sampling (trigger = 0) or independent event control (with lines enabled by DIN_EVENT).

Errors

?MINC-F-Another transfer is in progress for the array specified

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Data-name array is shorter than sweep length requested

The number of elements available for data must be greater than or equal to the length of the sweep.

?MINC-F-Data lost—transfer rate too high

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for DIN.

The DIN statement specifies incompatible mode designators. Refer to the table of valid mode designator combinations.

?MINC-F-Previous routine is already using the module requested

The statement specifies timestamp mode but the trigger argument has some value other than 0.

Examples

See Example D, page 62, and Example F, page 67.

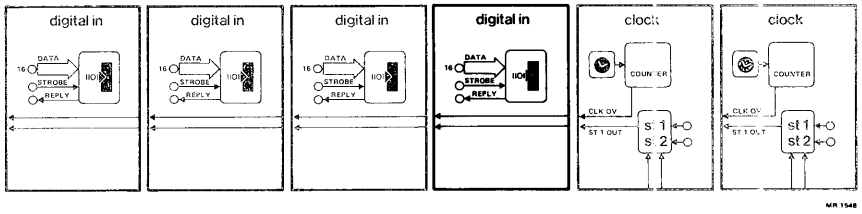
DIN_EVENT

Enable Response to Independent Digital Input Lines

Operation

DIN_EVENT enables single data lines in a specified digital input unit to control data transfer in DIN. The specified data lines act as independent events. Each bit set in the event-enable word enables DIN to read the whole unit when a transition occurs on the line corresponding to the bit. (See "Digital Sampling and Control," page 3.)

Configuration



Statement Form

DIN_EVENT(event-enable-word,unit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
event-enable-word	numeric expression	-32,768 to 32,767	all bits set (-1)
unit	numeric expression	0 to 3	0

Example DIN_EVENT(,2)
Result Enable DIN to sample when a signal occurs on any line of unit 2.

Example DIN_EVENT(BIN('1111 0000 0000 1111'),1)
Result Enable DIN to sample when a signal occurs on any of lines 0 to 3 and 12 to 15.

Example DIN_EVENT
Result Enable DIN to sample when a signal occurs on any line of unit 0.

Argument Descriptions

event-enable-word The word specifying which lines of the specified unit can function as independent events. (See "Bits and Lines," page 27.)

Values: Whole number in the range -32,768 to 32,767. (The event-enable word can contain any combination of 16 bits.)

Default: Bits for all lines are set (-1).

The event-enable word must have a whole number value in the range -32,768 to 32,767. You can calculate the numeric value for a given set of bits. However, all but the simplest values are most conveniently specified with the BIN function (Book 3) or, if only a few lines are involved, with SET_BIT.

unit The digital input unit whose lines are being enabled.

Values: 0 through 3

Default: 0

The digital input units are independent. Use one DIN_EVENT statement for each unit. The event-enable words for different units can all be different.

If no event-enable word has been specified for a unit, the event-enable bits for all lines in that unit are clear. That is, the lines in the unit function as a unit and cannot function as independent events.

Related Routines

DIN DIN collects data from the specified digital input unit. If the trigger argument in DIN is zero, and DIN_EVENT has enabled individual lines, then DIN reads the whole contents of the digital input unit into the data array whenever a transition occurs on an enabled line.

Restrictions

Immediate mode DIN_EVENT operates within a multistatement line in immediate mode. The system utility which produces the READY message clears the event-enable word.

Multiple units The digital input units are independent of each other. Use one DIN_EVENT statement for each unit to be controlled this way.

Errors

?MINC-F-Channel or unit # not in system for routine

Examples

See Example F, page 67.

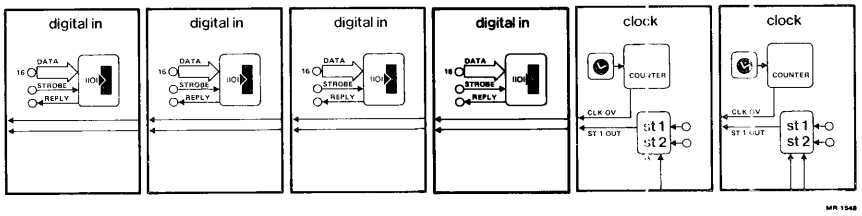
DIN_MASK

Define Digital Input Mask

Operation

DIN_MASK specifies a mask word for the specified digital input unit. DIN uses the mask word to mask the data collected from the unit. (See "Masking," page 27.)

Configuration



Statement Form

DIN_MASK(mask-word,unit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mask-word	numeric expression	-32,768 to 32,767	all bits set (-1)
unit	numeric expression	0 to 3	0

Example DIN_MASK(,1)
Result Specify a mask word for unit 1 that has the bits for all lines set.

Example DIN_MASK(16,1)
Result Specify a mask word for unit 1 that has the bit for line 4 set.

Example DIN_MASK(BIN('0000 0000 1111 1111'),2)
Result Specify a mask word for unit 2 that has bits set for lines 0 to 7.

Example DIN_MASK
Result Specify a mask word for unit 0 that has the bits for all lines set.

Argument Descriptions

mask-word The value used by DIN to mask data from the digital input unit.

Values: Whole number in the range -32,768 to 32,767. The mask-word value can be any combination of 16 bits.

Default: Bits for all lines are set (-1).

Each mask word is composed of 16 bits. (See "Bits and Words,"

page 26.) The most convenient method for specifying mask word values is to use the BIN function (see Book 3) or, if only a few bits are involved, SET_BIT.

Each bit corresponds to a line of the digital input unit. When the bit for a line is clear, then DIN always reads the value of that line as 0, regardless of the actual condition of the line. When the bit for a line is set, then DIN reads the actual condition of the line (0 if clear, and 1 if set).

unit The digital input unit for which the mask is specified by the DIN_MASK statement.

Values: 0 through 3

Default: 0

The digital input units are independent. Use one DIN_MASK statement for each digital input unit to be masked. The mask words can all be different.

If no mask has been specified for a unit, DIN collects data from that unit as if a default mask word with all bits on had been specified.

Related Routines

DIN DIN collects data from a specified digital input unit. If no mask word has been specified, or if the default mask word of all bits set was specified, DIN reads the condition of each line of the unit. If a mask word was specified, DIN reads the condition only of the lines for which the corresponding bit in the mask word was set. The data bits corresponding to clear mask bits are all clear.

DIN_EVENT DIN_EVENT enables individual lines to function as independent events. When DIN reads the digital input under independent event control, the bit for the line causing the event is always on (by definition). You can use DIN_MASK to suppress reading the event bit.

Restrictions

Immediate mode DIN_MASK operates within a multistatement line in immediate mode. The system utility which produces the READY message clears the mask words.

Multiple units DIN_MASK specifies a mask word for one digital input unit. Use one DIN_MASK statement for each digital input unit whose input you want to mask.

Errors

Channel or unit # not in system for routine

Examples

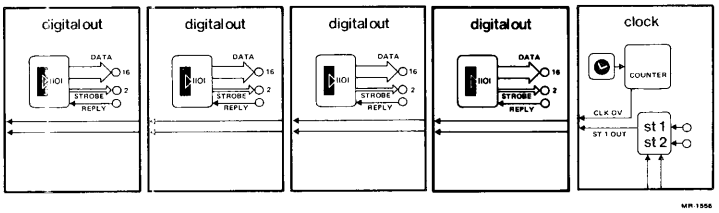
No example included.

DOUT
Send Digital Output

Operation

DOUT sends data to the digital output unit specified. DOUT can send a single value, a sweep of data, or a continuous stream of data. (See "Digital Sampling and Control," page 3.)

Configuration



Statement Form

DOUT(mode,**data-name**,data-length,trigger,unit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	CONTINUOUS, EXTERNAL, LINE,ST2	standard mode
data-name	numeric expression	-32,768 to 32,767	required argument
data-length	numeric expression	≥ 1	1
trigger	numeric expression	0; > 0 to 655.35 1 to 65,535	0
unit	numeric expression	0 to 3	0

Example DOUT(,V,1,0,2)
Result DOUT sends the contents of variable V to digital output unit 2 as soon as the DOUT statement executes.

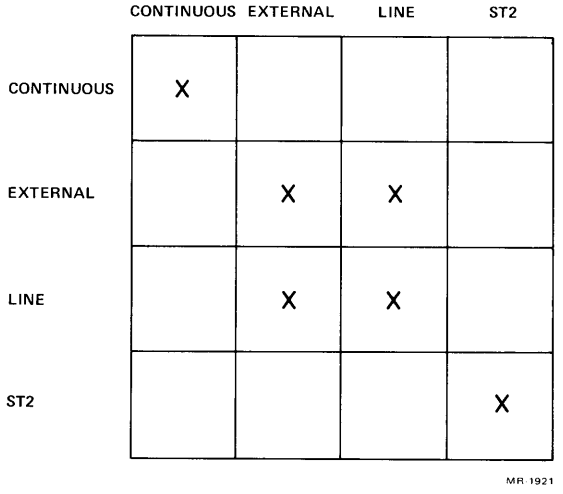
Argument Descriptions

mode The character string selecting optional modes for DOUT. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
CONTINUOUS	Continuous mode transfer. (See "Continuous Data Transfers," page 33.)
EXTERNAL	Enable an external source connected to ST1 to provide the time base.
LINE	Use line-frequency time base (50 or 60 Hz). (See "Time Base," page 43.)

ST2	Start the output process with a signal on ST2.
Default:	Standard mode (point or sweep transfers with either instrument triggering, trigger = 0, or clock module control, trigger > 0 to 655.35).

Some combinations of modes are valid and some are not. In the following table, "X" marks invalid mode pairs.



data-name The numeric data values to be sent by DOUT.

Values: -32,768 to 32,767

Default: Required argument

The data-name values represent the conditions for the digital output lines. DOUT interprets values in the integer range as 16 bit conditions. You can use SET_BIT or the BIN function to assign conditions to specified bits in the data word. If the output is going to a BCD instrument, use MAKE_BCD to create the necessary data-name values.

data-length The length of the data-name argument.

Values	*Meaning*
1	The data-name argument is a numeric expression.
> 1	The data-name argument is a numeric array

and the data-length argument is the number of array elements to be sent to the unit.

Default: 1

If data-length is greater than 1, data-name must be an array name.

trigger The code which specifies how to control the data transfer.

Values	*Meaning*
0	Send the digital output value immediately if data-length is 1.
	Send a digital output value as soon as a signal occurs on the reply terminal of the digital output unit.
> 0 to 655.35	Clock module control (without mode designators for external control)
1 to 65,535	External time base (with EXTERNAL or LINE designators)
Default:	0

(trigger = 0) When the trigger argument is zero, output to the digital output unit is controlled by signals on the reply line connected to the reply terminal of the digital output unit connector block. DOUT always sends the first value immediately. DOUT sends the next value as soon as the previous one has been acknowledged by the reply line (see Book 7).

(trigger > 0 to 655.35) When the trigger argument is greater than zero and none of the mode designators specifies an alternative time base, the rate of digital output is under internal clock control. DOUT chooses the appropriate clock module frequency (see "Time Base," page 43.)

In this case, the trigger argument specifies the transfer period in seconds. The transfer period is the time from the completion of one output transfer to the completion of the next.

The longest transfer period possible is 655.35 seconds. The shortest transfer period possible depends on the operating mode and on values of the other arguments.

The transfer frequency is the inverse of the transfer period

(trigger argument). For example, if the trigger argument is .015, the transfer frequency is 66 Hz; if the trigger argument is .002, the transfer frequency is 500 Hz. If the transfer period is 5, DOUT sends an output value every 5 seconds (0.2 Hz).

(trigger = 1 to 65,535) In external mode, DOUT sends data under the control of an external time base connected to ST1. In line mode, DOUT sends data using the line frequency mode of the clock module as the time base. The time base can have a regular rate (for example, line mode) or a varying rate. In either case, the trigger argument specifies the number of external signals necessary to cause each transfer. The maximum number possible is 65,535. If the trigger argument is 1, DOUT sends an output value on every timebase signal. If the trigger argument is 5, DOUT sends an output value on every fifth timebase signal.

If the external signals have a regular rate, the trigger argument acts as a rate divisor (or, equivalently, as a period multiplier). For example, if external signals are coming from a 60 Hz line frequency clock, then a trigger argument of 6 specifies a rate of 60/6 or 10 Hz, and a period of (1/60)*6 or 0.1 seconds (one transfer every 0.1 seconds).

unit The number of the digital output units carrying output data.

Values: 0 through 3

Default: 0

The digital output units are independent. Use multiple DOUT statements to send data to multiple digital output units.

Related routines

CONTINUE CONTINUE allows the program to resume executing during a continuous mode transfer and to continue executing until the currrent array partition has been transferred. Whenever an array partition becomes empty, CONTINUE transfers control to the service subroutine which refills that partition with data, while DOUT continues to output from the other partition. On returning from the subroutine, the program continues executing the statements following the CONTINUE statement. (See "Continuous Data Transfers," page 33, and CONTINUE.)

DOUT_MASK DOUT_MASK specifies a mask for lines in a digital output unit. If a line is masked with 0, then the condition of that line in the digital output unit is always cleared, regard-

less of the value in the output word. If a line is masked with 1, then the condition of that line in the digital output unit (set or clear) depends on the actual value in the output word (1 or 0). (See "Masking," page 27, DOUT_MASK, and Book 7.)

MAKE_BCD MAKE_BCD converts standard numeric data to BCD data format. If the instrument connected to the digital output unit expects to receive BCD data, your program must use MAKE_BCD to convert the numeric data to BCD format.

SET_LINE SET_LINE specifies the condition of a single output line in a specified digital output unit.

TERMINATE TERMINATE terminates continuous mode data output.

WAIT_FOR_DATA WAIT_FOR_DATA stops the program during a continuous mode transfer and the program waits for the current array partition to be output. When the array partition is empty, the program resumes executing to fill the empty array partition while DOUT continues to send values from the other one. (See "Continuous Data Transfers," page 33 and WAIT_FOR_DATA.)

Restrictions

Immediate mode Point and sweep DOUT transfers operate in immediate mode; continuous transfers do not.

Multiple units The digital output units are independent of each other. Use one DOUT statement for each unit being used. Different units must use different data arrays and, with CONTINUE, different service subroutines.

Multiple continuous mode DOUT transfers must be managed by the CONTINUE routine. (WAIT_FOR_DATA management is impossible in this case.)

Only one such transfer can use the clock module (internal clock control, line mode, or ST1 time base). The other DOUT transfer(s) must be using reply line control (trigger = 0).

Unit numbers The unit number specified in the DOUT statement must match the number of the unit installed in the system (see Book 7).

Errors

?MINC-F-Another transfer is in progress for the array specified

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Data-name array is shorter than sweep length requested

The number of elements available for data must be greater than or equal to the length of the sweep.

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for DOUT.

The DOUT statement specifies incompatible mode designators. Refer to the table of valid mode designator combinations.

?MINC-F-Value exceeds valid range for argument

The output values must be in the integer range, -32,768 to +32,767.

See Example D, page 62.

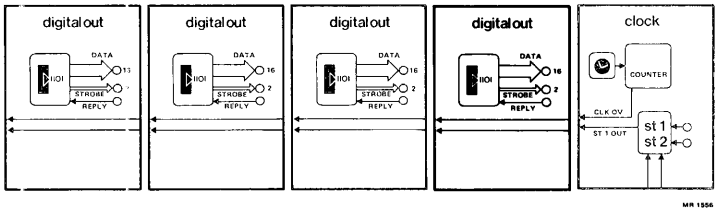
Examples

DOUT_MASK
Define Digital Output Mask

Operation

DOUT_MASK specifies a mask word for the specified digital output unit. DOUT uses the mask word to mask the data sent from the unit. (See "Masking," page 27.)

Configuration



Statement Form

DOUT_MASK(mask-word,unit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mask-word	numeric expression	-32,768 to 32,767	all bits set (-1)
unit	numeric expression	0 through 3	0

Example DOUT_MASK(,2)
Result Specify a mask word for unit 2 that has the bits for all lines set.

Example DOUT_MASK(15,1)
Result Specify a mask word for unit 1 that has bits set for lines 0 to 3.

Example DOUT_MASK(BIN('1111 1111 0000 0000'),2)
Result Specify a mask word for unit 2 that has bits set for lines 8 to 15.

Example DOUT_MASK
Result Specify a mask word for unit 0 that has the bits for all lines set.

Argument Descriptions

mask-word The value used by DOUT to mask data sent to the digital output unit.

Values: Whole number in the range -32,768 to 32,767. The mask-word argument can be any combination of 16 bits.

Default: Bits for all lines are set (-1).

If the mask-word argument is a numeric expression, its range of

values is from -32,768 to 32,767. If the mask-word argument is a real value, it must be a whole number in the integer range. Any fractional part is truncated.

The most convenient method for specifying a mask-word value is to use the BIN function (see Book 3) or, if only a few bits are involved, the SET_BIT routine.

Each mask word is composed of 16 bits. (See "Bits and Words," page 26.) Each bit corresponds to a line of the digital output unit (see Book 7). When the mask bit for a line is clear, then DOUT always clears that line, regardless of the value for that line in the output word. When the mask bit for a line is set, then DOUT sets the condition of the line according to the value for that line in the output word (0 if clear, and 1 if set).

unit The digital output unit for which the mask is specified by the DOUT_MASK statement.

Values: 0 through 3

Default: 0

The digital output units are independent. Use one DOUT_MASK statement for each digital output unit to be masked. The mask words can all be different.

If no mask has been specified for a unit, DOUT sends data to that unit as if a default mask word with all bits on had been specified.

Related Routines

DOUT DOUT sends data to a specified digital output unit. If no mask word has been specified, or if the default mask word of all bits set was specified, DOUT sets the condition of each line of the unit. If a mask word was specified, DOUT sets the condition only of the lines for which the corresponding bit in the mask word was set.

Restrictions

Immediate mode The DOUT_MASK routine operates within a multistatement line in immediate mode. The system utility which produces the READY message clears the mask words.

Multiple units DOUT_MASK specifies a mask word for one digital output unit. Use one DOUT_MASK statement for each digital output unit that requires a mask word.

Errors

?MINC-F-Channel or unit # not in system for routine

Examples

No example included.

FFT

Perform Fast Fourier Transform

Operation

FFT performs a discrete Fourier transform on a data array. The fast Fourier transform algorithm provides an efficient method for numerically approximating a continuous Fourier transform.

The continuous Fourier transform converts functions in the time domain to expressions in the frequency domain. Although the FFT routine is based on the discrete Fourier transform algorithm, it takes advantage of certain computational shortcuts to reduce the time required to produce the results.

FFT calculates the real and imaginary parts of the Fourier coefficients.

Configuration

FFT is a valid statement in any MINC program. It has no configuration requirements.

Statement Form

FFT(mode,data-length,real-component,imag-component,scale-factor)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	FORWARD or REVERSE	FORWARD
data-length	numeric expression	2^n (n = 3 to 11)	required argument
real-component	integer array name	input data	required argument
imag-component	integer array name	input data	required argument
scale-factor	integer variable name	integer range	required argument

Example FFT(,128,R%(),C%(),S%)
Result Perform a discrete Fourier transform of the first 128 values in the integer arrays R% and C%, storing the results of the transform in arrays R% and C%. Store the scale factor in variable S%.

Example FFT('REVERSE',128,R%(),C%(),S1%)
Result Perform a reverse (or inverse) discrete Fourier transform of the first 128 values in the integer arrays R% and C%, storing the results of the transform in arrays R% and C%. Store the scale factor in variable S1%.

mode The character string specifying the direction of the transform. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
FORWARD	Perform a forward transform.
REVERSE	Perform a reverse transform (also called an inverse transform).
Default:	FORWARD (forward transform)

Provided the appropriate scale factors are applied to the results, a forward transform followed by a reverse transform produces results exactly the same as the input to the forward transform.

After a reverse transform, you must divide each element of the resulting arrays by data-length.

data-length The number of points in each of the component arrays to use as input to the Fourier transform.

Values: 2^n (where n = 3 to 11)

Default: Required argument

The real-component and imag-component arrays must each contain at least data-length elements. The maximum value permitted for data-length is 2048. The actual maximum array length depends on the workspace available for arrays (see Book 3, LENGTH).

real-component The name of the integer array containing the real component of the (complex) input data to be transformed.

Values: -32,768 to 32,767

Default: Required argument. FFT replaces the input data with the calculated frequency coefficients.

The nature of the FFT algorithm requires that the data satisfy certain conditions (see Restrictions).

imag-component The name of the integer array containing the imaginary component of the (complex) input data to be transformed.

Values: -32,768 to 32,767

Default: Required argument. FFT replaces the input data with the calculated frequency coefficients.

The nature of the FFT algorithm requires that the data satisfy certain conditions (see Restrictions). If the input data are real rather than complex, all elements in the imag-component array are 0.

scale-factor The integer scale factor used to scale the real and imaginary components of the result to provide the final result.

Values: Integer range

Default: Required argument. FFT assigns the value to the scale-factor argument.

For efficient calculations, FFT requires integer input arrays. However, because integer values are limited in range, FFT requires a mechanism for keeping the data values within the integer range during its calculations. It does this by dividing all the values by 2 whenever necessary to keep within the range. Therefore, the results are proportionally correct but need to be scaled in order to restore the original absolute values. After FFT has finished, the scale-factor argument contains the number of divisions. Create final results in two real arrays by multiplying each element of the integer result arrays by 2^n (where n is the scale factor).

Related Routines

POWER POWER calculates the power spectrum of a set of data by using the real and imaginary coefficient arrays calculated by FFT.

Restrictions

Conditions The discrete Fourier transform algorithm provides an approximation to the theoretically desired continuous Fourier transform. The goodness of the approximation is highly dependent on how well the input data satisfy the following conditions:

1. The function to be transformed must be periodic.
2. The function to be transformed must be band-limited. That is, the highest frequency component of the function must be finite.
3. When you collect the input data, the sampling rate must be higher than twice the highest frequency component of the function.

4. The input data sampling must collect an exact multiple number of the periodic waveforms.

When the above conditions are not met, discrepancies begin to accumulate between the continuous Fourier transform and its discrete numerical approximation. If the function is not periodic and band-limited, then the other conditions cannot be satisfied, by definition. If the sampling does not result in an exact integer multiple of the period of the input waveform, then the FFT results are distorted by *leakage*.

If the sampling rate is too slow, then the FFT results exhibit *aliasing*. That is, the FFT coefficients identify frequency components that are not present in the input waveform. The following figure illustrates the meaning of aliasing.

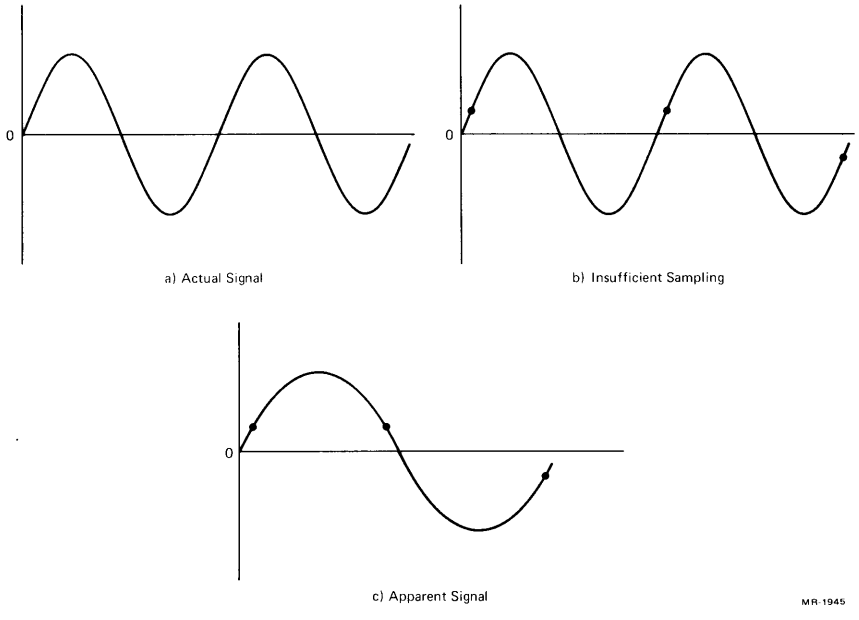


Figure 44. How Aliasing Occurs.

It is important to recognize causes of discrepancy between the discrete Fourier transform and the continuous transform in order to use the FFT routine effectively. Consult a good general text for a complete, formal description of the discrete transform itself, the discrepancies that can occur, and the methods necessary to minimize the discrepancies. (For example, see *The Fast Fourier Transform* by E. Oran Brigham, Prentice-Hall, 1974, or the manual for the Laboratory Subroutines Package, Order No. AA-C984A-TC, from Digital Equipment Corporation.)

Scaling Two different scale factors affect FFT results.

1. If you want absolute values (rather than proportional values), multiply all elements of the real-component and imag-component arrays by 2^n (where n is the scale-factor argument).

2. After a reverse transform, you must divide each element of the real-component and imag-component arrays by data-length.

Errors

?MINC-F-Arrays must be as large as number of points requested

?MINC-F-Fewer than 8 points requested

?MINC-F-Invalid or conflicting options requested

FFT permits only one mode designator.

?MINC-F-More than 2048 points requested

?MINC-F-Notify DIGITAL: FFT argument failure

?MINC-F-Number of points must be a power of 2

Examples

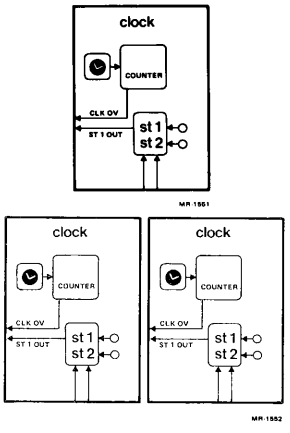
See F6FFT.BAS on the demonstration diskette.

GET_TIME
Read Current Elapsed-Time Count

Operation

GET_TIME reads the current value in the elapsed-time counter. GET_TIME interprets the counter value either as time or as number of external signals, depending on the operating mode of the elapsed-time counter. (See "Measuring Time Intervals," page 3.)

Configuration



Statement Form

GET_TIME(current-value)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
current-value	numeric variable name	≥ 0	required argument

Example GET_TIME(V)
Result Read the current elapsed time or signal count into variable V.

Example GET_TIME(A(5))
Result Read the current elapsed time or signal count into element 5 of array A.

Argument Descriptions

current-value The name of the variable that is to contain the current elapsed time or external signal count.

Values: ≥ 0

Default: Required argument. GET_TIME assigns the value to the argument.

The current-value argument contains either the time elapsed or the number of external signals occurring since the elapsed-time counter started. The operating mode of the elapsed-time counter

(set by START_TIME) determines whether the current value is a time or a count. If the elapsed-time counter is set for a regular rate (that is, any one of the LINE, CHZ, or KHZ mode designators) the current-value argument contains the time in seconds since the elapsed-time counter started. If the elapsed-time counter is set for a varying rate (that is, external mode), the current value contains the number of external signals occurring since the counter started.

Related Routines

START_TIME START_TIME sets the operating mode for the elapsed-time counter and starts the counter at zero.

DIN The timestamp mode in DIN provides a special case of time measurement for recording when digital input events occurred.

Restrictions

Clock conflict GET_TIME and START_TIME use clock 0 if only one clock module is present in the system. If GET_TIME is using clock 0, you cannot request any clock-controlled data transfers. If the system contains two clock modules (clock 0 and clock 1), GET_TIME uses clock 1 and any data transfer routine uses clock 0.

Immediate mode The GET_TIME routine does not operate in immediate mode.

Integer result The current-value argument can be an integer variable. If the elapsed-time value is a time interval, the variable contains time truncated to integral seconds. That is, if the actual elapsed time is 5.62 seconds, the integer current value is 5 seconds. If the elapsed-time value is counts, the variable contains the number of counts, just as it would if the variable were real. However, the maximum number of counts possible with an integer variable is 32,767. If more than 32,767 counts have elapsed since the elapsed-time counter started, the count is too large for an integer variable and the program halts with an error.

Errors

?MINC-F-START_TIME statement must precede the time request

Examples

See Example E, page 65.

MAKE_BCD

Convert Variable to BCD Format

Operation

MAKE_BCD converts a standard numeric value to BCD format. (See "Format Conversion," page 25.) Its usual application is to convert numeric values to BCD format for digital output with DOUT.

Configuration

MAKE_BCD is a valid statement in any program. It has no configuration requirements.

Statement Form

MAKE_BCD(numeric-value,BCD-value)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
numeric-value	numeric expression	0 to 9999	required argument
BCD-value	numeric variable name	0 to 9999 BCD	required argument

Example MAKE_BCD(100,B)
Result Assigns 100 in BCD format to variable B.

Example MAKE_BCD(G%,B)
Result Converts the value in G% to BCD format, and assigns the BCD value to variable B.

Example MAKE_BCD(V,C%)
Result Converts the value in V to BCD format, and assigns the BCD value to variable C%.

Argument Descriptions

numeric value The number to be converted to BCD format.

Values: 0 to 9999

Default: Required argument

BCD-value The variable to contain the BCD value.

Values: 0 to 9999 in BCD format

Default: Required argument. MAKE_BCD assigns a value to the argument.

Related Routines

MAKE_NUMBER MAKE_NUMBER reverses the effect of MAKE_BCD. MAKE_NUMBER converts a value in BCD format to a numeric value.

DOUT DOUT sends data to digital output units. The digital

output unit could be connected to an instrument that sends and receives BCD format data. Use MAKE_BCD to convert numeric data to BCD format data before sending it to these instruments.

Restrictions

None included

Errors

?MINC-F-Value exceeds BCD range for argument #

Examples

See Example D, page 62.

MAKE_NUMBER

Convert BCD Format to Numeric Value

Operation

MAKE_NUMBER converts a value in BCD format to a standard numeric value. (See "Format Conversion," page 25.)

Configuration

MAKE_NUMBER is a valid statement in any program. It has no configuration requirements. However, it does require BCD values, which are often acquired from digital input units (with DIN).

Statement Form

MAKE_NUMBER(BCD-value,numeric-value)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
BCD-value	numeric expression	0 to 9999 BCD	required argument
numeric-value	numeric variable name	0 to 9999	required argument

Example MAKE_NUMBER(B%,N%)
Result Converts the BCD value in variable B% to numeric format and assigns the new value to N%.

Example MAKE_NUMBER(V,V1)
Result Converts the BCD value in variable V to numeric format and assigns the new value to V1.

Argument Descriptions

BCD-value The value in BCD format which is to be converted to numeric format.

Values: BCD values whose numeric equivalents are in the range 0 to 9999

Default: Required argument

numeric-value The variable to contain the numeric equivalent of the BCD value.

Values: 0 to 9999

Default: Required argument. MAKE_NUMBER assigns a value to the argument.

Related Routines

MAKE_BCD MAKE_BCD reverses the effect of MAKE_NUMBER. MAKE_BCD converts a numeric value to BCD format.

DIN DIN collects data from digital input units. If an input unit

is connected to an instrument that sends and receives BCD values, then DIN collects data from that instrument in BCD format. Use MAKE_NUMBER to convert the BCD format data to numeric values that the program can use.

Restrictions None

Errors ?MINC-F-Invalid BCD value specified in argument #

Examples See Example D, page 64.

MAKE_TIME

Convert DIN Timestamp Values to Numeric Values

Operation

MAKE_TIME converts the DIN format timestamp values collected by DIN to standard numeric values. (See "Number Systems," page 25.)

Configuration

MAKE_TIME is a valid statement in any program. It has no configuration requirements.

Statement Form

MAKE_TIME(,old-value,result)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
rate	string expression	CHZ,EXTERNAL, KHZ,LINE,ST2	current rate
old-value	integer expression	-32,768 to 32,767	required argument
result	real variable name	0 to 65,535 counts 0 to 1092.2 seconds	required argument

Example MAKE_TIME('CHZ,ST2',I%,R)
Result Convert the DIN format value contained in I% to time in seconds and assign it to variable R. (The ST2 mode designator has no effect.)

Example MAKE_TIME('EXTERNAL',I%,R)
Result Convert the DIN format value in I% to standard numeric format and assign the count value to variable R.

Example MAKE_TIME(,I%,R)
Result Convert the DIN format value in I% to standard numeric format. Interpret I% either as time or counts depending on the operating mode most recently set for the elapsed-time counter. Assign the value to variable R.

Example MAKE_TIME('LINE',P%(8),V(4))
Result Convert the DIN format value in array element P%(8) to time in seconds and assign the time value to element 4 of array V.

Argument Descriptions

rate The character string specifying the operating mode of the elapsed-time counter when DIN collected the timestamp values. (See "Operating Modes and Mode Designators," page 23, and DIN.)

The values have the same meanings as for START_TIME,

which starts the elapsed-time counter.

Values	*Meaning*
CHZ	Assume 100 Hz time base; values are times in seconds
EXTERNAL	Assume ST1 signals provided the time base; values are counts
KHZ	Assume 1000 Hz time base; values are times in seconds
LINE	Assume line-frequency time base; values are times in seconds
ST2	Use for compatibility with the START_TIME mode string if required
Default:	The current timestamp counter operating mode, or the one most recently specified (excluding HALT designator).

The rate string can contain more than one mode designator. However, many combinations of modes are invalid. Any rate string specifying more than one rate designator is invalid. In the following table, "X" marks the invalid mode combinations.

	CHZ	EXTERNAL	KHZ	LINE	ST2
CHZ	X	X	X	X	
EXTERNAL	X	X	X	X	
KHZ	X	X	X	X	
LINE	X	X	X	X	
ST2					X

MR-1920

MAKE_TIME uses the rate argument to interpret the timestamp values collected by DIN. If the elapsed-time counter used a regular time base with mode designator LINE, CHZ, or

KHZ, MAKE_TIME converts the values to time in seconds. If the elapsed-time counter used an external signal source (EXTERNAL mode designator), MAKE_TIME converts the values to counts.

If no rate argument appears, then MAKE_TIME assumes that the timestamp values were collected using the most-recently specified elapsed-time counter rate. If the timestamp values were collected in a previous program, then MAKE_TIME halts with an error if no rate argument appears in the statement (or earlier in the program). That is, if the values were collected by a previous program, the rate argument must specify the operating mode of the elapsed-time counter when the values were collected.

old-value The nonstandard integer timestamp value collected by DIN.

Values: -32,768 to 32,767

Default: Required argument

DIN collects timestamp values in a nonstandard integer format. (See "Number Systems," page 25.) The program cannot perform valid arithmetic operations on these nonstandard values.

result The converted timestamp value ready for use by the program.

Values: 0 to 65,535 counts

0 to 1092.25 seconds

Default: Required argument. MAKE_TIME assigns a value to the argument.

The converted value contains time in seconds if the operating mode had a regular rate or a signal count if the elapsed-time counter was counting external signals. The signal count values can range from 0 to 65,535. The time value range depends on the operating mode of the elapsed-time counter, as shown in the following chart.

Operating Mode	*Time Range*
CHZ	0 to 655.35 seconds (about 11 minutes)
KHZ	0 to 65.535 seconds (about 1 minute)

LINE	0 to 1092.25 seconds (about 18 minutes)

Related Routines

DIN DIN can collect digital input data in timestamp mode. In timestamp mode, DIN time stamps every sample by reading the value of the elapsed-time counter (when it happened) and reading the value of the digital input unit (what happened). DIN reads the elapsed-time counter in a nonstandard integer format that the program must later convert to numeric format in order to perform arithmetic on the timestamp values.

START_TIME, GET_TIME You can collect elapsed-time data using GET_TIME after a START_TIME statement starts the elapsed-time counter. GET_TIME collects data in standard numeric format, with no limit on the maximum time or number of counts. Therefore, do not use MAKE_TIME for elapsed-time values collected with GET_TIME.

Restrictions

Timestamp limits DIN and MAKE_TIME produce values within ranges limited by the operating mode of the elapsed-time counter. It is possible to infer longer times or larger count values by inspecting the whole series of timestamp values.

For example, consider the timestamp series shown in Figure 45.

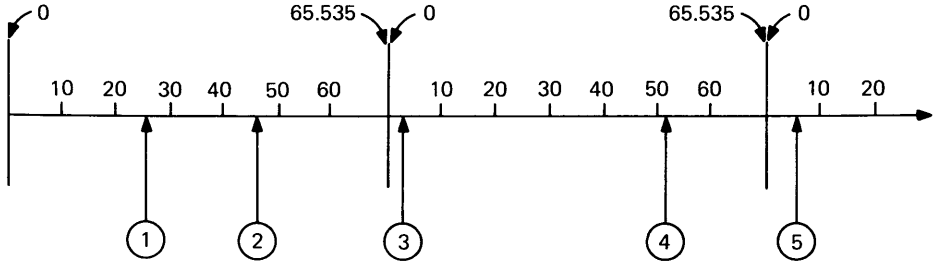


Figure 45. Inferring Elapsed Time from Timestamp Values

The five circled numbers represent five timestamp points. Table 6 shows timestamp values for each of the points and the elapsed-time values that you could infer.

Table 6. Inferred Elapsed Time Values

Point	*Value of Result Argument*	*Inferred Elapsed Time*
1	26.1	26.1
2	46.8	46.8
3	3.9	69.4
4	51.9	117.4
5	6.5	137.6

Inferring total elapsed time is reasonably safe if you can assume that the variance of the timestamp distribution is small and that the timestamp intervals themselves are short relative to the maximum timestamp interval. (Note: For the purposes of illustration, the example above violates both of these assumptions.)

Errors

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for MAKE_TIME.

The MAKE_TIME statement specifies incompatible mode designators. Refer to the table of valid mode designator combinations.

?MINC-F-Must specify clock operating mode

Examples

No example included.

PAUSE

Suspend Program Execution

Operation

PAUSE suspends program execution for a specified time interval. After the time interval expires, program execution continues with the statement following the PAUSE statement. (See "Program Dynamics for Control Programs," page 29.)

Configuration

PAUSE is a valid statement in any MINC program. It has no configuration requirements because it uses the internal system clock, not the clock module.

Statement Form

PAUSE(delay-interval)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
delay-interval	numeric expression or string expression	0 to 86,400 sec 0 to 24:00:00	required argument

Example PAUSE(0.5)
Result The program waits half a second before continuing.

Example PAUSE(120)
Result The program waits two minutes before continuing.

Example PAUSE('1:5:30')
Result The program waits one hour, 5 minutes, and 30 seconds before continuing.

Example PAUSE('12::')
Result The program waits twelve hours before continuing.

Example PAUSE('10') or PAUSE(10)
Result The two statements are equivalent. The program waits 10 seconds before continuing.

Argument Descriptions

delay-interval The length of time to wait before continuing execution.

Values	*Meaning*
numeric	0 to 86,400 seconds.
string	0 to 24:00:00 (time interval in hours:minutes:seconds format).

Default: Required argument

(numeric expressions) When the delay interval is a numeric expression, 0.1 seconds is the shortest interval guaranteed to be precise. Intervals shorter than 0.1 seconds are accurate to the nearest system clock tick.

(string expression) When the delay interval is a string expression, one second is the shortest possible interval. The time string has the frame "hours:minutes:seconds". Each of the slots in the frame (hours, minutes, seconds) can hold up to two digits, or can be empty (null). The maximum interval possible in the time string is 24:00:00. The string itself can contain only digits and colons; blanks are invalid in the time string.

If the string contains only one number, PAUSE assumes that the number specifes seconds, unless the number is followed by one or two colons. The colons indicate the position of the number in the string frame. For example, the string '5' means 5 seconds, the string '5:' means 5 minutes, and the string '5::' means 5 hours. Thus, the string ':10:' is equivalent to '0:10:0' and also to '10:'.

Related Routines

SCHEDULE SCHEDULE designates a service subroutine and schedules it to execute after a specified time interval or at a specified time of day. The program continues executing during the time interval.

Restrictions

CTRL/C The CTRL/C key combination works normally during a pause. In fact, CTRL/C is the only method available for aborting a pause. It is probably not advisable to use the RCTRL/C function (see Book 3) to disable CTRL/C during pauses.

During a pause While the program is waiting, it does not recognize any external signals, or perform any scheduled service subroutines. Continuous mode data transfers continue until both array partitions have been transferred (and then halt with an error if the pause is still in progress).

Response to external signals or requested events is postponed until the delay interval completes. That is, the response is delayed, not lost. The program can delay responding to up to eight intervening requests. If more than eight signals or events requiring response occur during a pause, the program halts with an error.

Long pauses PAUSE accepts long time intervals. Therefore, if

you initiate a long pause, the system might appear broken or someone else might mistakenly enter CTRL/C and start using the system for something else, destroying your program. To avoid problems like these, you could print a message on the screen before beginning the pause, for example,

```
110 PRINT T$' hour delay in progress, started at 'CLK$
120 PAUSE(T$)
```

or

```
110 PRINT T' second delay in progress was started at ';CLK$
120 PAUSE(T)
```

With valid long delays that span midnight, MINC correctly changes the time from 24:00 to 00:00. Except at the end of a month, when MINC also automatically changes the date.

Time of day PAUSE accepts only time interval arguments. As a result, you cannot specify an absolute time of day as the end of a pause. There are two solutions to the problem.

1. You can compute the time interval between the current time and the desired time and use that interval in a PAUSE statement.

2. You can use SCHEDULE with the desired time of day as the argument and follow the SCHEDULE statement with a loop which continues to execute until the time of day arrives. For example:

```
50 F = 0
60 SCHEDULE('Absolute','12::',200)
70 IF F = 0 GOTO 70
.
.
.
200 REM Service subroutine for SCHEDULE
210 F = 1
220 RETURN
.
.
.
```

Errors

?MINC-F-Time string format must be hh:mm:ss

?MINC-F-Value of argument # exceeds valid range

Examples

See Example B, page 58, and Example D, page 62.

POWER

Calculate Power Spectrum Coefficients

Operation

POWER calculates the power spectrum for a set of complex Fourier coefficients. The power spectrum is the relationship between power level and signal frequency.

Configuration

POWER is valid in any MINC program. It has no specific configuration requirements.

Statement Form

POWER(data-length,real-component,imag-component, spectrum-array)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
data-length	numeric expression	$\geqslant 1$	required argument
real-component	integer array	array from FFT	required argument
imag-component	integer array	array from FFT	required argument
spectrum-array	real array name	result array	required argument

Example POWER(128,R%(),C%(),P())
Result Calculate the power spectrum using the real-coefficient array R% and the imaginary-coefficient array C% as input. Put the power spectrum in array P.

Argument Descriptions

data-length The number of elements in each of the three arrays.

Values: $\geqslant 1$

Default: Required argument

The destination array for the power spectrum must contain at least data-length elements. In practice, the length of the arrays is limited by the workspace available for arrays (see Book 3, LENGTH).

real-component The integer array containing one set of coefficients.

Values: Integer range

Default: Required argument

The real-component array can contain any set of coefficients.

One such set of coefficients is the real component coefficients produced by the FFT routine.

imag-component The integer array containing one set of coefficients.

Values: Integer range

Default: Required argument

The imag-component array can contain any set of coefficients. One such set of coefficients is the imaginary coefficients produced by the FFT routine.

spectrum-array The real array to contain the power spectrum coefficients.

Values: Real number range
Default: Required argument. POWER assigns values to the array.

Related Routines

FFT FFT calculates complex Fourier coefficients given a set of complex data, using a discrete fast Fourier transform algorithm.

Restrictions

Relative power Each power spectrum coefficient is the sum of the squares of the corresponding real and imaginary coefficients.

The traditional definition of power is the square root of the sum of the squares of the coefficients.

$$P_a = \sqrt{(R_a^2 + I_a^2)}$$

Therefore, to obtain actual power coefficients, take the square root of each element in the spectrum array.

In general, POWER and FFT are used to obtain proportionally correct values, not absolute values.

Generality The POWER routine is not limited to power spectra for FFT coefficients. You can use POWER to calculate the sums of the squares of any pairs of values.

Errors

?MINC-F-Arrays must be as large as number of points requested

?MINC-F-Number of points requested must be positive

Examples None included.

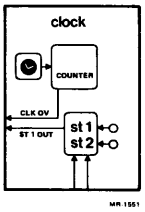
PST_HIST

Generate Post-Stimulus Time Histogram

Operation

PST_HIST generates a post-stimulus time histogram. PST_HIST measures the time intervals between an initial (*stimulus*) signal on ST1, and subsequent (*response*) signals on ST2. For each response signal, PST_HIST measures the elapsed time relative to the stimulus signal. PST_HIST collects multiple sweeps of time interval data (one sweep for each stimulus signal) and generates a histogram array using all the time interval data. (See "Frequency Histograms," page 45.)

Configuration



Statement Form

PST_HIST(mode,histogram-name,tick-rate,no.-of-sweeps,lower-endpoint,upper-endpoint)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	DISPLAY,ZERO	standard mode
histogram-name	integer array name	0 to 32,767	required argument
tick-rate	numeric expression	2 to 6	required argument
no.-of-sweeps	numeric expression	1 to 32,767	required argument
lower-endpoint	numeric expression	0 to 65,535 ticks	0
upper-endpoint	numeric expression	0 to 65,535 ticks	65,535

Example PST_HIST(,H%(),2,10)
Result Collect 10 sweeps of time interval data with a time base of 100 Hz. Generate the frequency histogram H% using the full time range.

Example PST_HIST('DISPLAY',H%(),3,100,1,500)
Result Collect 100 sweeps of time interval data with a time base of 1 KHz. Generate the frequency histogram H% for time intervals in the range 1 to 500 msec.

Argument Descriptions

mode The character string selecting an optional mode for PST_HIST. (See "Operating Modes and Modes Designators," page 23.)

Values	*Meaning*
DISPLAY	Display the generated histogram on the screen.
ZERO	Set all elements of the histogram array to zero before starting.
Default:	Standard mode (no display; use existing array)

Combining the two mode designators is valid.

(mode = DISPLAY) In display mode, PST_HIST displays on the screen the contents of the first 512 elements of the histogram array. Each element contains the count for the corresponding bin of the histogram. That is, the value shown on the screen for a bin represents the number of occurrences of time intervals within that bin range.

PST_HIST clears the screen before beginning the display. The display includes a horizontal axis, but no vertical axis. The count data are displayed so that the full vertical scale of the screen represents the largest current count value. No units are shown. Therefore, the display shows the relative counts in the histogram bins, but not the absolute counts.

PST_HIST produces a display suitable for monitoring purposes. Use the specialized graphic routines for final, labeled displays (see Book 4).

histogram-name The name of the integer histogram array.

Values:	0 to 32,767 counts
Default:	Required argument. PST_HIST assigns values to the histogram array.

PST_HIST uses the full length of the array to store the histogram. The number of elements in the array definition determines the number of bins in the histogram. (See "Frequency Histograms," page 45.) The histogram array must contain at least four elements.

PST_HIST reserves the first and last elements of the array as overflow bins to accumulate the counts for intervals outside the range of interest. That is, for an interval shorter than the specified minimum, PST_HIST increments the first element in the array. For an interval longer than the specified maximum, it in-

crements the last element in the array. The remaining array elements contain the number of occurrences of intervals in the corresponding bin ranges.

tick-rate The time base for measuring time intervals.

Value	*Time Base*
2	100 Hz
3	1 KHz (1000 Hz)
4	10 KHz (10,000 Hz)
5	100 KHz (100,000 Hz)
6	1 MHz (1,000,000 Hz)

Default: Required argument

The values for tick rates are the powers of ten for the rates expressed in Hz. For example, a tick rate of 4 specifies 10 to the power 4, that is, 10,000 Hz.

no.-of-sweeps The number of time interval sweeps required.

Values: 1 to 32,767

Default: Required argument

PST_HIST measures the time of occurrence of a response signal on ST2 relative to an initial stimulus signal on ST1.

The occurrence of the stimulus signal on ST1 defines the beginning of the sweep. The occurrence of the next stimulus signal on ST1 defines the end of the current sweep, and the beginning of the next sweep. Therefore, the number of ST1 signals occurring must be one greater than the number of sweeps required, because the final ST1 signal defines the end of the last sweep and the end of sampling.

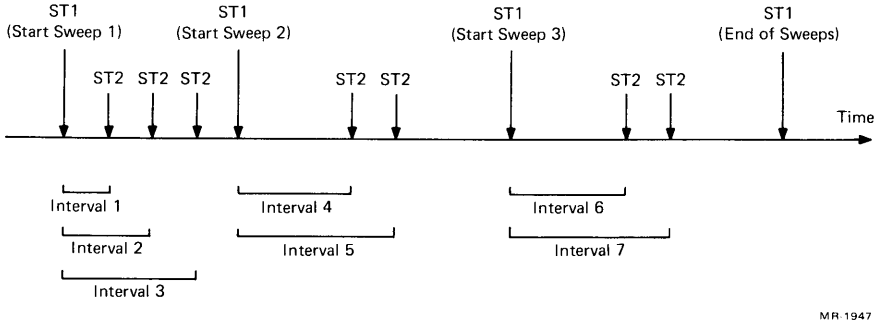


Figure 46. Defining Sweeps and Intervals for PST_HIST.

lower-endpoint The lower-endpoint of the range of interest (shortest time interval of interest).

Values: 0 to 65,535 ticks

Default: 0 ticks

The lower-endpoint argument defines the lower endpoint of the histogram range of interest. Any time intervals shorter than the minimum increase the count in the lower overflow bin (element 0) of the histogram array. If the minimum interval is zero, then the first element of the array is always empty (because no time interval can be shorter than zero, by definition).

upper-endpoint The upper-endpoint of the range of interest (longest time interval of interest).

Values: 0 to 65,535 ticks

Default: 65,535 ticks

The upper-endpoint argument defines the upper-endpoint of the histogram range of interest. Any time intervals longer than the maximum increase the count in the upper overflow bin (last element) of the histogram array. If the maximum interval is 65,535, then the last element of the histogram array contains the number of time intervals that exceeded the capacity of the clock counter.

The longest valid interval is 65,535 clock ticks. The absolute value of this interval depends on the tick rate.

Tick Rate	*Time Interval Range*
2	0 to 655.35 seconds
3	0 to 65.535 seconds
4	0 to 6.5535 seconds
5	0 to 0.65535 seconds
6	0 to 0.06554 seconds

If the maximum interval elapses before the stimulus signal for the next sweep occurs, then the last element of the histogram could contain a large count if many ST2 signals occur before the next ST1 signal starts the next sweep.

Related routines

TIME_HIST TIME_HIST measures the time intervals between successive signals on ST2, and generates a frequency histogram using the time interval values. TIME_HIST measures

the time elapsed since the most recent ST2 signal, unlike PST_HIST which measures the time elapsed since the initial ST1 signal.

Restrictions

Bin count limit The maximum count possible in any histogram bin is 32,767. This places an upper limit on the number of sweeps and intervals. When any count reaches the upper limit, PST_HIST "locks" the value at the limit. In practice, the limit should rarely be reached unless you have very wide histogram bins and large samples and do repeated histograms without using the ZERO mode designator.

DISPLAY PST_HIST displays the histogram array continuously so that you can monitor the progress of the data acquisition. However, the display rate is different from the acquisition rate and the screen display could be updated before the full sweep of response signals has been collected. This occasionally results in displays that are hard to interpret because the full vertical scale of the display is based on the largest count in the last complete sweep.

Errors

?MINC-F-Clock too fast for system to respond

?MINC-F-Histogram arrays must contain at least 4 elements

?MINC-F-Existing display conflicts with display requested

A prior AIN statement specified continuous display mode. PST_HIST cannot erase this display.

?MINC-F-Value of argument # exceeds valid range

Examples

No example included.

SCAN_BIT

Test Condition of All Bits in a Word

Operation

SCAN_BIT examines the condition of the bits in a specified word sequentially. SCAN_BIT scans the bits in the word, starting with bit 0. It assigns to an argument the bit position of the first bit it encounters that has been set and clears that bit. (See "Bits and Words," page 26.)

Configuration

SCAN_BIT is valid in any program. It has no configuration requirements.

Statement Form

SCAN_BIT(bit-position,word)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
bit-position	numeric variable name	-1; 0 to 15	required argument
word	numeric variable name	-32,768 to 32,767	required argument

Example SCAN_BIT(B1,B)
Result Assume that B contains the value 2. The statement scans the value 2 starting with bit 0, finds the first bit set in bit 1, clears that bit in B, and assigns 1 as the value of variable B1. The variable B now contains the value 0.

Example SCAN_BIT(P%,E)
Result Assume that E contains the value 5. The statement scans the value 5 starting with bit 0, finds the first bit set in bit 1, clears that bit in E, and assigns 1 as the value of variable P%. The variable E now contains the value 4.

Example SCAN_BIT(V,V%)
Result Scans the value in V%, starting with bit 0, and assigns to V the bit position of the first bit set (if any have been set). If the value of V% were 0, the value of V after the statement executed would be -1.

Argument Descriptions

bit-position The variable name receiving the value determined by SCAN_BIT.

Values	*Meaning*
-1	All bits were clear; the value in the word argument was 0.

Values	*Meaning*
0 to 15	The bit position of the first set bit encountered within the word.
Default:	Required argument. SCAN_BIT assigns a value to the argument.

word The numeric value whose bits are to be tested.

Values:	Whole number in the range -32,768 to 32,767
Default:	Required argument. SCAN_BIT changes the value in the argument.

SCAN_BIT clears the set bit immediately after detecting it.

Related Routines

DIN DIN reads the value of a specified digital input unit into a variable. You can then use SCAN_BIT within a loop to determine which lines were set in the digital input unit at the time it was read in.

SET_BIT SET_BIT specifies the condition of a single bit in a word.

SET_LINE SET_LINE specifies the condition of a single line in a digital output unit.

TEST_BIT TEST_BIT tests the condition of a single bit in a specified word.

TEST_LINE TEST_LINE tests the condition of a single line in a specified digital input unit.

Restrictions

Range of values The word argument can be either integer or real but must be a whole number. If the value of the word argument is outside the integer range, the program halts with an error.

Errors

?MINC-F-Value of argument # exceeds valid range

Examples

No example included.

SCHEDULE

Schedule Program Response to a Time Event

Operation

SCHEDULE designates a service subroutine and schedules it to execute when a specified time event occurs. The time event is either the completion of a specified time interval or the occurrence of a specified time of day. The program continues executing until the time event occurs. (See "Service Subroutines," page 29.)

Configuration

SCHEDULE is a valid statement in any MINC program. It has no configuration requirements because it uses the system clock, not the clock module.

Statement Form

SCHEDULE(mode,time,subroutine)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	ABSOLUTE or INTERVAL	INTERVAL
time	numeric expression or string expression	0 to 86,400 sec 0 to 24:00:00	0
subroutine	numeric expression	0;1 to 32,767	0

Example SCHEDULE('INTERVAL',30,640)
Result Execute the service subroutine which begins at statement 640 after 30 seconds.

Example SCHEDULE('INTERVAL','2::',1100)
Result Execute the service subroutine which begins at statement 1100 after two hours.

Example SCHEDULE('ABSOLUTE',30,640)
Result Execute the service subroutine which begins at statement 640 at 30 seconds after midnight (12:00:30 a.m.).

Example SCHEDULE('ABSOLUTE','16:30:',L)
Result Execute the service subroutine which begins at the statement specified in variable L at 16:30:0 (4:30 in the afternoon).

Example SCHEDULE
Result Cancel the pending request.

Argument Descriptions

mode The character string specifying whether the time event is a time interval or an absolute time of day (see "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
ABSOLUTE	Absolute time of day
INTERVAL	Time interval
Default:	INTERVAL (Time interval)

The longest time interval possible with either mode is 24 hours. In interval mode, you can schedule a time event for 24 hours (86,400 seconds) later. In absolute mode, if the time string represents a time earlier than the current time, SCHEDULE schedules the time event for the next day.

(mode = ABSOLUTE) Absolute time of day is expressed using the 24-hour format, that is, time measured relative to midnight. Five o'clock in the morning is 5 hours past midnight; five o'clock in the afternoon is 17 hours past midnight.

time The time of day or time interval required.

Values	*Meaning*
numeric	0 to 86,400 seconds
string	0 to 24:00:00 (time interval in hours:minutes:seconds format).
Default:	0

(numeric expression) When the time argument is a numeric expression, 0.1 is the shortest interval guaranteed to be precise. SCHEDULE does not consider times shorter than 0.1 to be errors, but these times are not precise. If the time argument is 0, SCHEDULE schedules the time event for the next tick of the system clock. (See "Time Base," page 43.)

(string expression) When the time argument is a string expression, the string has the frame "hours:minutes:seconds". Each of the slots (hours, minutes, seconds) can hold up to two digits, or can be empty (null). The longest time that can be specified is '24:00:00'. The string itself can contain only digits and colons; blanks are invalid in the time string.

If the string contains only one number, SCHEDULE assumes that the number specifies seconds, unless the number is followed by one or two colons. The colons indicate the position of the number in the string frame. For example, the string '8' means 8 seconds, the string '8:' means 8 minutes, and the string '8::'

means 8 hours. Thus, the string ':8:' is equivalent to the string '8:'.

subroutine The statement number of the service subroutine to execute when the time event occurs.

Values	*Meaning*
0	Cancel the scheduled request
1 to 32,767	Statement number of service subroutine
Default:	0 (Cancel the scheduled request.)

SCHEDULE designates the statement number of the beginning of a service subroutine. MINC transfers program control to this statement at the end of the time interval or at the specified time of day. (See "Service Subroutines," page 29.)

Related Routines

PAUSE PAUSE suspends program execution for a specified time interval. After the time interval expires, program execution continues with the statement following the PAUSE statement.

Restrictions

ABSOLUTE With absolute time specifications, if the current time is greater than the time requested, the event is scheduled for the next day. As a result, no event can be scheduled more than 24 hours in advance.

Cancel request A SCHEDULE statement with the subroutine argument defaulted cancels a pending request. The cancellation request executes without error (but does nothing) if no request is pending.

Immediate mode The PAUSE routine does not operate in immediate mode.

Replace request To replace a pending request, you cancel the first request and then schedule the new request.

RESEQ The resequencing command, RESEQ, resequences normal program statement numbers. It does not resequence the service subroutine statement numbers in the SCHEDULE statement. When you resequence a program, you have to determine the new statement number of the service subroutine, and change that number in the SCHEDULE statement.

For this reason, you might find it more convenient to use a varia-

ble name for the subroutine argument. Put the variable assignment statement and an explanatory remark at the beginning of the program where you can locate it quickly. Then, each time you resequence the program, assign new values to the statement number variables, and save searching for all occurrences of the SCHEDULE statement.

SCHEDULE limit SCHEDULE schedules only one time event at a time.

Successive events SCHEDULE schedules only one time event at a time. When the requested time event occurs, the request has been satisfied and there is no longer any request pending. Therefore, to schedule repetitive time events, use a SCHEDULE statement in the service subroutine to reschedule the event.

TIME Set the system time with the TIME command (see TIME, Book 3). If the system time is not the actual time of day, then time interval requests operate correctly but absolute time of day requests operate incorrectly.

Errors

?MINC-F-Could not find service subroutine #### requested

?MINC-F-Invalid or conflicting options requested

The mode designator specified is invalid for SCHEDULE. SCHEDULE permits only one mode designator.

?MINC-F-Previously scheduled event pending. No new request

Only one time event request can be active at a time. You cannot schedule multiple time events.

?MINC-F-Time string format must be hh:mm:ss

?MINC-F-Value of argument # exceeds valid range

The longest valid time interval is 86,400 seconds.

Examples

See Example F, page 67.

SCHMITT

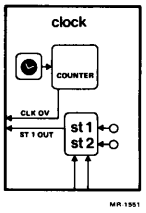
Enable Program Response to a Schmitt Trigger Event

Operation

SCHMITT enables the program to respond immediately to an event occurring on either of the Schmitt triggers. When a signal event occurs on the specified Schmitt trigger, MINC transfers program control to the specified service subroutine in the program. SCHMITT can cancel response requests for the specified trigger. (See "Service Subroutines," page 29.)

Configuration

SCHMITT itself requires only the clock module. The rest of the program and the service subroutine might have their own configuration requirements.



Statement Form

SCHMITT(Schmitt-trigger,subroutine,ST1-count)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
Schmitt-trigger	numeric expression	1 or 2	1
subroutine	numeric expression	0;1 to 32,767	0
ST1-count	numeric expression	1 to 65,535	1

Example SCHMITT(1,650)
Result When a signal occurs on ST1, start executing the service subroutine that begins at statement number 650.

Example SCHMITT(2,S%)
Result When a signal occurs on ST2, start executing the service subroutine whose beginning statement number is in variable S%.

Example SCHMITT(,1000,10)
Result After every tenth signal on ST1, start executing the service subroutine that begins at statement number 1000.

Example SCHMITT
Result Cancel the response request for Schmitt trigger 1.

Argument Descriptions

Schmitt-trigger The Schmitt trigger whose signals the program is to respond to.

Values: 1 or 2

Default: 1

subroutine The number of the first statement in the service subroutine.

Values	*Meaning*
0	Cancel Schmitt trigger response request
1 to 32,767	Statement number of service subroutine

Default: 0 (Cancel the response request)

(subroutine = 0) When no subroutine argument appears, the SCHMITT statement cancels the response request for the specified trigger. That is, the system ignores any further signals on the specified SCHMITT trigger.

(subroutine = 1 to 32,767) SCHMITT designates a service subroutine by the statement number of the first statement in the subroutine. When a signal occurs on the specified Schmitt trigger, MINC transfers program control to the service subroutine as soon as the current statement finishes executing. (See "Program Dynamics for Control Programs," page 29.) After the service subroutine, the program returns to the statement following the one it was executing when the Schmitt trigger signal occurred.

ST1-count The number of signals occurring on Schmitt trigger 1 between transfers to the service subroutine.

Values: 1 to 65,535

Default: 1

The ST1-count argument specifies the number of signals that occur on Schmitt trigger 1 before the service subroutine executes. Therefore, you can have a service subroutine execute periodically rather than in response to every signal.

Schmitt trigger 2 does not allow a count argument. However, you could simulate the ST1-count argument in a service subroutine for ST2 by counting the signals and by ignoring all but the relevant signals. (This procedure would work only if the

Schmitt trigger frequency were low. MINC can execute at most 300 service subroutines per second.)

Related Routines

DIN_EVENT DIN_EVENT enables the system to respond immediately to a signal event on a single digital input line. DIN then collects a single digital input unit value every time a signal occurs on one of the enabled lines. Both DIN_EVENT and SCHMITT allow the program to respond to similar signals. (Connect the signal line either to a line on a digital input unit or to one of the Schmitt triggers on the first clock; see Book 7.)

However, SCHMITT permits a more general response to a signal than do DIN_EVENT and DIN. With DIN_EVENT and DIN, the only response to the signal is to collect the value of the digital input unit. With SCHMITT, the response to the signal is contained in the service subroutine and can be anything required.

Restrictions

Clock 0 only SCHMITT refers to the Schmitt triggers on clock 0. If the system has two clocks installed, SCHMITT refers only to the triggers on the left-most clock.

Immediate mode The SCHMITT routine does not operate in immediate mode.

Multiple events When you enable a Schmitt trigger, it remains active until you cancel it. If a second Schmitt trigger event occurs while the service subroutine for the first is still executing, the program delays responding until the service subroutine finishes. The program can delay responding to at most eight events. If eight event requests are pending, the program halts with an error when the next event occurs.

Multiple triggers Both triggers can be active at the same time. The triggers can be independent or both can use the same service subroutine.

RESEQ The resequencing command, RESEQ, resequences normal program statement numbers. It does not resequence the service subroutine statement number in the SCHMITT statement. When you resequence a program, you have to determine the new statement number of the service subroutine, and change that number in the SCHMITT statement.

For this reason, you may find it more convenient to use a variable name for the subroutine argument. Put the variable assignment statement and an explanatory remark at the beginning of

the program where you can locate it quickly. Then, each time you resequence the program, assign new values to the statement number variables and save searching for all occurrences of the SCHMITT, CONTINUE, and SCHEDULE statements.

Errors

?MINC-F-Could not find service subroutine #### requested

?MINC-F-Schmitt trigger must be 1 or 2

?MINC-F-Schmitt trigger # is already active. No new request

Examples

No example included.

SET_BIT

Change Condition of a Single Bit in a Word

Operation

SET_BIT specifies the condition (set or clear) for a single bit in a numeric variable. It can also set all bits or clear all bits in a variable. (See "Number Systems," page 25.)

Configuration

SET_BIT is valid in any program. It has no configuration requirements.

Statement Form

SET_BIT(bit-position,condition,word)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
bit-position	numeric expression	-1; 0 to 15	required argument
condition	numeric expression	0 or > 0	required argument
word	numeric variable name	-32,768 to 32,767	required argument

Example SET_BIT(0,0,W)
Result Clears bit 0 of variable W.

Example SET_BIT(7,1,V%)
Result Sets bit 7 of variable V%.

Example SET_BIT(-1,1,M)
Result Sets all bits of variable M.

Argument Descriptions

bit-position The position of the bit whose condition is to be changed.

Values	*Meaning*
-1	All bits of the specified word.
0 to 15	The bit position.
Default:	Required argument

See "Bits and Words," page 26.

condition The condition to assign to the bit specified.

Values	*Meaning*
0	Clear the specified bit or bits
> 0	Set the specified bit or bits

Default: Required argument

word The word containing the bit whose condition is being changed.

Values: Whole number in the range -32,768 to 32,767

Default: Required argument. SET_BIT changes the value of the word argument.

Related Routines

AOUT AOUT sends control signals to the D/A converter as part of the output word. Use SET_BIT to set the required control bit values.

BIN The BIN function provides the capability for specifying the conditions of all bits in a word. See BIN, Book 3.

DIN_EVENT, DIN_MASK, and DOUT_MASK DIN_EVENT, DIN_MASK, and DOUT_MASK all require specially constructed mask words. SET_BIT assigns the condition of single bits in a word. You can use SET_BIT in statements which calculate dynamic masks for digital input and digital output sampling.

SCAN_BIT SCAN_BIT tests the conditions of the bits in a specified word starting with bit 0. It returns the bit position of the first bit it finds set and clears the bit in the word.

SET_LINE SET_LINE assigns the condition of a single line in a specified digital output unit.

TEST_BIT TEST_BIT tests the condition of a single bit in a specified word.

TEST_LINE TEST_LINE tests the condition of a single line in a specified digital input unit.

Restrictions

None included

Errors

?MINC-F-Value of argument # exceeds valid range

Examples

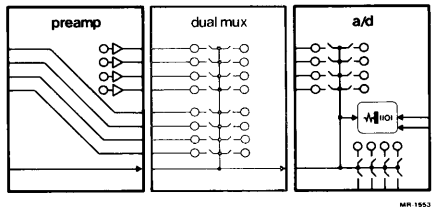
No example included.

SET_GAIN
Set Preamp Gain

Operation

SET_GAIN sets the gain on the preamp module. SET_GAIN sets the specified channels to the specified gains. (See Book 7.)

Configuration



Statement Form

SET_GAIN(mode,gain-code,A/D-channel,no.-of-channels)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	RANDOM	standard mode
gain-code	numeric expression or numeric array	⩾ 0; 0 to 4	1
A/D-channel	integer expression or integer array	8 to 63	required argument
no.-of-channels	numeric expression	1 to 56 or channel array length	1

Example SET_GAIN(,0,8,1)
Result Set channel 8 to autogain.

Example SET_GAIN(,,8,2)
Result Set the gain on channels 8 and 9 to 1.

Example SET_GAIN(,G(),8,8)
Result Set channels 8 through 15 to the gains specified in array G.

Example SET_GAIN(,G(),C(),4)
Result Set the four sequential channels beginning with the channel whose number is in array element C(0) to the gains whose codes are in array G.

Example SET_GAIN('RANDOM',G(3),C(1),4)
Result Set the four channels whose numbers are in array elements C(1), C(2), C(3), and C(4) to the gains whose codes are in array elements G(3), G(4), G(5), and G(6).

Argument Descriptions

mode The character string specifying an operating mode for SET_GAIN. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
RANDOM	Use random mode to set the gain for nonsequential channels. (See "Random Channels," page 42.)
Default:	Standard mode

gain-code The code for the channel gain (see Book 7).

Values	*Meaning*
< 0	No gain specification for the corresponding channel.
0	Autogain (Determines optimal gain for each conversion)
1	Gain = 0.5 (Multiplies input signal by 0.5)
2	Gain = 5 (Multiplies input signal by 5)
3	Gain = 50 (Multiplies input signal by 50)
4	Gain = 500 (Multiplies input signal by 500)
Default:	Gain code = 1. That is, input gain is 0.5.

The gain code can be either a numeric expression or a numeric array. If the gain code is a numeric expression, all of the channels specified are set to the same gain. If the gain-code argument is an array and the no.-of-channels argument is greater than 1, SET_GAIN sets each channel to the gain code specified in the corresponding array element. That is, the first channel specified is set to the first gain specified, the second channel specified is set to the second gain specified, and so on for as many channels as are required.

(gain-code = autogain) You can specify a wide dynamic range for analog sampling with a procedure called *autogain*. With autogain, the input transfer routine determines the gain to use for each conversion. It does this by actually performing two conversions for each data value. (Therefore, the maximum sampling rate with autogain is less than for fixed gain.)

The first conversion, called the *ranging conversion*, always occurs at minimum gain (0.5). The transfer routine uses the result of this conversion to select the gain for the second conversion. The second conversion thus uses the highest gain possible for the current input signal.

A/D-channel The A/D channel whose gain is to be set.

Values: 8 through 63

Default: Required argument

The A/D-channel argument can be either an integer expression or an integer array. If it is an integer expression, SET_GAIN sets the gain for that channel and for as many higher-numbered channels as are specified by the no.-of-channels argument. If it is an array and the RANDOM designator is specified, SET_GAIN sets the gains for the channels specified in the array. (See "Random Channels," page 42.)

The valid channel numbers depend on which of the analog modules are installed in the system, and on how the instruments are connected to the A/D channels (see Book 7).

no.-of-channels The number of channels to be set.

Values: 1 to the maximum number of channels installed (for sequential channels).

1 to the length of the channel array (for random mode).

Default: 1

If the A/D-channel argument is a single expression, then SET_GAIN sets the specified number of channels, beginning with the channel specified. For example, if the A/D-channel argument is 4, and the number requested is 3, then SET_GAIN sets the gain for channels 4, 5, and 6. If the A/D-channel argument is an array name and the RANDOM designator is specified, then SET_GAIN sets the gain for the specified number of channels, beginning with the first one specified in the array. For example, suppose the array contains the elements 6, 2, 5, 0, and 1, and the number requested is 4. Then SET_GAIN sets the gain for channels 6, 2, 5, and 0.

Related Routines

AIN, AIN_HIST, and AIN_SUM AIN, AIN_HIST, and AIN_SUM collect data from A/D channels.

TEST_GAIN TEST_GAIN tests the condition of any A/D channel. Use TEST_GAIN to determine whether the preamp is actually connected and has been set to programmable mode.

Restrictions

Configuration All of the channels whose gain is to be changed must be connected to a preamp module.

Negative gain-code If the gain-code argument is negative, SET_GAIN does not change the gain for the corresponding

channel. This is needed for situations in which you want to use a channel array in which some of the channels in the array are not connected to a preamp module.

Programmable mode The front panel control on the preamp module must be set to programmable mode (P setting) for each channel whose gain is to be set. Otherwise, the program halts with an error.

Errors

?MINC-F-Cannot set gain with no preamp connected to channel #

?MINC-F-Channel or unit # not in system for routine

?MINC-F-No. of channels must match size of gain array

?MINC-F-Set front panel switch to P mode for channel # requested

?MINC-F-Value of argument # exceeds valid range

The gain code must be negative or a value between 0 an 4 (inclusive).

Examples

See Example A, page 56.

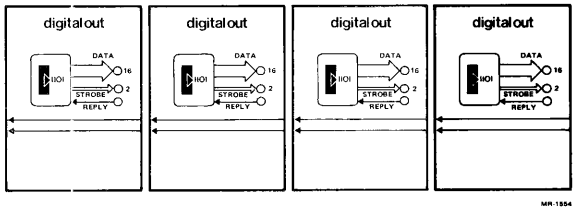
SET_LINE

Change Condition of Digital Output Line

Operation

SET_LINE specifies the condition (set or clear) of a single digital output line on a specified digital output unit. It can also set all lines or clear all lines of the unit. (See "Digital Sampling and Control," page 3.)

Configuration



Statement Form

SET_LINE(line-number,condition,unit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
line-number	numeric expression	-1; 0 to 15	-1
condition	numeric expression	0 or > 0	0
unit	numeric expression	0 to 3	0

Example SET_LINE
Result Clear all lines of digital output unit 0.

Example SET_LINE(,1)
Result Set all lines of digital output unit 0.

Example SET_LINE(0,,1)
Result Clear line 0 of digital output unit 1.

Example SET_LINE(1)
Result Clear line 1 of digital output unit 0.

Argument Descriptions

line number The digital output line whose condition is being specified.

Values	*Meaning*
-1	All lines of the specified unit.
0 to 15	The single digital output line.

Default: -1

The line numbers appear on the connector-block labels as terminals D00 through D15 (for lines 0 through 15, respectively).

condition The specified condition for the line, or lines.

Values	*Meaning*
0	Clear the specified line, or lines.
> 0	Set the specified line, or lines.
Default:	0 (Clear the lines)

unit The digital output unit containing the line to be set or cleared.

Values:	0 to 3
Default:	0

Related Routines

DOUT DOUT can set the conditions of all lines of a digital output unit. DOUT is more useful for situations when you need to specify different conditions for different lines. However, SET_LINE is more convenient for changing the condition of a single line.

Restrictions

None

Errors

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Value of argument # exceeds valid range

Examples

See Example B, page 58, and Example D, page 62.

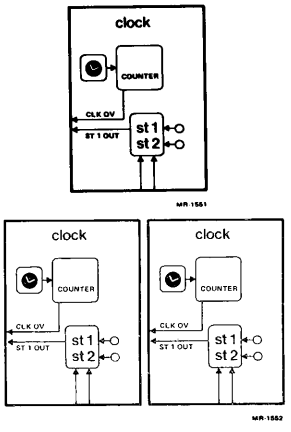
START_TIME

Start the Elapsed-Time Counter

Operation

START_TIME starts the elapsed-time counter with a specified rate or stops the elapsed-time counter. (See "Measuring Time Intervals," page 3.)

Configuration



Statement Form

START_TIME(rate)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
rate	string expression	CHZ,EXTERNAL, HALT,KHZ,LINE,ST2	HALT

Example START_TIME('CHZ,ST2')
Result Set the clock module rate to 100 Hz, and start the elapsed-time counter when a signal occurs on ST2.

Example START_TIME('LINE')
Result Set the clock module to line frequency, and start the elapsed-time counter immediately.

Example START_TIME('HALT')
Result Stop the elapsed-time counter immediately.

Argument Descriptions

rate The character string specifying the operating mode for the elapsed-time counter. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
CHZ	Set the elapsed-time counter rate to 100 Hz.

Values	*Meaning*
EXTERNAL	Use signals from external source connected to ST1 as the time base for the elapsed-time counter.
HALT	Halt the clock.
KHZ	Set the elapsed-time counter to 1000 Hz.
LINE	Use line frequency (50 to 60 Hz) as the time base for the elapsed-time counter.
ST2	Start the elapsed-time counter when a signal occurs on ST2.
Default:	HALT

The rate string can contain more than one mode designator. However, many combinations of modes are invalid. Essentially, any rate string that specifies more than one rate designator is invalid. The ST2 mode designator is invalid alone and must be paired with an explicit rate. In the following table, "X" marks the invalid mode combinations.

	CHZ	EXTERNAL	HALT	KHZ	LINE	ST2
CHZ	X	X	X	X	X	
EXTERNAL	X	X	X	X	X	
HALT	X	X	X	X	X	X
KHZ	X	X	X	X	X	
LINE	X	X	X	X	X	
ST2			X			X

MR-1949

DIN DIN permits timestamp mode. In timestamp mode, every DIN sample consists of two values, the value of the elapsed-time counter (when it happened) and the value of the digital input unit (what happened). Use START_TIME to start the elapsed-time counter before using DIN in timestamp mode.

Related Routines

GET_TIME GET_TIME reads the current value of the elapsed-time counter.

MAKE_TIME MAKE_TIME converts DIN-format elapsed-time counter values to standard numeric values. (See "Number Systems," page 25, and "Format Conversion," page 25.)

Restrictions

Counter rate START_TIME increments the elapsed-time counter value every time a time base signal occurs. When the time base has a regular rate (for example, with the line frequency time base), the elapsed-time count corresponds directly to time. When the time base can be varying (for example, with the time base supplied on ST1), the elapsed-time count represents the number of timebase signals received.

With an external time base (external mode), the maximum signal rate is 1 MHz.

Immediate mode The START_TIME routine does not operate in immediate mode.

Initial value START_TIME sets the initial value of the elapsed-time counter to zero. You cannot select any other initial value, or reset the value while the elapsed-time counter is running.

ST2 start The ST2 mode designator specifies when to start the elapsed-time counter but it does not specify a rate. You must include an explicit rate designator.

Which clock START_TIME uses clock 0 if only one clock is present in the system. In this case, if START_TIME is using clock 0, you cannot request any clock-controlled data transfers. If the system contains two clocks (clock 0 and clock 1), START_TIME uses clock 1 and any data transfer routine uses clock 0.

Errors

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for START_TIME.

The START_TIME statement specifies incompatible mode designators. Refer to the table of valid mode designator combinations.

Examples

See Example E, page 65.

TERMINATE

Stop Continuous Data Transfer

Operation

TERMINATE stops a specified continuous data transfer, either immediately or after transfer of the current array partition is complete. TERMINATE can also stop all of the continuous mode transfers in progress. (See "Continuous Data Transfers," page 33.)

Configuration

The configuration depends on the kind of data transfer in progress.

Statement Form

TERMINATE(mode,data-name)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	DEFER or IMMEDIATE	DEFER
data-name	numeric array	transfer array	all transfers

Example TERMINATE('DEFER',D%())
Result Stop the data transfer involving array D% as soon as the current array partition is ready for processing.

Example TERMINATE(,A())
Result Stop the data transfer involving array A as soon as the current array partition is ready for processing.

Example TERMINATE('IMMEDIATE',V())
Result Stop the data transfer involving array V immediately.

Example TERMINATE
Result Stop all continuous transfers after each active transfer has finished transferring its current array partition.

Argument Descriptions

mode The mode code specifying when to stop the continuous transfer.

Values	*Meaning*
DEFER	Stop the specified transfer when the transfer of the current array partition is complete.
IMMEDIATE	Stop the specified transfer immediately.

Default: DEFER

data-name The name of the data array being used for the continuous transfer that is to be stopped.

Values: Name of a numeric array.

Default: All arrays currently involved in continuous transfers.

Related Routines

AIN, DIN, AOUT, and DOUT The data transfer routines AIN, DIN, AOUT, and DOUT can operate in continuous mode. The program must use a TERMINATE statement to stop continuous mode transfers.

CONTINUE, WAIT_FOR_DATA The CONTINUE and WAIT_FOR_DATA routines manage continuous mode data transfers. With CONTINUE management, the TERMINATE statement must appear in the service subroutine. With WAIT_FOR_DATA management, the end-of-transfer test and the TERMINATE statement must appear with the statements processing the array partitions. (See "Continuous Data Transfers," page 33, CONTINUE, and WAIT_FOR_DATA.)

Restrictions

Array index As part of the termination process with the DEFER designator, TERMINATE assigns a negative value to the array partition index. The statements that process the array partition (service subroutine or WAIT_FOR_DATA service statements) should include a test of the index variable. If the index variable is negative, then the transfer is complete. The statements which process the array partition must not execute the transfer management statement again.

IMMEDIATE With immediate termination, TERMINATE stops the transfer process and immediately removes all traces of the relationship between the array and the transfer. You must design the program carefully so that it cannot attempt to execute CONTINUE or WAIT_FOR_DATA again. If a transfer management routine does execute after an immediate termination, the program halts with an error.

Immediate mode The TERMINATE routine does not operate in immediate mode.

Normal stopping The TERMINATE statement stops only continuous mode transfers. It does not stop program execution. The portion of the program following the TERMINATE statement

continues to execute normally.

Timing collision In defer mode, TERMINATE always stops the transfer process at the end of the next partition. Under *very* unusual conditions, the TERMINATE('DEFER') statement can execute at exactly the same time as one partition fills and the next partition becomes the current one. This highly unlikely situation is a *timing collision.* It can happen only when the service subroutine or the WAIT_FOR_DATA service statements require exactly as long to execute as the array partition requires to fill.

The result of the timing collision is that you transfer and process one more partition of values than you expected. In cases where transferring an extra partition could cause a serious problem, you can try to design your program to avoid the problem. For example, you could count the number of partitions to be transferred. You could assign a special flag value to a variable when the transfer is complete and then test that variable to see whether or not to process another partition.

Transfer errors If a continuous transfer encounters any problems, the management routine terminates the transfer with an error message indicating the problem.

Errors

?MINC-F-Continuous transfer not in progress for array specified.

?MINC-F-Invalid or conflicting options requested

One of the mode designators specified is invalid for TERMINATE.

TERMINATE permits only one mode designator.

Examples

See Example C, page 6.

TEST_BIT

Test Condition of a Single Bit in a Word

Operation

TEST_BIT tests the condition of specific bit in a specified word. It can also test in general whether at least one of the bits is set (but does not report which one). (See "Bits and Words," page 26.)

Configuration

TEST_BIT is a valid statement in any MINC program. It has no configuration requirements.

Statement Form

TEST_BIT(bit-position,condition,word)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
bit-position	numeric expression	-1; 0 to 15	-1
condition	numeric variable name	0 or > 0	required argument
word	numeric variable	-32,768 to 32,767	required argument

Example TEST_BIT(15,C,W)
Result Test the condition of bit 15 in word W and assign the condition of bit 15 to variable C.

Example TEST_BIT(-1,C,W1)
Result Test the conditions of all bits in variable W1 and assign a positive value to variable C if one or more of the bits in W1 is set.

Example TEST_BIT(0,C,V%)
Result Test the condition of bit 0 in variable V%, and assign the condition of bit 0 to variable C.

Argument Descriptions

bit-position The position of the bit whose condition is to be tested.

Values	*Meaning*
-1	Test all of the bits
0 to 15	Test the single bit specified

Default: -1

See "Bits and Words," page 26.

condition The condition (clear or set) of the bit being tested.

Values	*Meaning*
0	The bit was clear.
> 0	The bit was set.
Default:	Required argument. TEST_BIT assigns a value to the condition argument.

word The numeric value to be tested.

Values:	Whole number in the range -32,768 to 32,767
Default:	Required argument

Related Routines

DIN DIN reads the value of a specified digital input unit into a specified variable. You can then use TEST_BIT to determine the condition of any set of lines in the unit because bit position in the word corresponds to line number in the unit.

SCAN_BIT SCAN_BIT examines the condition of bits in a specified word, starting with bit 0, reports the position of the first bit it finds set, and clears the bit.

SET_BIT SET_BIT specifies the condition of a single bit in a word.

SET_LINE SET_LINE specifies the condition of a single line in a digital output unit.

TEST_LINE TEST_LINE tests the current condition of a single line in a specified digital input unit.

Restrictions

Bits and lines When DIN collects a value from a digital input unit, it puts the value into a variable. Bit positions in the word correspond to line numbers in the unit. Remember that the word contains the condition of the lines at the point in time when DIN read the value from the digital input unit. At any time later, the conditions of the bits are not necessarily the same as the current conditions of the lines, because the arrival of external signals could have changed the conditions of the lines.

Errors

?MINC-F-Value of argument # exceeds valid range

Examples

No example included.

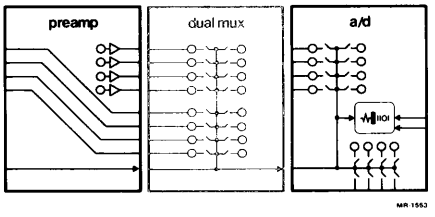
TEST_GAIN

Check Gain and Mode of Preamp

Operation

TEST_GAIN obtains the current status of the specified A/D channel. The channel status consists of the gain and the operating mode.

Configuration



Statement Form

TEST_GAIN(A/D-channel,gain-code,mode-code)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
A/D-channel	numeric expression	0 through 63	0
gain-code	real variable name	-3 to 4	required argument
mode-code	string variable name	*,V,I,R	required argument

Example TEST_GAIN(8,G,M$)
Result Determine the current status of channel 8. Report the current gain in variable G and the current mode in variable M$.

Example TEST_GAIN(C,G,M$)
Result Determine the current status of channel C. Report the current gain in variable G and the current mode in variable M$.

Argument Descriptions

A/D-channel The A/D-channel whose status is being checked.

Values: 0 to the highest-numbered channel in the system

Default: 0

The valid values depend on which analog modules are installed in the system, and on how the instruments are connected to the A/D channels. (See Book 7.)

gain-code The gain currently selected for the channel.

Values	*Meaning*
-3	Specified channel is not in system

-2	No preamp connected to the channel
-1	Gain control on the front panel is not set to programmable (P) setting
0	Autogain selected (see SET_GAIN)
1	Gain = 0.5 (Multiplies input signal by 0.5)
2	Gain = 5 (Multiplies input signal by 5)
3	Gain = 50 (Multiplies input signal by 50)
4	Gain = 500 (Multiplies input signal by 500)

Default: Required argument. TEST_GAIN assigns a value to the argument.

mode-code The operating mode currently selected by the front panel switches.

Values	*Meaning*
*	Channel not connected to preamp
V	Channel set to measure voltage
I	Channel set to measure current
R	Channel set to measure resistance

Default: Required argument. TEST_GAIN assigns a value to the argument.

TEST_GAIN assigns a one-character string to the string variable specified.

Related Routines

AIN, AIN_HIST, and AIN_SUM AIN, AIN_HIST and AIN_SUM collect data from A/D channels.

SET_GAIN SET_GAIN sets the gain for any A/D channel which is set for program control.

Restrictions

P setting TEST_GAIN requires that the channel be set for program control, with the preamp front panel switch set to P (see Book 7).

Preamp needed TEST_GAIN returns actual status information only if the A/D channel is connected to a preamp module. Without a preamp connection, the program cannot test or set the gain and the channel operates with a fixed gain of 1 in voltage measuring a mode.

Errors

?MINC-F-Use array element instead of array for argument #

?MINC-F-No workspace available for the string specified

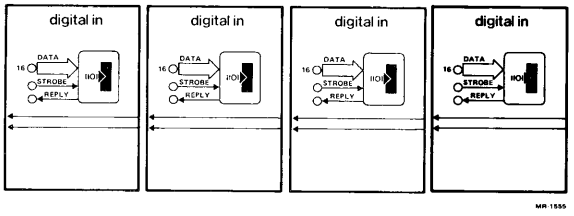
Examples See Example A, page 56.

TEST_LINE
Test Condition of Digital Input Line

Operation

TEST_LINE tests the condition of one line of a digital input unit. (See "Digital Sampling and Control," page 3.)

Configuration



Statement Form

TEST_LINE(line-number,line-condition,unit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
line-number	numeric expression	-1; 0 to 15	-1
line-condition	numeric variable name	0 or > 0	required argument
unit	numeric expression	0 to 3	0

Example TEST_LINE(,C%,1)
Result Test all lines of digital input unit 1 and assign a value greater than 0 to variable C% if one or more lines is set.

Example TEST_LINE(7,C,3)
Result Test line 7 of digital input unit 3 and put the line condition code in variable C.

Example TEST_LINE(0,C%)
Result Test line 0 of digital input unit 0 and put the line condition code in variable C%.

Argument Descriptions

line-number The number of the digital input line to be tested.

Values	*Meaning*
-1	Test whether any of the lines are set
0 to 15	Test the digital input line specified
Default:	-1

The line numbers appear on the connector-block labels as terminals D00 through D15 (lines 0 through 15, respectively).

line-condition The condition (set or clear) of the line being tested.

Values	*Meaning*
0	The line was clear when tested.
> 0	The line was set when tested.
Default:	Required argument. TEST_LINE assigns a value to the argument.

unit The digital input unit containing the line to be tested.

Values: 0 to 3

Default: 0

Related Routines

SCAN_BIT SCAN_BIT examines the conditions of bits in a data word starting with bit 0, reports the positions of the first bit it finds set, and clears the bit.

SET_BIT SET_BIT specifies the condition (set or clear) of a single bit in a word.

SET_LINE SET_LINE specifies the condition (set or clear) of a single line in a digital output unit.

TEST_BIT TEST_BIT tests the condition (set or clear) of a single bit in a word.

Restrictions

None

Errors

?MINC-F-Channel or unit # not in system for the routine

?MINC-F-Value of argument # exceeds valid range

The line-number argument must be in the range -1 to 15.

Examples

See Example B, page 58, and Example D, page 62.

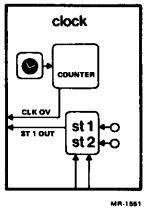
TIME_HIST

Generate Histogram of Time Interval Data

Operation

TIME_HIST measures the time intervals between successive signals on ST2, and generates a frequency histogram using the time interval values. The time intervals are specified in clock tick units. (See "Frequency Histograms," page 45.)

Configuration



Statement Form

TIME_HIST(mode,histogram-name,tick-rate,sweep-length, lower-endpoint,upper-endpoint)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
mode	string expression	DISPLAY,ZERO	standard mode
histogram-name	integer array name	0 to 32,767	required argument
tick-rate	numeric expression	2 to 6	required argument
sweep-length	numeric expression	1 to 32,767	required argument
lower-endpoint	numeric expression	0 to 65,535 ticks	0
upper-endpoint	numeric expression	0 to 65,535 ticks	65,535

Example TIME_HIST(,H%(),2,100)
Result Collect a sweep of 100 points with each clock tick measuring 1/100 second (100 Hz). Generate the frequency histogram using the full time range, and store the histogram in array H%.

Example TIME_HIST('DISPLAY',H%(),5,1 E6,2,2 E8)
Result Collect a sweep of 1,000,000 points (1 E6 points) with the clock running at 10,000 Hz. Generate and display the histogram for time intervals in the range of interest 2 to 256, and store the histogram in array H%.

Argument Descriptions

mode The character string selecting an optional mode for TIME_HIST. (See "Operating Modes and Mode Designators," page 23.)

Values	*Meaning*
DISPLAY	Display the generated histogram on the screen.

ZERO	Set all elements of the histogram array to zero before starting.
Default:	Standard mode

Combining the DISPLAY and ZERO designators is valid.

(mode = DISPLAY) In display mode, TIME_HIST displays on the screen the contents of up to 512 elements of the histogram array. Each element contains the count for the corresponding bin of the histogram. That is, the value shown on the screen for a bin represents the number of occurrences of time intervals within that bin range.

TIME_HIST clears the screen before beginning the display.

The display includes a horizontal zero axis, but no vertical axis. The count data are displayed so that the full vertical scale of the screen represents the largest count value. No units are shown. Therefore, the display shows the relative counts in the histogram bins, but not the absolute counts.

TIME_HIST produces a display suitable for monitoring purposes. Use the specialized graphic routines for final, labeled displays (see Book 4).

histogram-name The name of the integer histogram array.

Values:	0 to 32,767 counts
Default:	Required argument. TIME_HIST assigns values to the histogram array.

TIME_HIST uses the full length of the array to store the histogram. The number of elements in the array determines the number of bins in the histogram. (See "Frequency Histograms," page 45.) The histogram array must contain at least four elements.

TIME_HIST reserves the first and last elements of the array as overflow bins to count the occurrences of intervals outside the range of interest. That is, if TIME_HIST receives an interval shorter than the specified minimum, it increments the first element in the array. If it receives an interval longer than the specified maximum, it increments the last element in the array. The remaining array elements contain the counts for intervals in the corresponding bin ranges.

tick-rate The time base for measuring time intervals

Values	*Time Base*
2	100 Hz
3	1 KHz (1000 Hz)
4	10 KHZ (10,000 Hz)
5	100 KHZ (100,000 Hz)
6	1 MHz (1,000,000 Hz)

Default: Required argument

The values for tick rates are the powers of ten for the rates expressed in Hertz. For example, a rate of 5 specifies 10 to the power 5, that is, 100,000 Hz.

sweep-length The number of time intervals to be measured.

Values: 1 to 32,767

Default: Required argument

TIME_HIST measures the intervals between successive signals on ST2. Therefore, the total number of ST2 signals occurring must be one greater than the number of intervals to be measured because the first ST2 signal defines only the beginning of the first interval. All other signals define both the end of the preceding interval, and the beginning of the next.

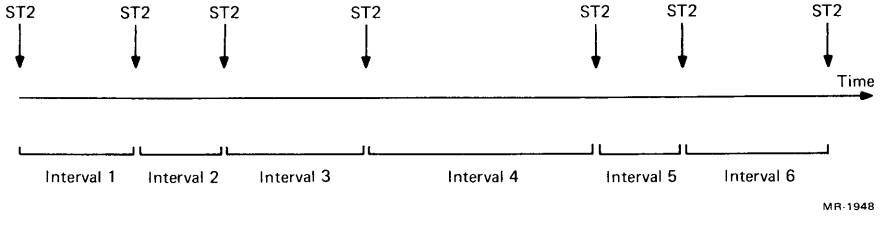


Figure 47. Defining Intervals for TIME_HIST.

lower-endpoint Lower-endpoint of the range of interest (shortest time interval of interest).

Values: 0 to 65,535 ticks

Default: 0 ticks

The lower-endpoint argument defines the lower endpoint of the range of interest. Any time intervals shorter than the minimum increase the frequency in the first element of the histogram array. If the minimum interval is zero, then the first element of the

histogram array is always empty (because, by definition, nothing can be out of range).

upper-endpoint Upper-endpoint of the range of interest (longest time interval of interest).

Values: 0 to 65,535 ticks

Default: 65,535 ticks

The upper-endpoint argument defines the upper endpoint of the range of interest. Any time intervals longer than the maximum increase the frequency in the last element of the histogram array. If the maximum interval is 65,535, then the last element of the histogram array contains the number of intervals which exceeded the capacity of the clock (65,535 ticks).

Related Routines

PST_HIST PST_HIST generates a post-stimulus time frequency histogram. PST_HIST measures the time intervals between an initial (stimulus) signal on ST1, and subsequent (response) signals on ST2. PST_HIST measures the time elapsed since the initial signal, unlike TIME_HIST which measures the time elapsed since the most recent ST2 signal. PST_HIST collects multiple sweeps of time interval data (one sweep for each stimulus signal) and generates a frequency histogram using all the time interval data.

Restrictions

DISPLAY TIME_HIST displays the histogram array continuously so that you can monitor the progress of the data acquisition. However, the display rate differs from the acquisition rate. Thus, the screen display can be difficult to interpret because the full vertical scale of the display is based on the largest count in the histogram at the beginning of the display.

Errors

?MINC-F-Clock too fast for system to respond

?MINC-F-Existing display conflicts with display requested

A prior AIN statement specified continuous display mode. TIME_HIST cannot erase this display.

?MINC-F-Histogram arrays must contain at least 4 elements

?MINC-F-Value of argument # exceeds valid range

Examples

See Example G, page 70.

WAIT_FOR_DATA

Wait for Complete Array Partition

Operation

WAIT_FOR_DATA manages data transfer and program execution in continuous mode transfers. WAIT_FOR_DATA holds program execution until the current array partition is ready for processing. When the partition is ready, program execution resumes with the statement following the WAIT_FOR_DATA statement. (See "Continuous Data Transfers," page 33.)

Configuration

The configuration depends on which data transfer routine is associated with the WAIT_FOR_DATA statement.

Statement Form

WAIT_FOR_DATA (data-name,index)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
data-name	numeric array	transfer array	required argument
index	numeric variable name	0,(n+1)/2	required argument

Example WAIT_FOR_DATA(S(),I%)
Result Wait until the next partition of array S is ready for processing. The array index of the partition is in variable I%.

Example WAIT_FOR_DATA(V%(),I)
Result Wait until the next partition of array V% is ready for processing. The array index of the partition is in variable I.

Argument Descriptions

data-name The data array being used for the continuous data transfer.

Values: Name of numeric array.

Default: Required argument

WAIT_FOR_DATA itself allows any numeric array name. The data transfer routine might have some restriction on the data type allowed for the array.

The data-name array must already have appeared in a data transfer statement specifying continuous mode. If no transfer is in progress for the data-name array, then WAIT_FOR_DATA halts with an error.

index The variable name containing the array index for the

next partition to be processed by the program.

Values: 0 or (n+1)/2 (where n is the array dimension for data-name)

Default: Required argument. WAIT_FOR_DATA assigns the value of the index argument.

WAIT_FOR_DATA changes the value of the index variable so that it always contains the index of the first element in the next available array partition. If continuous output is in progress, the index points to the next array partition to be filled with data for output. If continuous input is in progress, the index points to the next array partition containing valid data to be stored.

Related Routines

AIN, DIN The input transfer routines AIN and DIN can operate in continuous mode. WAIT_FOR_DATA holds program execution at the WAIT_FOR_DATA statement until the current array partition is full of input data. Then, MINC continues program execution with the statement following the WAIT_FOR_DATA statement.

AOUT, DOUT The output transfer routines AOUT and DOUT can operate in continuous mode. WAIT_FOR_DATA holds program execution at the WAIT_FOR_DATA statement until all data in the current partition have been output. Then MINC continues program execution with the statement following the WAIT_FOR_DATA statement.

CONTINUE CONTINUE allows the program to execute during the data transfer. (See "Continuous Data Transfers," page 33, and CONTINUE.) CONTINUE transfers control to a service subroutine whenever the array partition involved in the data transfer is ready for processing.

TERMINATE TERMINATE stops a specified continuous data transfer. The statements processing the array partition must test for some condition which defines the end of the transfer and then transfer control to a TERMINATE statement if the transfer is complete.

Restrictions

Immediate mode The WAIT_FOR_DATA routine does not operate in immediate mode.

Index argument The index argument must be a variable name. The portions of the program which process the array partition must not use the index variable for any purpose other than the array index.

Output transfers The first time WAIT_FOR_DATA executes for an output transfer, it immediately passes control to the partition processing statements which follow it. Those statements then prepare the first partition for the transfer and must execute WAIT_FOR_DATA again. The output transfer then starts at this point (see Figure 17).

Starting transfers The program must call WAIT_FOR_DATA before the data transfer routine can start transferring data. Even when the data transfer statement specifies an external start signal, the routine does not start transferring data when the signal occurs unless the WAIT_FOR_DATA statement has been executed. The program ignores any external signals that occur before the WAIT_FOR_DATA statement executes. The WAIT_FOR_DATA statement should logically follow the data transfer statement. The program statements following the WAIT_FOR_DATA statement must process the array partitions involved in the transfer.

Errors

?MINC-F-Clock too fast for system to respond

The continuous transfer associated with this WAIT_FOR_DATA statement has requested trigger intervals that are too short.

?MINC-F-Continuous transfer not in progress for array specified

?MINC-F-Data lost—transfer rate too high

The continuous transfer associated with this WAIT_FOR_DATA statement cannot process the array partitions quickly enough for the triggering rate.

?MINC-F-Value exceeds valid range for argument

The continuous transfer associated with this WAIT_FOR_DATA statement tried to transfer an invalid data value.

Examples

See Example F, page 67.

This program fragment prompts the person running the program to enter some essential information about the session. This is good programming practice for several reasons. It allows the program to be reused for many sessions, without the danger of losing data. The person running the program (who might not have written it) does not have to remember what information to provide, or in what order to enter it.

Middle The middle of a control program contains the statements for the measurement loop(s), for output transfers, and for any computations required during the process. The actual design of the loops depends on your configuration and application, so it is impossible to state detailed guidelines.

Some general guidelines are possible:

1. The time relationships between events are the most important and difficult aspects of the process to control. Verify that the program in fact meets the timing requirements of the process. Programs containing the wrong timing relationships often execute successfully, providing results misleadingly similar to the correct results.

2. The endpoints of the measurement process are more difficult to control than the middle. Pay close attention to the mechanism for starting measurements or transfers (and the time relationships involved) and to the mechanism for determining the end of the process and terminating it in an orderly way.

3. Unlike computational programs, the control program statements themselves convey little information about what they control or how they do it. Document the time and event relationships carefully and completely, both in program REM statements and in written records (like the time line sketches and instrumentation records in Book 7).

4. The lab module routines report on those erroneous conditions they can detect (see "Error Detection," page 51). However, the routines can detect only well-defined, general error conditions. You must be alert to the specialized error conditions that could occur in your application and monitor the process and results closely throughout to avoid invalid program runs.

End The end of a control program contains the procedures for terminating the session. The program end must halt any data

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____ Telephone_____

City_____ State_____ Zip Code_____
or
Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MR1-2/E37
MARLBOROUGH, MASSACHUSETTS 01752

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line