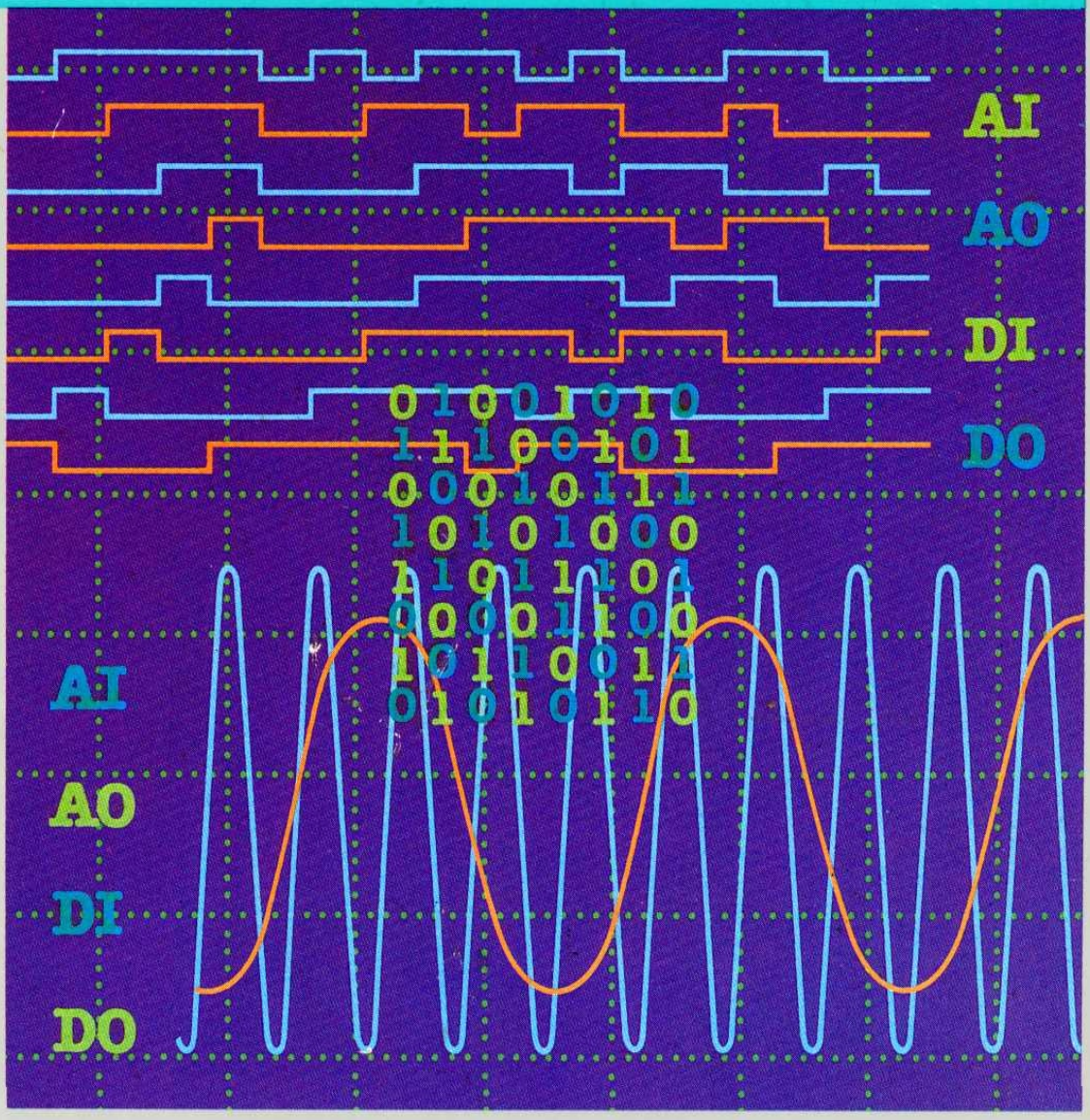
Program Interface Series

IBM Personal Computer Data Acquisition and Control Adapter Programming Support

Engineering/Scientific Family



IBM

Personal Computer Software

Provides application development support for the IBM Personal Computer Data Acquisition and Control Adapter. Automatically controls and monitors electronic sensor-based systems.

Guide to the Data Acquisition and Control Adapter Programming Support.

Program Interface Series

IBM Personal Computer Data Acquisition and Control Adapter Programming Support

Provides application development support for the IBM Personal Computer Data Acquisition and Control Adapter Automatically controls and monitors electronic sensor-based systems.

Engineering/Scientific Family

Boca Raton, Florida 33432

International Business Machines Corporation
IBM Program License Agreement

You should carefully read the following terms and conditions before opening this diskette package Opening this diskette package indicates your acceptance of these terms and conditions If you do not agree with them, you should promptly return the package unopened; and your money will be refunded.

IBM provides this program and licenses its use in the United States and Puerto Rico. You assume responsibility for the selection of the program to achieve your intended results, and for the installation, use and results obtained from the program.

License

You may

a. Use the program on a single machine.
b. copy the program into any machine readable or printed form for backup or modification purposes in support of your use of the program on the single machine (Certain programs, however, may include mechanisms to limit or inhibit copying. They are marked "copy protected "),
c. modify the program and/or merge it into another program for your use on the single machine (Any portion of this program merged into another program will continue to be subject to the terms and conditions of this Agreement.), and,

d. transfer the program and license to another party if the other party agrees to accept the terms and conditions of this Agreement If you transfer the program, you must at the same time either transfer all copies whether in printed or machine-readable form to the same party or destroy any copies not transferred; this includes all modifications and portions of the program contained or merged into other programs You must reproduce and include the copyright notice on any copy, modification or portion merged into another program.

You may not use, copy, modify, or transfer the program, or any copy, modification or merged portion, in whole or in part, except as expressly provided for in this license If you transfer possession of any copy, modification or merged portion of the program to another party, your license is automatically terminated.

Term

The license is effective until terminated You may terminate it at any other time by destroying the program together with all copies, modifications and merged portions in any form It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement You agree upon such termination to destroy the program together with all copies, modifications and merged portions in any form.

Limited Warranty

The program is provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you (and not IBM or an authorized Personal Computer dealer) assume the entire cost of all necessary servicing, repair or correction.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you. This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassettes on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

Limitations of Remedies

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette or cassette not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized **IBM Personal Computer** dealer with a copy of your receipt.

or

2. if IBM or the dealer is unable to deliver a replacement diskette or cassette which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

In no event will IBM he liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use such program even if IBM or an authorized IBM Personal Computer dealer has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

General

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use such program even if IBM or an authorized IBM Persona! Computer dealer has been advised of the possibility of such damages, or for any claim by any other party
Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

General

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

International Business Machines Corporation
P.O. Box 1328-S
Boca Raton, Florida 33432

Printed in the
United States of America

6024202

Engineering/Scientific Family

IBM Personal Computer Data Acquisition and Control Adapter Programming Support

Software required:

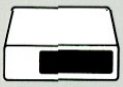
DOS 2.00 or higher for IBM PC and PC XT
DOS 2.10 or higher for IBM *Portable* PC
DOS 3.00 or higher for IBM Personal Computer AT

Software included:

One double-sided diskette

System requirements:

IBM display



IBM PC, PC XT, *Portable* PC or Personal Computer AT

64KB of memory (minimum)

One double-sided diskette drive

IBM Data Acquisition and Control Adapter

International Business Machines Corporation
P.O. Box 1328-S
Boca Raton, Florida 33432

Printed in the United States of America

6024202

International Business Machines Corporation **Boca Raton, Florida 33432**
IBM Program License Agreement

You should carefully read the following terms and conditions before opening this diskette package. Opening this diskette package indicates your acceptance of these terms and conditions. If you do not agree with them, you should promptly return the package unopened; and your money will be refunded.

IBM provides this program and licenses its use in the United States and Puerto Rico. You assume responsibility for the selection of the program to achieve your intended results, and for the installation, use and results obtained from the program.

License

You may:

a. use the program on a single machine;

b. copy the program into any machine readable or printed form for backup or modification purposes in support of your use of the program on the single machine (Certain programs, however, may include mechanisms to limit or inhibit copying. They are marked "copy protected.");

c. modify the program and/or merge it into another program for your use on the single machine (Any portion of this program merged into another program will continue to be subject to the terms and conditions of this Agreement.); and,

d. transfer the program and license to another party if the other party agrees to accept the terms and conditions of this Agreement. If you transfer the program, you must at the same time either transfer all copies whether in printed or machine-readable form to the same party or destroy any copies not transferred; this includes all modifications and portions of the program contained or merged into other programs.

You must reproduce and include the copyright notice on any copy, modification or portion merged into another program.

You may not use, copy, modify, or transfer the program, or any copy, modification or merged portion, in whole or in part, except as expressly provided for in this license.

If you transfer possession of any copy, modification or merged portion of the program to another party, your license is automatically terminated.

Term

The license is effective until terminated. You may terminate it at any other time by destroying the program together with all copies, modifications and merged portions in any form. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the program together with all copies, modifications and merged portions in any form.

Limited Warranty

The program is provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you (and not IBM or an authorized Personal Computer dealer) assume the entire cost of all necessary servicing, repair or correction.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you. This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassettes on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

Limitations of Remedies

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette or cassette not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized **IBM Personal Computer** dealer with a copy of your receipt.

or

2. if IBM or the dealer is unable to deliver a replacement diskette or cassette which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use such program even if IBM or an authorized IBM Personal Computer dealer has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

General

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of Florida.

Should you have any questions concerning this Agreement, you may contact IBM by writing to IBM Personal Computer, Sales and Service, P.O. Box 1328-W Boca Raton, Florida 33432.

You acknowledge that you have read this agreement, understand it and agree to be bound by its terms and conditions. You further agree that it is the complete and exclusive statement of the agreement between us which supercedes any proposal or prior agreement, oral or written, and any other communications between us relating to the subject matter of this agreement.

Program Interface Series

IBM Personal Computer Data Acquisition and Control Adapter Programming Support

Guide

Engineering/Scientific Family

Personal
Computer
Software

6137681

International Business Machines Corporation Boca Raton, Florida 33432

IBM Program License Agreement

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE OPENING THIS DISKETTE(S) OR CASSETTE(S) PACKAGE. OPENING THIS DISKETTE(S) OR CASSETTE(S) PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED, AND YOUR MONEY WILL BE REFUNDED.

IBM provides this program and licenses its use in the United States and Puerto Rico. You assume responsibility for the selection of the program to achieve your intended results, and for the installation, use and results obtained from the program.

LICENSE

You may:

a. use the program on a single machine;

b. copy the program into any machine readable or printed form for backup or modification purposes in support of your use of the program on the single machine (Certain programs, however, may include mechanisms to limit or inhibit copying. They are marked "copy protected.");

c. modify the program and/or merge it into another program for your use on the single machine (Any portion of this program merged into another program will continue to be subject to the terms and conditions of this Agreement.); and,

d. transfer the program and license to another party if the other party agrees to accept the terms and conditions of this Agreement. If you transfer the program, you must at the same time either transfer all copies whether in printed or machine-readable form to the same party or destroy any copies not transferred; this includes all modifications and portions of the program contained or merged into other programs.

You must reproduce and include the copyright notice on any copy, modification or portion merged into another program.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE PROGRAM, OR ANY COPY, MODIFICATION OR MERGED PORTION, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE.

IF YOU TRANSFER POSSESSION OF ANY COPY, MODIFICATION OR MERGED PORTION OF THE PROGRAM TO ANOTHER PARTY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

TERM

The license is effective until terminated. You may terminate it at any other time by destroying the program together with all copies, modifications and merged portions in any form. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the program together with all copies, modifications and merged portions in any form.

LIMITED WARRANTY

THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT IBM OR AN AUTHORIZED PERSONAL COMPUTER DEALER) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Continued on inside back cover

Program Interface Series

IBM Personal Computer Data Acquisition and Control Adapter Programming Support

Guide

Engineering/Scientific Family

Personal
Computer
Software

First Edition (November 1984)

About This Book

This book provides software information on the installation, use, and expansion of the IBM Data Acquisition and Control System. The software works in conjunction with the IBM Data Acquisition and Control Adapter, an optional accessory for your IBM Personal Computer. Topics covered in this book include:

- Digital to Analog conversion
- Analog to Digital conversion
- The capabilities and requirements of the programs in this product
- Installation and start-up procedures
- The use of Data Acquisition and Control functions with your application programs
- Considerations for expanding the capabilities of the system.

Book Structure

Chapter 1, "Before Using the Data Acquisition and Control System," outlines the software installation. This includes procedures for backing up your diskette, installing your files, and configuring the system.

Chapter 2, "Data Acquisition and Control Concepts," covers the basic concepts governing the structure and use of the Data Acquisition and Control software. It discusses analog to digital and digital to analog conversions, sampling theory, event counting, and data synchronization.

Chapter 3, "Using the Functions," introduces the functions. It explains the types and classes of functions and outlines their use in programming.

Chapter 4, "Function Arguments," explains arguments and variables for the supported languages.

Chapter 5, "BASIC Functions," contains an alphabetic list of the BASIC functions with examples, formats, and comments. It also covers the Interpreted BASICA header.

Chapter 6, "C Functions," contains an alphabetic list of the C[1] functions with examples, formats, and comments.

Chapter 7, "FORTRAN Functions," contains an alphabetic list of the FORTRAN functions with examples, formats, and comments.

Appendix A, "Error Codes," deals with error codes, what they mean, and what to do about them.

Appendix B, "Sample Programs," contains sample programs, and instructions on using them.

---

[1] This product was developed using the Lattice C Compiler, version 2.0, developed by Lattice, Inc., P.O. Box 648, Hoffman Estates, Illinois.

Appendix C, "Expanding Data Acquisition and Control Capabilities," deals with adapter expansion.

Appendix D, "Performance Consideration," describes device driver processing and performance of the multiple I/O functions.

The book ends with the Glossary and Index.

Audience

This manual is for programmers using IBM Data Acquisition and Control Adapter software on the IBM Personal Computer. To write successful programs, you must have a working knowledge of BASIC, C, or FORTRAN languages. You should be familiar with DOS and its commands to use this product.

Related Publications

- IBM Personal Computer *Disk Operating System* manual
- IBM Personal Computer *BASIC* manual
- IBM Personal Computer *BASIC Compiler* manual
- IBM Personal Computer *FORTRAN Compiler Version 2.00* manual
- IBM Personal Computer *Professional FORTRAN* manual
- IBM Personal Computer *Guide to Operations* manual
- IBM Personal Computer *Technical Reference* manual.

Machine Requirements

- An IBM Personal Computer with a minimum of 64K bytes of memory (BASIC, C, and FORTRAN compilers require more)
- At least one diskette drive with 320K or more bytes capacity
- A monitor (Parts of the BASIC sample program require a color monitor)
- The IBM Personal Computer Data Acquisition and Control Adapter
- Any cables needed to connect the adapter to the desired peripheral device
- Distribution Panel (optional).

Programming Requirements

To use the software, you must have the following available on your system:

- IBM Personal Computer Disk Operating System (DOS), Version 2.00 or later (Professional FORTRAN requres 2.10 later)
- IBM Personal Computer Data Acquisition and Control System software
- The BASIC Interpreter on an optional compiler for the language you are using, such as:
  - — IBM Personal Computer BASIC Compiler
  - — IBM Personal Computer FORTRAN Compiler Version 2.00
  - — IBM Personal Computer Professional FORTRAN
  - — Lattice C Compiler version 2.0.

Terms

A number of terms in this book can be new or unfamiliar. For example:

A/D	Analog-to-Digital. The conversion of data from analog form to digital form (A/D Conversion).
Analog Signal	A signal whose measured value is a continuously variable physical quantity.
D/A	Digital-to-Analog. The conversion of data from digital form to analog form (D/A Conversion).
Digital Signal	A discrete or discontinuous signal; one whose various states are discrete intervals apart.
Range	The difference between the highest and lowest value that a quantity or function can assume.
Sample	A single piece of input data.
TTL	Transistor-Transistor Logic. This indicates the level of a binary signal. The two levels are TTL low, which is less than or equal to 0.4 volts, and TTL high, which is greater than or equal to 2.4 volts.

For other terms, see the Glossary at the back of this guide.

Contents

Chapter 1. Before Using the Data Acquisition and Control System

Contents

This chapter describes how to set up your software to work with your system configuration. This chapter includes a description of the files located on the Master Diskette, and information on how to select the software appropriate to your environment and install it on your system.

Overview

The Data Acquisition and Control System software includes a loadable device driver and bindings to the following languages:

- Interpreted BASICA
- Compiled BASIC
- IBM FORTRAN Version 2.00
- IBM Professional FORTRAN
- Lattice C version 2.0.

Interpreted BASICA requires a program header to calculate absolute addresses for all program functions.

The functions are included in the object modules listed on the following page. The device driver, which must be attached to DOS at start-up time, supports a shared interrupt protocol and is not compatible with vectored-interrupt handlers that interrupt at the same level.

Master Diskette Contents

The Data Acquisition and Control System software is distributed on one Master Diskette. This diskette contains the following files:

DACSET.BAT	Installation program
CONFIG.SYS	Sample configuration file
DAC.COM	Device driver
DACHDR.BAS	Interpreted BASICA header
DACB.OBJ	Compiled BASIC object module
BEXAM.BAS	BASIC example program
DACC.OBJ	C compiler object module
CEXAM.C	C language example program
CEXAM.EXE	C language executable example program
DACF.OBJ	FORTRAN Version 2.00 object module
DACPF.OBJ	Professional FORTRAN object module
FEXAM.FOR	FORTRAN example program
FEXAM.EXE	FORTRAN executable example program.

Check for these files by typing DIR at the DOS prompt. If any file is missing, contact your dealer and obtain another diskette.

Backing Up Your Master Diskette

Before you use the software, you should make a copy of the Master Diskette. From now on, use the backup diskette only. Because your IBM Personal Computer may have one or more diskette drives and

one or more fixed disks, you should go to the correct IBM Personal Computer *Guide to Operations* and the DOS manual to create a backup copy of the Master Diskette.

Creating a Working Environment

Before You Run the Installation Program

The Master Diskette contains a software installation batch file to install the Data Acquisition and Control System software. This batch file makes one diskette or installs the programs on a fixed disk.

The batch file also installs a CONFIG.SYS file in the root directory on the drive you select. This file is necessary to run the software.

> **Note:** The batch file installs this file on the drive you select. If you already have this file on a fixed disk, it is replaced with the one on the Master Diskette. If you want to keep the original file, rename it before running the installation program.

To use the software, you must first run DACSET.BAT, the installation program. Then you must edit your CONFIG.SYS file and initialize your system. The following pages explain these procedures.

The Master Diskette contains the DACSET.BAT installation program. This program performs basic installation procedures. It copies system files to their proper locations and prepares the software to handle programs written in the language you select.

You must perform the following steps before running the installation program:

1. If you are using diskettes, format one blank diskette using the DOS command:

   ```
   format b:/s
   ```

   If you are using a fixed disk system, we recommend that you install the software on the fixed disk. Make sure you have DOS in the root directory.

2. Make your root directory the current directory by typing:

   ```
   cd\
   ```

3. Once the directory is current, make drive A the default by typing:

   ```
   a:
   ```

   at the DOS prompt.

4. Complete the steps listed below. For more information on directories, see your IBM Personal Computer *DOS* manual.

Run the Installation Program

To run the installation program:

1. Insert the Master Diskette into drive A.

2. Type:

   dacset *d: language*

   where *d:* is the drive where you want the programs to be installed (drive B or C) and *language* is the language you want to use. The *language* can be one of the following:

BASICA	for Interpreted BASICA
CBASIC	for Compiled BASIC
FORT	for FORTRAN Version 2.00
PFORT	for Professional FORTRAN
C	for Lattice C compiler

   Note: If your system has only one diskette drive, use B: as the drive parameter. The computer treats drive A as both drive A and B. Also, DOS does not search for a CONFIG.SYS file on drive D. Use drive C only.

3. Press Enter.

4. Answer the prompts that appear on the screen.

When the DOS prompt appears, proceed to the next step.

Modify the CONFIG.SYS File

The installation program installs CONFIG.SYS, a configuration file that allows DOS to access the device driver. You can use this file or modify an existing file. DAC.COM, adding to CONFIG.SYS To add DAC.COM to an existing CONFIG.SYS file, use any line or text editor. Open CONFIG.SYS and add this statement:

DEVICE=*d*:DAC.COM

where *d:* is drive A or drive C.

> **Note:** If you are programming in Interpreted BASICA, DAC.COM must be the last device driver listed in the file.

All Data Acquisition and Control Adapters in your computer must be configured for the same interrupt level. Note also that DAC.COM is compatible only with other devices (and drivers) on the same interrupt level that support a shared interrupt protocol.

When reset, DOS searches for the CONFIG.SYS file and executes the commands in it. Note that CONFIG.SYS should not be confused with (nor be part of) an AUTOEXEC.BAT file. DOS runs the CONFIG.SYS file before moving to an AUTOEXEC.BAT file.

Initialize the System

If your system has only diskette drives, reset the system from the "Data Acquisition and Control System Diskette."

If you are working with a fixed disk system, a reset installs the device driver from the root directory.

After you have completed the installation steps described above, your system is ready to accept calls from your programs.

Hardware Dependent Information

Before you can use the software, you must set switches on the adapter. These switches give the software and your computer the following values:

- Interrupt level
- Adapter number
- Analog input and output ranges.

See the IBM Personal Computer *Guide to Operations* for details on how to set the adapter switches.

Setting Interrupts

Unlike some Personal Computer peripherals, the Data Acquisition and Control Adapter uses a shared interrupt system. While you can use vectored and shared interrupt devices in the same system, no vectored interrupt can have the same priority as that of the adapter.

To configure the adapter's interrupt scheme, set the adapter's on-board dip switches. This assigns interrupt levels for each card in your system. All adapters must be set to the same interrupt level.

The adapter can use interrupt request levels IRQ3 through IRQ7. Be careful to limit the number of peripherals that have a higher assigned priority than the adapter.

Assigning Adapter Numbers

Each adapter has a number that identifies it to the system. This is the *adapter number*. Include this number in the argument list of any function that uses that adapter. Before installing the adapter, set the switches to assign the adapter number. You can assign only one number per adapter. If your system contains only one adapter, assign it 0.

Setting Analog Ranges

The switches govern the voltage range of the analog input and analog output devices. The settings determine the relationship between the analog input voltages and the values returned by the analog input device. They also determine the relationship between the analog output values and voltages output by the analog output device. To accurately compute analog I/O values, you must know the full-scale range of your analog I/O hardware. These ranges are:

-5 to 5 volts

-10 to 10 volts

0 to 10 volts.

Chapter 2. Data Acquisition and Control Concepts

Contents

The Data Acquisition and Control System software consists of a set of functions specifically designed to control and monitor sensor-based systems. Using Compiled BASIC, Interpreted BASICA, Lattice C, or FORTRAN, programmers can take advantage of the adapter's I/O capabilities to perform a wide variety of data acquisition and control tasks.

Your IBM Personal Computer, when equipped with the adapter and supporting software, can read signals from various sensors, transducers, and other instruments. The computer can also generate output signals. By integrating the software with the adapter, you can use your computer to automate nearly any process dealing with analog and binary signals.

The Adapter

The Data Acquisition and Control System Adapter is an integrated package of data acquisition and control hardware. It plugs into any full-length slot in your IBM computer and maps into a range of addresses in the system's I/O space. All on-board devices operate under the control of your application programs.

The adapter contains three on-board I/O devices with the following I/O capabilities:

- Four channels of analog input
- Two channels of analog output
- A 16-bit binary input port
- A 16-bit binary output port
- A one-channel, 16-bit counter.

The adapter also has an expansion bus interface that allows you to increase the number of programmable devices within your system to 256. For more information on the expansion bus interface, see Appendix C, "Expanding Data Acquisition and Control Capabilities."

Understanding Data Acquisition and Control

Nearly all forms of data in the real world are analog in nature. Anything that measures or monitors variable data produces a variable electronic signal that corresponds to changes in the data. The Data Acquisition and Control Adapter includes an analog input device that converts analog input into digital form that the computer can store and use.

Binary (or digital) data has only two expressions, 1 and 0, corresponding to On and Off states. The computer reading these values can check the status of two-state devices. These devices can be on or off, open or closed, high or low. Binary data can also indicate whether the output from an instrument lies within or exceeds acceptable limits.

Because the adapter translates analog information into the digital form required for processing, it can communicate with a wide variety of instruments that provide only analog signals. On the other hand, the adapter also understands and processes digital data produced by a source instrument.

The device driver software provides the interface between the hardware adapter and the subroutine functions. It is written specifically to run with DOS, and is loaded when the computer is reset with the CONFIG.SYS initialization file. The format conforms to other DOS shared interrupt device drivers as specified in the DOS manual.

Application programs that call the library functions can perform a large number of tasks. They can scan, record, and manipulate source data. They can search for particular conditions, and, when these conditions are found, trigger corresponding actions. For all practical purposes, the data gathered is exactly like any other data available to a computer.

A/D and D/A Conversion

Analog to Digital (A/D) conversion is the process whereby the adapter converts analog signals (voltages) over a given range to a series of discrete digital values. A/D converters are usually categorized by the number of bits of resolution they support. The greater the number of bits, the greater the number of discrete voltage levels that the converter can represent. For instance, the 12-bit converter that the adapter uses can express 4096 different voltage levels.

The relationship between the analog signal and the digital value depends on the configuration of the A/D converter. The difference between the high and low limits of the reference voltage is called the *range*; you can set the range to either 10 or 20 volts.

Analog voltages are either bipolar (positive or negative) or unipolar (positive only), depending on polarity. You select polarity with a switch on the on-board analog device.

D/A conversion is, generally speaking, a simple A/D conversion in reverse. A digital value from the data bus converts to a voltage. The on-board D/A device, with its 12-bit resolution, can output 4096 discrete voltages. The D/A converters can also use several voltage ranges.

See the IBM Personal Computer *Guide to Operations* for more information.

Sampling Theory

When monitoring a constantly changing quantity, you need to sample at slightly more than twice the highest frequency of the event. This ensures accurate detection of changes in analog or digital data. Therefore, if you expect changes to occur at a maximum rate of once every 20 seconds, sample it at least every 9 seconds. A lower sampling rate fails to accurately detect the changes and distorts the digitized data. In this example, a sampling rate of every 4 to 5 seconds is preferable. The higher the sampling rate, the more accurate the digitized data.

Critical timing of multiple samples is extremely important. All critically-timed iterative sampling uses your IBM Personal Computer's CPU. Other demands on the CPU can affect the timing of iterative sampling. These adverse effects typically signify themselves as jitter in the inter-sampling period.

Sampling rates for iterative functions are dependent on the system in a number of ways. Any of the following operations will affect sampling rate integrity by interrupting the process.

- Using the DOS PRINT command, you specify a print queue of files to be printed concurrently with other operations. An active print queue will disrupt sampling and distort the inter-sample period.

- Any device in the system with a higher priority interrupt than the adapter can also interrupt the CPU in the middle of a sampling operation and affect the integrity of the sampling rate.

- The time-of-day clock, which uses interrupt request level IRQ0, always distorts the inter-sample period to some degree. This is more pronounced at higher sampling rates, or when there is any device in the system that uses the clock tick for its own purposes.

- Memory refresh can also distort the inter-sample rate. Again, the distortion is most noticeable when the sampling rate is high.

Synchronizing Data

Not all data remains continuously available to the software. It is impractical to keep a system in constant readiness to receive sporadic signals. Also, if the system is performing other operations when the data is available, the information can be lost completely.

A process called *handshaking* synchronizes input and output devices to solve this problem. One device either signals the other when it is ready, or simply prevents interaction until that time.

Handshaking is optional with Data Acquisition and Control System software; programs are not affected if you don't use it.

Triggering is a means of signaling that data is available. When using an external device, you must write the input program with this in mind. Software triggering involves a program loop that continually tests for a particular condition. The program allows the reception of data when the condition is fulfilled.

Clocking involves an external mechanism that controls the frequency of iterative I O functions. You set the rate you want. Selecting a rate of zero (0) and driving the 'interrupt request' (IRQ) line to low whenever a sample is collected permits clocking of slower or irregular events.

See Chapter 3 for more information on triggering, clocking, and handshaking.

Binary Signals and Ports

The adapter stores binary signals as data words. When monitoring a set of devices, it expresses the state of each device as a digit in a binary number. When a binary input value is read, the value of the input word is returned. A 1 indicates a positive reading from the device, such as on, open, or high. A 0 indicates a negative reading, such as off, closed, or low.

All bits of the binary input port are pulled to their high state internally. If nothing is connected to the input port, a function returns a value of hex FFFF, all bits set to 1. If each switch connected to the binary input port grounds the input when it is turned on, a 1 in the input word represents a switch set OFF; a 0 represents a switch set ON. If a switch connected to a binary input port supplies a high signal when turned on, the only possible value in the input word is 1. You cannot then determine if the switch is OFF or ON.

An analog input device can accept binary data, but this is a waste of capability. The input port of the on-board binary device senses the state of up to 16 binary signals. The binary word's value, in this case, is less important than the values of the individual bits within it.

This port also accepts data words from other devices. In this case, word values rather than bit values are of primary concern.

The output port supplies up to 16 high/low signals. These can be used individually or in groups as data.

Counting Events

The adapter has an on-board counter/timer that keeps track of the number of times an event occurs during a session of sampling. It begins at a given number and counts down (decrements) to track the events. When the countdown reaches 0, it rolls over to the original setting.

A typical counting operation might run like this:

1. Initialize the counter to a known value.
2. Begin counting.
3. Read the counter.
4. Obtain a count by comparing the value read with the initial value.

The on-board counter is a 16-bit count-down device. Its maximum setting is hex FFFF.

See "More About...Counter Functions" in Chapter 3 for more information.

Chapter 3. Using the Functions

Contents

This chapter explains the concepts that apply to the Data Acquisition and Control System functions. These are listed by type in Chapters 5, 6, and 7 and are individually explained there in detail.

Types of Functions

When programming, you can use three types of functions:

- **Input Functions** collect input data and move it to memory.
- **Output Functions** move data from memory to an external device.
- **Utility Functions** control counter/timers and program execution.

Functions also fall into distinct classes according to the rate or frequency at which the adapter performs them:

- **Simple functions**: Also called non-iterative functions, these execute only once.
- **Multiple Functions**: These iterative functions execute according to the number of times specified by a count argument in the function. A rate argument governs the rate of execution.
- **Scanning Functions**: Scans are sets of single inputs collected from a range of consecutively numbered channels. These samples are taken as close together as the external device allows.

Function Names

Function names reflect the function type and the device involved. The first letter signifies the type of device:

- A = analog device
- B = binary device
- Bit = a binary function that works with only a single binary word
- C = counter device.

The next two letters indicate input and output:

- IN = input function
- OU = simple function

The last letters show the rate type:

- S = simple function
- M = multiple function
- SC = scanning instruction.

For instance, *BINM* is a multiple binary input (*B*inary *IN*put *M*ultiple). *AINS* is a simple analog input (*A*nalog *IN*put *S*imple).

Some combinations of these never occur or are unusable. However, you should be able to identify most of the functions with this information.

More About...Analog Functions

Analog Input Functions

The Data Acquisition and Control Adapter System recognizes three analog input functions:

- AINM - Analog Input Multiple
- AINS - Analog Input Simple
- AINSC - Analog Input Scan.

These analog-to-digital functions control the A/D conversion hardware on the adapter and on external devices. The analog input device on the adapter is always device 9. It has:

- Four multiplexed analog input channels. The four input channels multiplex into one A/D converter. You set the input ranges with switches on the adapter.
- 12-bit resolution. This resolution level allows the adapter to distinguish between 4096 discrete voltage levels.
- Switch-selectable ranges. As mentioned earlier, you choose switch settings to control analog input ranges.
- A data conversion handshaking option using the 'A/D convert out' (A/D CO) and 'A/D convert enable' (A/D CE) lines.

Analog Output Functions

The adapter recognizes two analog output functions:

- AOUM - Analog Output Multiple
- AOUS - Analog Output Simple.

These digital-to-analog conversion functions involve the D/A conversion hardware on the adapter and on optional expansion devices. The analog output device on the adapter is device 9 and has:

- Two discrete analog output channels.
  Set the appropriate switches as explained below.
- 12-bit resolution.

  This resolution level allows the adapter to output 4096 discrete voltage levels.
- Switch-selectable ranges.

  As mentioned earlier, you choose switch settings to control the ranges of analog values.

The setting of the switches determines the relationship between analog output values and the voltages transmitted by the analog device. When computing analog I/O values, be sure to consider the full-scale range of your I/O hardware. Analog output range settings differ for each channel. See the IBM Personal Computer *Guide to Operations* for more information.

Analog Handshaking

Handshaking is a way to signal the readiness of an I/O device to input or output data. You can handshake in any of three ways. One method uses the IRQ line; another uses the A/D CO line; a third uses the A/D CE line. All operate on TTL digital signals, and all are available on the adapter. These are options. They do not affect the A/D conversion process.

Analog Handshaking with the IRQ Line

All handshaking schemes control the way in which the computer accepts data from an external device. Normally, the adapter clocks the sampling rate of multiple analog input functions. You specify the rate in the rate argument of the AINM and AINSC functions. This controls a timer-generated interrupt that in turn controls the rate at which samples are taken. When you select a sampling rate based on data that is irregularly available, or available less than once per second, you may want to use external clocking. To do this, perform the following steps:

1. Set the function rate argument to 0.

2. Use a tri-state external device to drive the IRQ line to a TTL low voltage state. This triggers an analog sample. The sample is taken on the low-going transition of the pulse.

   The A/D CO line goes high as soon as the sample has been taken.

3. Have the external device immediately release the IRQ line.

Handshaking with the A/D CO Line

You may want an external device to transmit data only when your computer is ready to receive it. If so, set up a handshaking scheme using the following:

1. Write a program to collect data from an external device that broadcasts when that device detects a TTL high pulse on the A/D CO line.

2. Connect the external device's analog output to an on-board analog input.

3. Connect the external device's 'data requested' line to the A/D CO line.

Now, when an analog input function executes, the on-board analog device sends a pulse, via the A/D CO line, to the external device. The external device immediately places data on the analog input device's input line.

Handshaking with the A/D CE Line

You may want your system to perform A/D conversion only when the external device has data to transmit. If so, use the A/D CE line as follows:

1. Write a program to collect data from an external device that transmits a TTL high pulse when data is available.

2. Connect the external device's analog output to an on-board analog input channel.

3. Connect the external device's 'data available' line to the A/D CE line.

4. Have the external device hold the A/D CE line low until it is ready to send data. This suspends program execution until the first analog input data comes in.

5. Have the external device set the A/D CE line to high as soon as it sends the data. The program then converts the data.

6. After conversion is complete, have the external device set the A/D CE line to low again to repeat the procedure.

When the A/D CE line is set to low, all programmed operations suspend at the first analog input instruction. Therefore, avoid using this method in a multi-tasking environment.

> **Note:** The A/D CE line must remain high until the A/D CO line goes low again. Typically, a 2 microsecond delay occurs between the time A/D CE goes high and the start of the A/D conversion.

More About...Binary Functions

There are six binary I/O functions:

- BINM - Binary Input Multiple
- BINS - Binary Input Simple
- BITINS - Binary Bit Input Simple
- BITOUS - Binary Bit Output Simple
- BOUM - Binary Output Multiple
- BOUS - Binary Output Simple.

These functions call both the binary I/O hardware on the adapter and your optional expansion devices. The binary I/O device has:

- A 16-bit binary output port
- A 16-bit binary input port
- Input and output handshaking control
- Optional external clocking/triggering.

The adapter digital device consists of two subsystems, one for input and one for output.

Binary Input Functions

This subsystem consists of standard TTL level-sensitive devices. An external device can hold (latch) the entire digital input device at any time by pulling the 'binary input hold' (BI HOLD) line to low. Internal resistors (and communication lines) pull these high to +5 volts.

Binary Input Handshaking

You may want to use a handshaking scheme for binary input functions. You can handshake using the 'binary input clear to send' (BI CTS) line or the IRQ line.

Handshaking with the BI CTS Line

The external device involved must be able to transmit data upon receiving a signal from the adapter. The following sequence of events occurs with each sample:

1. Have the external device hold the BI STROBE line low until it is ready to send data. This suspends program execution until the first binary data comes in.

2. The adapter drives the BI CTS line to high.

3. The external device puts new data on the input digital lines and drives the BI STROBE line high. This data must remain valid until the adapter negates the BI CTS line.

4. The adapter negates the BI CTS line.

5. The external device negates the BI STROBE line to repeat the handshaking.

When the BI STROBE line is set to low, all programmed operations suspend at the first binary input instruction. Therefore, avoid using this method in a multi-tasking environment.

Binary Handshaking with the IRQ Line

Normally, the adapter clocks the sampling rate of multiple binary input functions internally. You specify this rate in the rate argument of the BINM function. This controls a timer-generated interrupt that in turn controls binary sampling. When you select a sampling rate based on data that is irregularly available, you may want to use external clocking. The same holds true for data available less than once per second. To do this, perform the following steps:

1. Set the function rate argument to 0.

2. Use a tri-state device to pull the IRQ line to low and trigger a digital output or strobe a digital input.

3. As soon as the data transfer is complete, have the external device release the IRQ line.

Note: You have the option of using the 'binary input strobe' (BI STROBE), BI HOLD, and BI CTS lines. Ignoring them has no affect on the digital I/O process.

Binary Output Functions

The output subsystem uses high-power, tri-state, bus-driving devices. Changes in digital output occur on a per-bit basis. This eliminates unnecessary state transitions and timing glitches. Only bits affected by the changes are actually handled; all others remain the same.

Internal resistors pull all data, handshaking, and control lines high to +5 volts. Unless your application calls for handshaking or control, you do not need to make connections to these lines. To tri-state (float) the output port, the 'binary output gate' (BO GATE) line must be low.

Binary Output Handshaking

You may want to use a handshaking scheme for binary output functions. The external device involved must be able to accept data upon receiving a signal from the adapter. It musts also be able to generate a TTL signal on the 'binary output strobe' (BO STROBE) line to indicate that it is ready to receive. The following sequence of events must occur with each sample:

1. Have the external device hold the BO CTS line low. This suspends program execution until the first binary data comes in.

2. Have the external device set the BO CTS line to high. This indicates it is ready for new data.

3. The adapter sends new data through the binary output port, activates the BO STROBE line for 12 microseconds, and then negates the BO STROBE line.

4. Have the external device negate the BO CTS line to repeat the handshaking.

When the BO CTS line is set to low, all programmed operations suspend at the first binary output instruction. Therefore, avoid using this method in a mult-tasking environment.

More About...Utility Functions

Counter Functions

There are three counter/timer functions in the software:

- CINM - Counter Input Multiple
- CINS - Counter Input Simple
- CSET - Counter Set.

The adapter includes three 16-bit counter/timers: 0, 1, and 2. 0 and 1 are cascaded together to perform internal clocking functions. Counter 2 is used as a countdown timer and is accessible through the CSET, CINM, and CINS functions as channel 0. Once set to a "start-of-count" value, each transition of a signal on the 'count in' (COUNT IN) line from TTL high to TTL low decrements (decreases) the count by one. When the counter reaches 0, it rolls over and begins again. Set the "start-of-count" value when initializing the counter. You may read either single or multiple values from the counter at any time.

Notes:

1. Each high to low transition on the COUNT IN line decreases the count by one.

2. After a CSET function, the first input on the COUNT IN line initializes the counter to the "start-of-count" value. This cycle is not counted.
3. The on-board counter device has a single channel, 0.

DELAY Function

The software has one more utility function. The DELAY function halts program execution for a specified period of time and then returns the status to the status variable.

Chapter 4. Function Arguments

Contents

This chapter explains the function arguments and software information for use with the Data Acquisition and Control System software. At the beginning of each language section in the following chapters is information on compiling and linking your programs.

Arguments Are...

Every function requires at least one argument; most require several. The argument list determines:

- Which I/O device the function accesses
- On which adapter it is located
- Channel numbers
- The number of samples to read or write
- The variable or array that receives returning input data or sends output data
- The variable that receives the returning execution status.

Most arguments take the form of either integer variables or 2-byte unsigned integers. The language you are using may place other constraints on values or variables. These are explained in the "Remarks" section for each argument.

Types of Arguments

Arguments appear in an argument list following each function. Their purpose determines their place in the list. Not all arguments appear in the argument list of every function; however, the order of the arguments never changes. This order, divided into the following groups, is as follows:

- Adapter, device, and channel numbers. These tell the function which adapter to call. Further, they identify the specific device within that adapter, and the specific channel of that device, if applicable.

- Execution parameters. These supply additional information on execution and data storage.

- Count and rate. These tell iterative functions how many iterations to perform and how fast to perform them.

- Data variable. This is the variable to which an input function writes data, or from which an output function retrieves data.

  Note: In all languages other than C, iterative I/O functions require the data variable to be the first element of a data array.

- Status variable. This is the variable to which the status of the function returns. It indicates the success or type of failure of the function.

When assigning values to arguments (integer arguments in particular), it is important to use the correct data type. It is also important to stay within the appropriate range. To assign a value greater than

32767, convert the unsigned integer value to the signed integer required by BASIC and FORTRAN. Lattice C programmers avoid this problem by using type *unsigned* or type *short integers* for these arguments.

Values greater than 32767, when assigned to integer variables, generate an overflow condition during execution. When returned to integer variables, they usually come out in two's complement form (as values in the range -32768 to -1.) This may affect the way your program tests and uses them.

One way to avoid this is to specify values in hexadecimal form, especially where the bitmasks AND and XOR are concerned. (For more on AND and XOR, see the function pages in the following chapters.)

Reading the Argument Pages

The following pages contain detailed descriptions of each argument. In the examples, arbitrary alphabetic labels represent the arguments. You may change them in the code you write. Or, depending on the language you use, you can name them in more or less the same way. They are intended to clarify the purpose of each argument and to indicate its position in the argument list.

Arguments are position-specific. Be sure that arguments for adapter, device, and channel are consistent with the hardware you're accessing. Also make sure that commas (or other recognized delimiters) separate adjacent arguments.

Adapter Number

Label: adapt

Type: Integer value

Range: 0 to 3

Purpose: The adapter number indicates which of the adapters that function accesses.

Remarks: A single Personal Computer can accommodate up to four adapters. Switches on the card assign each an adapter number of 0 to 3. If a value for this argument lies outside this range (or is not assigned to an adapter currently installed in the system), an Unknown Adapter (128) error returns in the status variable.

Related Arguments:

Device Number

Label: andmsk

Type: Integer value

Range: 0 to hex FFFF

Purpose: The mask value is bit-wise ANDed with values input or output by the binary I/O function.

Remarks: Masking occurs as shown in the table:

Raw Value	ANDmsk Bit	Masked Value
0	0	0
0	1	0
1	0	0
1	1	1

If you don't want to ANDmask, set *andmsk* to a value of hex FFFF. Bit 15 is the most significant bit in the range, and bit 0 the least significant.

Related Arguments:

XORmask

Bit Number

Label: bit

Type: Integer value

Range: 0 to 15

Purpose: This argument specifies a binary input or output bit to be tested, set, or cleared by the function.

Remarks: You must assign an integer of value 15 to 0 to this argument. Bit 15 is the most significant bit, and 0 is the least significant. Other values return an Unknown Bit Value (137) error in the status variable.

Channel High

Label: chanhi

Type: Integer value

Range: *0 to 255*

Purpose: This argument, when included with scanning I/O functions, selects the highest-numbered channel included in the channel scan. Functions accessing a single channel do not require it.

Remarks: Functions accessing a single channel need only a single channel argument. By convention, that argument is *chanlo*.

AINSC is the only function that uses *chanhi*.

The wide range of this argument provides maximum room for the expansion bus interface. The accessed device determines the effective range. The allowable values are shown:

Device Name	Device #	Channel Range
Analog Input	9	0 to 3
Analog Output	9	0 to 1
Binary I/O	8	Not Applicable
Counter	10	0

Channel High

For an expansion device, the range of values for this argument depends on the number of channels the device supports. If the value you use is outside the valid channel range for the device, the actual channel selected is determined by the value of *chanlo modulo* the number of channels supported by the device. If *chanlo* is less than 0 or greater than 255, then *chanlo modulo* determines the number of channels supported.

The value of *chanlo* must be less than or equal to the value of *chanhi*. Otherwise, an Invalid Channel Range (134) error returns in the status variable.

Related Arguments:

Channel Low

Channel Low

Label: chanlo

Type: Integer value

Range: 0 to 255

Purpose: This argument selects the channel that the input or output function accesses. In a scanning input function, it specifies the lowest numbered channel included in the scan.

Remarks: Functions accessing a single channel require only a single channel argument. By convention that argument is *chanlo*.

The wide range for this argument provides maximum room for the expansion bus interface. In practice, the accessed device determines the argument's effective range. The allowable values for the on-board devices are:

Device Name	Device #	Channel Range
Analog Input	9	0 to 3
Analog Output	9	0 to 1
Binary I/O	8	Not Applicable
Counter	10	0

Channel Low

For an expansion device, the range of values for this argument depends on the number of channels the device supports. If the value you use is outside the valid channel range for the device, the actual channel selected is determined by the value of *chanlo modulo* the number of channels supported by the device. If *chanlo* is less than 0 or greater than 255, an Invalid Channel Range (134) error returns in the status variable.

Related Arguments:

Channel High

Label: count

Type: Long integer (or real) value

Range: 0 to 16 000 000

Purpose: This determines the number of times an iterative (multiple) function is performed. It also determines the time value of the DELAY function.

Remarks: The value for this argument must not exceed the amount of storage allocated for the target array of the function. It also must not exceed the amount of data in the source array. This is especially true when the function performs a scanning input. These involve count scans, each of which may generate several values.

In Compiled BASIC and Interpreted BASICA, this argument must be a real variable with an integer value in the specified range. Any fractional component is ignored. In C, this argument must be either a variable of type *long int*, or an expression that evaluates to type *long int*. In FORTRAN, this argument must be either a variable of type INTEGER*4 or an expression that evaluates to type INTEGER*4.

If *count* is 0, the function is called but not performed. If *count* is less than 0 or greater than 16 000 000, an Invalid Count Range (135) error returns in the status variable.

Related Arguments:

Data Variable.

Data Variable

Label: data

Type: Integer variable (integer array)

Range: -32768 to 32767

Purpose: This argument references a variable or array element to which a function will write data.

Remarks: If the function is simple (non-iterative), the variable must be an integer variable. If it is a multiple (iterative), or scanning function, the variable must be the first element of an integer array.

The on-board analog input and output devices have a resolution of 12 bits in the range 0 to 4095. Analog output data outside this range is interpreted *modulo* 4096.

The on-board binary and counter timer devices have a resolution of 16 bits; they return data in the range -32768 to 32767. The most significant bit is 15, and the least significant is 0.

Device Number

Label: device

Type: Integer value

Range: 0 to 255

Purpose: This argument determines which I/O device the function accesses. Every I/O device has a unique device number. Each adapter includes the following on-board devices:

Device #	Device
8	Binary I/O device
9	Analog I/O device
10	Counter device

Remarks: As noted above, values for this argument must fall in the range of 8 to 10. Values from 0 to 7 and from 12 to 255 access devices installed through the expansion bus interface. If a value outside this range appears in this argument, an Unknown Device (131) error returns in the status variable.

The device number chosen must correspond to either an adapter installed in the computer or an expansion device. Attempts to access a device that does not exist, or to access a device with an inappropriate function call, can return erroneous values or a Device Timeout (138) error.

Related Arguments:

Adapter Number

Expansion Device Control

Label: ctrl

Type: Integer value

Range: -32768 to 32767

Purpose: This argument controls certain programmable analog features for the optional expansion devices. If set to 0, it has no effect.

Remarks: This argument allows additional expansion hardware support. Its value can specify many enhanced expansion features, such as programmable analog gain and transducer compensation.

See Appendix C, or the literature accompanying your expansion device hardware, for more information.

Handshake

Label: hndshk

Type: Integer value

Range: 0

Purpose: This argument is reserved.

Remarks: Zero is the only allowed value. For values other than zero, an Unknown Handshake Value (136) error returns in the status variable.

Mode

Label: mode

Type: Integer value

Range: 0 or 128

Purpose: This argument determines if system interrupts are enabled or disabled during the processing of multiple I/O functions.

Remarks: This argument applies only to the AINM, AOUM, BINM, BOUM, and CINM multiple I/O functions. Zero is the only allowed value for other functions.

If *mode* is 0, normal system interrupt processing continues during the processing of the multiple I/O functions. If *mode* is 128, the device driver inhibits system interrupts to increase I/O performance.

Appendix D describes the *mode* and *rate* arguments and their relationship to the device driver processing and performance options.

For values other than 0 and 128, an Unknown Mode (133) error returns in the status variable.

Related Arguments:

Rate.

Label: rate

Type: Long integer (or real) value

Range: 0 to 1 000 000

Purpose: This specifies the rate, in samples-per-second, at which the function executes.

Remarks: If this value is 0, an external clock signal on the IRQ line determines the sampling rate. Also, at 0, the function does not execute until the IRQ line goes from high to low.

If you enter a value greater than either the function or the current device supports, then iterations occur at the maximum rate. A Timer Overrun (1) error or Excessive Timer Overrun (142) error returns in the status variable. See Appendix D for more information on *rate* values.

In Compiled BASIC and Interpreted BASICA programs, this argument must be a real variable containing an integer in the specified range. Decimal components are ignored. In C programs, this argument must be either a variable of type *long int* or an expression that evaluates to type *long int*. In FORTRAN programs, this argument must be either a variable of type INTEGER*4 or an expression that evaluates to type INTEGER*4.

If *rate* is less than 0 or greater than 1 000 000, an Invalid Rate Range (139) error returns to the status variable.

Related Arguments:

Count, Mode

Status Variable

Label: stat

Type: integer

Range: -32768 to 32767

Purpose: This argument references the integer variable to which the function's status code returns.

Remarks: A non-zero return indicates a general execution failure of the function.

See Appendix A for a list of error codes.

Label: stor

Type: Integer value

Range: 0

Purpose: This argument is reserved.

Remarks: Zero is the only allowed value. For values other than zero, an Unknown Storage Operation (132) error returns in the status variable.

XORmask

Label: xormsk

Type: Integer value

Range: 0 to hex FFFF

Purpose: The mask value is bit-wise XORed (eXclusively ORed) with values input or output by the associated function.

Remarks: Masking occurs as shown in this table:

Raw Value	XORmsk Bit	Masked Value
0	0	0
0	1	1
1	0	1
1	1	0

If you don't want to XORmask, set *xormsk* to 0. Bit 15 is the most significant bit, and bit 0 the least significant.

Related Arguments:

ANDmask

Chapter 5. BASIC Functions

Contents

BASIC Functions List

This chapter contains information on the following functions:

AINM	Analog Input Multiple
AINS	Analog Input Simple
AINSC	Analog Input Scan
AOUM	Analog Output Multiple
AOUS	Analog Output Simple
BINM	Binary Input Multiple
BINS	Binary Input Simple
BITINS	Binary Bit Input Simple
BITOUS	Binary Bit Output Simple
BOUM	Binary Output Multiple
BOUS	Binary Output Simple
CINM	Counter Input Multiple
CINS	Counter Input Simple
CSET	Counter Set
DELAY	Delay Execution

Programming with Compiled BASIC

The Compiled BASIC binding is supplied as an object module, DACB.OBJ. Include this module to make the functions available to your compiled BASIC program.

Editing, Compiling, and Linking in Compiled BASIC

Compiled BASIC programs do not require a header. You can create source code for programs in compiled languages by using EDLIN or any other ASCII editor. Call the functions just like any other external function. You must observe the variable-declaration, parameter-passing, and array-dimensioning conventions of the language.

After compiling the source code, link the resulting object modules with the proper object modules and libraries to form an executable (.EXE) file. Enter the module in response to the linker's prompt:

```
DACB.OBJ
```

for Compiled BASIC.

Once the DAC.COM is loaded, the .EXE files execute in the normal way.

See the IBM Personal Computer *BASIC Compiler* manual for more information on compiling and linking your programs.

Programming with Interpreted BASICA

The device driver includes all code needed to access the adapter and interface with Interpreted BASICA. The header file, DACHDR.BAS, calculates values for absolute function calls. Include this file in all Interpreted BASICA programs that use the functions. The sample programs all note that the header, though not shown, is an essential part of the program.

Reserved Variables in BASICA

The following variables are included in the Interpreted BASICA header, and should be treated as reserved words. Most are the functions in real variable form, and are used frequently. The function names are:

AINM	BITOUS
AINS	BOUM
BINM	BOUS
AINSC	CINM
AOUM	CINS
AOUS	COUNT
BINS	CSET
BITINS	DELAY

The following header names should also be treated as reserved words:

ADAPT%	FOUND%
AI	HNAME%
COUNT	SG%
DSEG	STAT%

See the listing of the header in Appendix B for more information about these names.

Using The Header

The first lines of any interpreted BASICA program using the functions must be the header.

The header, DACHDR.BAS, reserves the first 99 program lines. It consists of a series of statements that calculate the absolute values for each function call; this makes the calls available to the program. The header also performs all other required initializations and checks to see if a device driver has been loaded. If it does not find the driver, it displays the message:

```
ERROR: DAAC DEVICE DRIVER NOT FOUND
```

and exits to BASICA.

If it finds the driver, the header executes a DEF SEG. This specifies the segment that contains the device driver and the BASICA interface code. The base address of that segment is stored in DSEG. You can execute DEF SEG again if you need to use other external (to BASICA) code.

The header also initializes the variables used in the function calls, in a series of lines like these:

```
27 AINM      = PEEK(&H13) * 256 + PEEK(&H12)
28 AINS      = PEEK(&H15) * 256 + PEEK(&H14)
```

The contents of these variables represent the offset from DSEG to the code's entry point for that function.

After the header is executed, every function is accessible as an real variable.

> **Note:** The header is supplied as an ASCII file. You can incorporate it into your BASICA program by using the BASICA MERGE command. For more information on using MERGE, refer to the BASIC manual.

See Appendix B for more information on the BASICA header.

Analog Input Multiple
AINM

Purpose: AINM samples analog values from the specified adapter, device, and channel.

Format: CALL AINM (adapt, device, chanlo, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel accessed
ctrl	Expansion device control
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status

Remarks: See Appendix D for more information on programming options and performance considerations.

Analog Input Multiple
AINM

After sampling, AINM stores values sequentially in memory. The data element must be the first element of the storage array. AINM acquires count samples at *rate* samples-per-second. After sampling is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Reads 100 analog values at a rate of 1000 samples-per-second from channel 3 of the on-board analog input device. This is device 9 of adapter 0.
- Stores the values in integer array RAWDATA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDATA.

Analog Input Multiple
AINM

```
100 'AINM example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 9
140 CHANLO% = 3
150 CTRL% = 0
160 MODE% = 0
170 STOR% = 0
180 COUNT = 100
190 RATE = 1000
200 STAT% = 0
220 DIM RAWDATA%(99)
230 CALL AINM (ADAPT%, DEVICE%, CHANLO%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
240 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
250 PRINT RAWDATA%(0)
```

Analog Input Simple
AINS

Purpose: AINS selects a single analog value from the adapter, device, and channel and stores it in the data variable.

Format:

```
CALL AINS (adapt, device, chanlo, ctrl, data, stat)
```

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel accessed
ctrl	Expansion device control
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: The ctrl argument should be set to 0 when accessing the on-board device.

Example: The following program performs these tasks:

- Makes all required declarations.
- Reads a single analog value from channel 3 of the on-board analog input device. This is device 9 of adapter 0.
- Stores the value in integer variable RAWVALUE.
- Checks execution status.

- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
100 'AINS example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 9
140 CHANLO% = 3
150 CTRL% = 0
160 STAT% = 0
170 RAWVALUE% = 0
180 CALL AINS (ADAPT%, DEVICE%, CHANLO%,
CTRL%, RAWVALUE%, STAT%)
190 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
200 PRINT RAWVALUE%
```

Analog Input Scan
AINSC

Purpose: AINSC scans a range of channels on the input device.

Format: CALL AINSC (adapt, device, chanlo, chanhi, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	First channel that scan samples
chanhi	Last channel that scan samples
ctrl	Expansion device control
mode	Must be 0
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: After sampling, AINSC stores values sequentially in memory. The data variable determines the first element of the storage array. AINSC takes count scans at *rate* scans-per-second. When sampling is finished execution status returns to the status variable.

The maximum rate for this function is obtained if only one on-board channel is scanned. As the number of channels increases, the maximum rate decreases.

In the following example, AINSC takes 100 scans at 500 scans-per-second. Each scan inputs one sample from channels 0, 1, 2, and 3. Samples are stored sequentially in RAWDATA%, as shown below:

Array Element	Channel
RAWDATA%(0)	0
RAWDATA%(1)	1
RAWDATA%(2)	2
RAWDATA%(3)	3
RAWDATA%(4)	0
RAWDATA%(5)	1
RAWDATA%(6)	2
RAWDATA%(7)	3
•	•
•	•
•	•
RAWDATA%(396)	0
RAWDATA%(397)	1
RAWDATA%(398)	2
RAWDATA%(399)	3

The computer collects the samples as close to simultaneously as the analog input hardware allows. The nominal skew is approximately 250 microseconds.

Set the control (ctrl) argument to 0 when accessing the on-board device.

Analog Input Scan
AINSC

Example: The following program performs these tasks:

- Makes all required declarations.
- Performs 100 scans of channels 0-3 of the on-board analog input device. This is device 9 of adapter 0. The rate is 500 scans-per-second.
- Stores the value in the integer array RAWDATA.
- Checks execution status.
- If execution status in non-zero, prints an error message and ends the program. Otherwise, prints the first four elements (that is, the first scan) of RAWDATA.

```
100 'AINSC example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 9
140 CHANLO% = 0
150 CHANHI% = 3
160 CTRL% = 0
170 MODE% = 0
180 STOR% = 0
190 COUNT = 100
200 RATE = 500
220 STAT% = 0
230 DIM RAWDATA% = (399)
240 CALL AINSC (ADAPT%, DEVICE%, CHANLO%,
CHANHI%, CRTL%, MODE%, STOR%, COUNT, RATE
RAWDATA%, STAT%)
250 IF STAT% <> 0 THEN PRINT USING
''Execution error '': END
260 FOR N = 0 TO 3
270 PRINT RAWDATA%(N)
280 NEXT
```

Analog Output Multiple AOUM

Purpose: AOUM outputs values to the adapter, device, and channel.

Format: CALL AOUM (adapt, device, chanlo, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
ctrl	Expansion device control
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate in samples-per-second
data	First element of the sampled array
stat	Variable that receives the returning execution status.

Remarks: See Appendix D for more information on programming options and performance considerations.

The data variable determines the first element of the array. AOUM performs *count* samples at *rate* samples-per-second. When sampling is finished, execution status returns to the status variable.

Analog Output Multiple
AOUM

The example iterates the on-board device through all of its possible output voltages, from the lowest to the highest.

Example: The following program performs these tasks:

- Makes all required declarations.
- Fills the integer array RAWDATA with 4096 values in the range 0 to 4095.
- Using AOUM, writes the contents of RAWDATA at a rate of 1000 samples-per-second to channel 1 of the on-board analog output device. This is device 9 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an Execution Complete message.

Analog Output Multiple AOUM

```
100 'AOUM example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 9
140 CHANLO% = 1
150 CTRL% = 0
160 MODE% = 0
170 STOR% = 0
180 COUNT = 4096
190 RATE = 1000
200 STAT% = 0
210 DIM RAWDATA%(4095)
220 FOR N = 0 TO 4095
230      RAWDATA%(N) = N
240 NEXT
250 CALL AOUM (ADAPT%, DEVICE%, CHANLO%,
CTRL%, MODE%, STOR%, COUNT, RATE,
RAWDATA%(0), STAT%)
260 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
270 PRINT ''Execution complete.''
```

Analog Output Simple AOUS

Purpose: AOUS outputs the data variable to the adapter, device, and channel, where it converts it to an analog voltage.

Format: CALL AOUS (adapt, device, chanlo, ctrl, data, stat).

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
ctrl	Expansion device control
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: The data value, latched in the device, remains in effect until it is changed by another function. If output channel 1 is configured for a 0 to 10 volt range, the example sets analog output to +5 volts. This remains constant until changed by AOUS or AOUM.

Analog Output Simple AOUS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using AOUS, writes a value in the middle of the analog range to channel 1 of the on-board analog output device. This is device 9 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution complete" message.

```
100 'AOUS example in BASIC
110 'in BASICA headers must be executed first
120 ADAPT% = 0
130 DEVICE% = 9
140 CHANLO% = 1
150 CTRL% = 0
160 RAWVALUE% = 2048
170 STAT% = 0
180 CALL AOUS (ADAPT%, DEVICE%, CHANLO%,
CTRL%, RAWVALUE%, STAT%)
190 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
200 PRINT ''Execution complete.''
```

Binary Input Multiple
BINM

Purpose: BINM inputs binary words from the adapter and device.

Format: CALL BINM (adapt, device, hndshk, mode, stor, andmsk, xormsk, count, rate, data, stat).

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
mode	Execution mode
stor	Must be 0
andmsk	Value that is bit-wise logically ANDed with input value
xormsk	Value that is bit-wise logically XORed with input value
count	Number of times to execute this function
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Binary Input Multiple
BINM

Remarks: BINM inputs words at *rate* words-per-second. It ANDs with *andmsk*, XORs them with *xormsk*, then stores them sequentially in memory. The data element must be the first element of the storage array. When the function is finished, execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

If you don't want to use masking, specify an AND mask of hex FFFF and an XOR mask of 0. Mask values can be in any number base supported by the language.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BINM, inputs 100 binary words at a rate of 250 samples-per-second from the on-board binary input device. This is device 8 of adapter 0.
- Stores the values in integer array RAWDATA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDATA.

Binary Input Multiple
BINM

```
100 'BINM example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 8
140 HNDSHK% = 0
150 MODE% = 0
160 STOR% = 0
170 ANDMSK% = &HFFFF
180 XORMSK% = &H0
190 COUNT = 100
200 RATE = 250
210 STAT% = 0
220 DIM RAWDATA%(99)
230 CALL BINM (ADAPT%, DEVICE%, HNDSHK%,
MODE%, STOR%, ANDMSK%, XORMSK%, COUNT,
RATE, RAWDATA%(0), STAT%)
240 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
250 PRINT RAWDATA%(0)
```

Binary Input Simple
BINS

Purpose: BINS inputs a 16-bit binary word from the adapter and device. Then it stores the word in the data variable.

Format:

```
CALL BINS (adapt, device, hndshk, data, stat).
```

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: When the function is finished, execution status returns to the status variable. The most significant bit is 15, and the least significant bit is 0.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BINS, reads the value of the binary input word from the on-board binary device. This is device 8 of adapter 0.
- Stores the value in integer variable RAWVALUE.
- Checks execution status.

Binary Input Simple
BINS

- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
100 'BINS example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 8
140 HNDSHK% = 0
150 RAWVALUE% = 0
160 STAT% = 0
170 CALL BINS (ADAPT%, DEVICE%, HNDSHK%,
RAWVALUE%, STAT%)
180 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
190 PRINT RAWVALUE%
```

Binary BIT Input Simple
BITINS

Purpose: BITINS inputs the state of the bit in the binary input port located in the adapter and device.

Format: CALL BITINS (adapt, device, bit, data, stat)

adapt	Adapter number accessed
device	Device number accessed
bit	Bit number (15-0) read
data	Variable that receives the returning bit value
stat	Variable that receives the returning execution status.

Remarks: The bit value (1 or 0) is stored in the data variable. When the function is finished, execution status returns to the status variable.

BITINS does not affect handshaking lines on the specified input port. Bits are numbered 15 to 0, from most significant to least significant.

Binary BIT Input Simple
BITINS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BITINS, reads the value of bit 15 of the binary input word on the on-board binary device. This is device 8 of adapter 0.
- Stores the bit value in the integer variable RAWVALUE.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
100 'BITINS example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 8
140 BIT% = 15
150 RAWVALUE% = 0
160 STAT% = 0
170 CALL BITINS (ADAPT%, DEVICE%, BIT%,
RAWVALUE%, STAT%)
180 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
190 PRINT RAWVALUE%
```

Binary BIT Output Simple
BITOUS

Purpose: BITOUS sets the state of a bit in the binary output word of the adapter and device. The bit takes the value of the data variable.

Format: CALL BITOUS (adapt, device, bit, data, stat)

adapt	Adapter number accessed
device	Device number accessed
bit	The bit number (15 to 0) read
data	Variable from which the bit value is retrieved (must be 0 or 1)
stat	Variable that receives the returning execution status.

Remarks: When the function is finished, execution status returns to the status variable.

BITOUS neither reads nor affects handshaking lines on the binary input port. The function acts only on the bit specified. It numbers bits from 15 to 0, beginning with the most significant. The single exception to this rule occurs when BITOUS is the first binary output function executed after system initialization. In that case, BITOUS sets or clears the specified bit and zeroes all other bits.

Binary BIT Output Simple BITOUS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BITOUS, sets to 1 the value of bit 15 of the binary output word in the on-board binary device. This is device 8 at adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution complete" message.

```
100 'BITOUS example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 8
140 BIT% = 15
150 RAWVALUE% = 1
160 STAT% = 0
170 CALL BITOUS (ADAPT%, DEVICE%, BIT%,
RAWVALUE%, STAT%)
180 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
190 PRINT ''Execution complete.''
```

Binary Output Multiple
BOUM

Purpose: BOUM outputs binary words from the adapter and device.

Format: CALL BOUM (adapt, device, hndshk, mode, stor, andmsk, xormsk, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
mode	Execution mode
stor	Must be 0
andmsk	Value to be bit-wise logically ANDed with output value
xormsk	Value to be bit-wise logically XORed with output value
count	Number of times this function executes
rate	Execution rate in samples-per-second
data	First element of the sampled array
stat	Variable that receives the returning execution status.

Binary Output Multiple
BOUM

Remarks: BOUM ANDs the words with *andmsk* and XORs them with *xormsk*. The words are then output as specified in the *rate* argument. The data variable determines the first element of the array from which BOUM outputs variables. When the function is finished, execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

If you don't want masking, specify an AND mask of hex FFFF and an XOR mask of 0. Mask values may be in any number base supported by the language.

Example: The following program performs these tasks:

- Makes all required declarations (no masking).
- Fills integer array RAWDATA with 100 values in the range 0 to 99.
- Using BOUM, outputs the contents of RAWDATA at a rate of 250 samples-per-second from the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution complete" message.

Binary Output Multiple
BOUM

```
100 'BOUM example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 8
140 HNDSHK% = 0
150 MODE% = 0
160 STOR% = 0
170 ANDMSK% = &HFFFF
180 XORMSK% = &H0
190 COUNT = 100
200 RATE = 250
210 STAT% = 0
220 DIM RAWDATA%(99)
230 FOR N = 0 TO 99
240      RAWDATA%(N) = N
250 NEXT
260 CALL BOUM (ADAPT%, DEVICE%, HNDSHK%,
MODE%, STOR%, ANDMSK%, XORMSK%, COUNT,
RATE, RAWDATA%(0), STAT%)
270 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
280 PRINT RAWDATA%(0)
```

Binary Output Simple
BOUS

Purpose: BOUS outputs the contents of the data variable as a 16-bit binary word.

Format: CALL BOUS (adapt, device, hndshk, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
data	Variable from which data is retrieved
stat	Variable that receives the returning execution status.

Remarks: BOUS operates through the adapter and device. When the function is finished, execution status returns to the status variable. The most significant bit is 15 and the least significant bit is 0.

A data value of 9 (binary word 0000 0000 0000 1001) sets bits 0 and 3 of the binary output word. This latched value remains in effect until changed by another binary output function.

Binary Output Simple BOUS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BOUS, sets bits 0 and 3 of the binary output word of the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints "Execution complete" message.

```
100 'BOUS example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 8
140 HNDSHK% = 0
150 STAT% = 0
160 RAWVALUE% = 9
170 CALL BOUS (ADAPT%, DEVICE%, HNDSHK%,
RAWVALUE%, STAT%)
180 IF STAT% <> 0 THE PRINT USING
''Execution error ###''; STAT% : END
190 PRINT ''Execution complete.''
```

Counter Input Multiple
CINM

Purpose: CINM reads count values from chanlo of the adapter and device.

Format: CALL CINM (adapt, device, chanlo, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: CINM collects values at rate values-per-second and stores them sequentially in memory. The data variable determines the first element of the storage array. When the function is finished, the execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

Counter Input Multiple CINM

Use CSET to initialize the counter before anything is counted.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CINM, reads the on-board counter (device 10) of adapter 0; it performs 100 readings at a rate of 250 samples-per second.
- Stores the values in integer array RAWDATA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDATA.

```
100 'CINM example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 10
140 CHANLO% = 0
150 MODE% = 0
160 STOR% = 0
190 COUNT = 100
200 RATE = 250
210 STAT% = 0
220 DIM RAWDATA%(99)
230 CALL CINM (ADAPT%, DEVICE%, CHANLO%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0),
STAT%)
240 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
250 PRINT RAWDATA%(0)
```

Counter Input Simple
CINS

Purpose: CINS reads the current number of counts input to chanlo of the adapter and device. It then stores the number in the data variable.

Format: CALL CINS (adapt, device, chanlo, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number being accessed
data	Variable that receives the returning data
stat	Variable that receives the returning execution status

Remarks: The on-board counter device has one channel, 0. Use CSET to initialize the counter. When the function is finished, execution status returns to the status variable.

Counter Input Simple CINS

Example: The following program accomplishes these tasks:

- Makes all required declarations.
- Using CINS, reads a value from the on-board counter device (10) of adapter 0.
- Stores the value in integer variable RAWVALUE.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
100 'CINS example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 10
140 CHANLO% = 0
150 RAWVALUE% = 0
160 STAT% = 0
170 CALL CINS (ADAPT%, DEVICE%, CHANLO%,
RAWVALUE%, STAT%)
180 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
190 PRINT RAWVALUE%
```

Counter Setup
CSET

Purpose: CSET initializes the counter of the adapter or device.

Format: CALL CSET (adapt, device, chanlo, setup, data, stat).

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
setup	Must be 0
data	First element of the array that receives the returning data
stat	Variable that receives the returning execution status.

Remarks: The *setup* value determines counter procedures. The data variable fixes the value at which the count begins (or resets). When the function is finished, execution status returns to the status variable.

You must assign *setup* a value of 0. This sets the counter to begin at the value of the data argument and, when the count reaches 0, roll over and begin counting at that value.

The on-board counter device has one channel, channel 0.

While applications can read the counter at any time, CSET provides the only way of initializing it to a specific value and mode of operation.

When CSET executes, the first count initializes the counter and is not itself counted. If you read the counter immediately after executing CSET (and before any counts have occurred), the counter value will not reflect this initialization.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CSET, initializes the on-board counter device (10) of adapter 0. Counting begins at 65535 (the maximum count value).
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program.
- Otherwise, prints a "Counter set" message.

```
100 'CSET example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 DEVICE% = 10
140 CHANLO% = 0
150 SETUP% = 0
160 RAWVALUE% = &HFFFF
170 STAT% = 0
180 CALL CSET (ADAPT%, DEVICE%, CHANLO%,
SETUP%, RAWVALUE%, STAT%)
190 IF STAT% <> 0 THEN PRINT USING
''Execution error ###''; STAT% : END
200 PRINT ''Counter set.''
```

Delay Execution
DELAY

Purpose: DELAY interrupts program execution.

Format: CALL DELAY (adapt, count, stat)

adapt	Adapter number accessed
count	Length of delay (milliseconds)
stat	Variable that receives the returning execution status.

Remarks: DELAY allows you to perform timed sampling at intervals longer than one second allowed by iterative I/O functions. Software overhead must be taken into account, especially when you are using interpreted BASICA. The time required to execute a DELAY function and a subsequent I/O function increases the length of delay by several to several hundred milliseconds.

When the function is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Stops (DELAYs) execution for 1 second (1000 milliseconds).
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution complete" message.

```
100 'DELAY example in BASIC
110 'in BASICA header must be executed first
120 ADAPT% = 0
130 COUNT = 1000
140 STAT% = 0
150 CALL DELAY (ADAPT%, COUNT, STAT%)
160 IF STAT% <>0 THEN PRINT USING
''Execution error ###''; STAT% : END
170 PRINT ''Execution complete.''
```

Chapter 6. C Functions

Contents

C Function List

This chapter contains information on the following functions:

AINM	Analog Input Multiple
AINS	Analog Input Simple
AINSC	Analog Input Scan
AOUM	Analog Output Multiple
AOUS	Analog Output Simple
BINM	Binary Input Multiple
BINS	Binary Input Simple
BITINS	Binary Bit Input Simple
BITOUS	Binary Bit Output Simple
BOUM	Binary Output Multiple
BOUS	Binary Output Simple
CINM	Counter Input Multiple
CINS	Counter Input Simple
CSET	Counter Set
DELAY	Delay Execution

Programming with C

The C binding is supplied as an object module, DACC.OBJ. To make the functions accessible to your C program, include this module in the object modules list that the linker requires.

Note that arguments which are modified when the function executes must be passed as addresses, and are preceded by the & symbol. Certain arguments that are not modified (for example, data values retrieved from memory and output) must also be passed in this form. The example programs demonstrate correct parameter passing technique.

Editing, Compiling, and Linking in C

You can create source code for programs in compiled languages by using EDLIN or any other ASCII editor. Call the functions just like any other external function. You must observe the variable-declaration, parameter-passing, and array dimensioning conventions of the language.

Your programs must be compiled with the "Large Program and Data Model" of the Lattice C version 2.0 compiler for your programs to work properly. After compiling the source code, link the resulting object modules with the proper object modules and libraries to form an executable (.EXE) file. Enter the correct one in response to the linker's prompt:

```
DACC.OBJ
```

Once the DAC.COM is loaded, the .EXE files execute in the normal way.

See your C compiler manual for more information to compile and link your programs.

Analog Input Multiple
AINM

Purpose: AINM samples analog values from the specified adapter, device, and channel.

Format: ainm (adapt, device, chanlo, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel accessed
ctrl	Expansion device control
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: See Appendix D for more information on programming options and performance considerations.

Analog Input Multiple
AINM

After sampling, AINM stores values sequentially in memory. The data element must be the first element of the storage array. AINM acquires count samples at *rate* samples-per-second. After sampling is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Reads 100 analog values at a rate of 1000 samples-per-second from channel 3 of the on-board analog input device. This is device 9 of adapter 0.
- Stores the values in integer array RAWDATA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDATA.

Analog Input Multiple AINM

```
/* AINM example in C*/
#include ''stdio.h''
main()
{
int rawdata[100];
int adapt, device, chanlo, ctrl, mode, stor,
    stat;
long int count, rate;
adapt = 0;
device = 9;
chanlo = 3;
ctrl = 0;
mode = 0;
stor = 0;
count = 100;
rate = 1000;
stat = 0;
ainm (adapt, device, chanlo, ctrl, mode, stor,
      count, rate, rawdata, &stat);
if (stat !=0)
printf (''Execution error %d\n'',stat);
else
printf(''%d\n'' ,rawdata[0]);
}
```

Analog Input Simple
AINS

Purpose: AINS selects a single analog value from the adapter, device, and channel and stores it in the data variable.

Format: ains (adapt, device, chanlo, ctrl, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel accessed
ctrl	Expansion device control
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: The ctrl argument should be set to 0 when accessing the on-board device.

Example: The following program performs these tasks:

- Makes all required declarations.
- Reads a single analog value from channel 3 of the on-board analog input device. This is device 9 of adapter 0.
- Stores the value in integer variable RAWVALUE.
- Checks execution status.

- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
/* AINS example in C */
#include ''stdio.h''
main( )
{
int adapt, device, chanlo, ctrl, rawvalue,
    stat;
adapt = 0;
device = 9;
chanlo = 3;
ctrl = 0;
stat = 0;
ains (adapt, device, chanlo, ctrl, &rawvalue,
      &stat);
if (stat != 0)
printf(''Execution error %d\n'',stat);
else
printf(''%d\n'',rawvalue);
}
```

Analog Input Scan
AINSC

Purpose: AINSC scans a range of channels on the input device.

Format: ainsc (adapt, device, chanlo, chanhi, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	First channel that scan samples
chanhi	Last channel that scan samples
ctrl	Expansion device control
mode	Must be 0
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: After sampling, AINSC stores values sequentially in memory. The data variable determines the first element of the storage array. AINSC takes count scans at *rate* scans-per-second. When sampling is finished execution status returns to the status variable.

The maximum rate for this function is obtained if only one on-board channel is scanned. As the number of channels increases, the maximum rate decreases.

In the following example, AINSC takes 100 scans at 500 scans-per-second. Each scan inputs one sample from channels 0, 1, 2, and 3. Samples are stored sequentially in RAWDATA, as shown below:

Array Element	Channel
RAWDATA(0)	0
RAWDATA(1)	1
RAWDATA(2)	2
RAWDATA(3)	3
RAWDATA(4)	0
RAWDATA(5)	1
RAWDATA(6)	2
RAWDATA(7)	3
•	•
•	•
•	•
RAWDATA(396)	0
RAWDATA(397)	1
RAWDATA(398)	2
RAWDATA(399)	3

The computer collects the samples as nearly simultaneously as possible. The nominal skew is approximately 250 microseconds.

Set the control (*ctrl*) argument to 0 when accessing the on-board device.

Analog Input Scan
AINSC

Example: The following program performs tasks:

- Makes all required declarations.
- Performs 100 scans of channels 0 to 3 of the on-board analog input device. This is device 9 of adapter 0. The rate is 500 scans-per-second.
- Stores the values in the integer array RAWDATA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first four elements (that is, the first scan) of RAWDATA.

```
/* AINSC example in C */
#include ''stdio.h''
main( )
{
int rawdata[400];
int n, adapt, device, chanlo, chanhi, ctrl,
    mode, stor, stat;
long int count, rate;
adapt = 0;
device = 9;
chanlo = 0;
chanhi = 3;
ctrl = 0;
mode = 0;
stor = 0;
count = 100;
rate = 500;
stat = 0;
ainsc (adapt, device, chanlo, chanhi, ctrl,
       mode, stor, count, rate, rawdata, &stat);
if (stat != 0)
    printf(''Execution error %d\n'', stat);
else
     for (n = 0; n <= 3; n++)
    printf(''%d\n'', rawdata[n]);
}
```

Analog Output Multiple
AOUM

Purpose: AOUM outputs values to the adapter, device, and channel.

Format: aoum (adapt, device, chanlo, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
ctrl	Expansion device control
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate in samples-per-second
data	First element of the sampled array
stat	Variable that receives the returning execution status.

Remarks: See Appendix D for more information on programming options and performance considerations.

The data variable determines the first element of the array. AOUM performs *count* samples at *rate* samples-per-second. When sampling is finished, execution status returns to the status variable.

Analog Output Multiple AOUM

The example iterates the on-board device through all of its possible output voltages, from the lowest to the highest.

Example: The following program performs these tasks:

- Makes all required declarations.
- Fills the integer array RAWDATA with 4096 values in the range 0 to 4095.
- Using AOUM, writes the contents of RAWDATA at a rate of 1000 samples-per-second to channel 1 of the on-board analog output device. This is device 9 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution complete" message.

Analog Output Multiple
AOUM

```
/* AOUM example in C */
#include ''stdio.h''
main()
{
int rawdata[4096];
int n, adapt, device, chanlo, ctrl, mode,
    stor, stat;
long int count, rate;
for (n = 0; n <= 4095; n++)
        rawdata[n] = n;
adapt = 0;
device = 9;
chanlo = 1;
ctrl = 0;
mode = 0;
stor = 0;
count = 4096;
rate = 1000;
stat = 0;
aoum (adapt, device, chanlo, ctrl, mode, stor,
      count, rate, rawdata, &stat);
if (stat != 0)
    printf(''Execution error %d\n'', stat);
else
    printf(''Execution complete.\n'');
}
```

Analog Output Simple
AOUS

Purpose: AOUS outputs the data variable to the adapter, device, and channel, where it converts it to voltage.

Format: aous (adapt, device, chanlo, ctrl, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
ctrl	Expansion device control
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: The data value is latched in the device and remains in effect until it is changed by another function. If output channel 1 is configured for a 0 to 10 volt range, then the example sets analog output to +5 volts. This remains constant until changed by AOUS or AOUM.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using AOUS, writes a value in the middle of the analog range to channel 1 of the on-board analog output device. This is device 9 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution complete" message.

```
/* AOUS example in C */
#include ''stdio.h''
main()
{
int adapt, device, chanlo, ctrl, rawvalue,
    stat;
adapt = 0;
device = 9;
chanlo = 1;
ctrl = 0;
rawvalue = 2048;
stat = 0;
aous (adapt, device, chanlo, ctrl,
      &rawvalue, &stat);
if (stat != 0)
    printf(''Execution error %d\n'' ,&stat);
else
    printf(''Execution complete.\n'');
}
```

Binary Input Multiple
BINM

Purpose: BINM inputs binary words from the adapter and device.

Format: binm (adapt, device, hndshk, mode, stor, andmsk, xormsk, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
mode	Execution mode
stor	Must be 0
andmsk	Value that is bit-wise logically ANDed with input value
xormsk	Value that is bit-wise logically XORed with input value
count	Number of times to execute this function
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Binary Input Multiple
BINM

Remarks: BINM inputs words at *rate* words-per-second. It ANDs with *andmsk*, XORs them with *xormsk*, then stores them sequentially in memory. The data element must be the first element of the storage array. When the function is finished, execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

If you don't want to use masking, specify an AND mask of hex FFFF and an XOR mask of 0. Mask values may be in any number base supported by the language.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BINM, inputs 100 binary words at a rate of 250 samples-per-second from the on-board binary input device. This is device 8 of adapter 0.
- Stores the values in integer array RAWDATA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDATA.

Binary Input Multiple BINM

```
/* BINM example in C */
#include ''stdio.h''
main()
{
int rawdata [100];
int adapt, device, hndshk, mode, stor, andmsk,
    xormsk, stat;
long int count, rate;
adapt = 0;
device = 8;
hndshk = 0;
mode = 0;
stor = 0;
count = 100;
rate = 250;
andmsk = -1;
xormsk = 0;
stat = 0;
binm (adapt, device, hndshk, mode, stor, andmsk,
      xormsk, count, rate, rawdata, &stat);
if (stat != 0)
    printf(''Execution error %d\n'' ,stat);
else
    printf(''%d\n'' ,rawdata[0]);
}
```

Binary Input Simple
BINS

Purpose: BINS inputs a 16-bit binary word from the adapter and device. Then it stores the word in the data variable.

Format: bins (adapt, device, hndshk, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: When the function is finished, execution status returns to the status variable. The most significant bit is 15 and the least significant is 0.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BINS, reads the value of the binary input word from the on-board binary device. This is device 8 of adapter 0.
- Stores the value in integer variable RAWVALUE.
- Checks execution status.

Binary Input Simple
BINS

- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
/* BINS example in C */
#include ''stdio.h''
main()
{
int adapt, device, hndshk, rawvalue, stat;
adapt = 0;
device = 8;
hndshk = 0;
stat = 0;
bins (adapt, device, hndshk, &rawvalue,
      &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''%d\n'' ,rawvalue);
}
```

C

Binary BIT Input Simple
BITINS

Purpose: BITINS inputs the state of the bit in the binary input port located in the adapter and device.

Format: bitins (adapt, device, bit, data, stat)

adapt	Adapter number accessed
device	Device number accessed
bit	Bit number (15 to 0) read
data	Variable that receives the returning bit value
stat	Variable that receives the returning execution status.

Remarks: The bit value (1 or 0) is stored in the data variable. When the function is finished, execution status returns to the status variable.

BITINS does not affect handshaking lines on the specified input port. Bits are numbered 15 to 0, from most significant to least significant.

Binary BIT Input Simple
BITINS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BITINS, reads the value of bit 15 (the MSB) of the binary input word on the on-board binary device. This is device 8 of adapter 0.
- Stores the bit value in the integer variable RAWVALUE.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
/* BITINS example in C */
#include ''stdio.h''
main()
{
int adapt, device, bit, rawvalue, stat;
adapt = 0;
device = 8;
bit = 15;
stat = 0;
bitins (adapt, device, bit, &rawvalue, &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''%d\n'' ,rawvalue);
}
```

Binary BIT Output Simple
BITOUS

Purpose: BITOUS sets the state of a bit in the binary output word of the adapter and device. The bit takes the value of the data variable.

Format: bitous (adapt, device, bit, data, stat)

adapt	Adapter number accessed
device	Device number accessed
bit	The bit number (15 to 0) for output
data	Variable from which the bit value is retrieved (must be 0 or 1)
stat	Variable that receives the returning execution status.

Remarks: When the function is finished, execution status returns to the status variable.

BITOUS neither reads nor affects handshaking lines on the binary input port. The function acts on only the bit specified. It numbers bits from 15 to 0, beginning with the most significant. The single exception to this rule occurs when BITOUS is the first binary output function executed after system initialization. In that case, BITOUS sets or clears the specified bit and zeroes all other bits.

Binary BIT Output Simple BITOUS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BITOUS, sets to 1 the value of bit 15 (the MSB) of the binary output word in the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution complete" message.

```
/* BITOUS example in C*/
#include ''stdio.h''
main()
{
int adapt, device, bit, rawvalue, stat;
adapt = 0;
device = 8;
bit = 15;
rawvalue = 1;
stat = 0;
bitous (adapt, device, bit, &rawvalue, &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''Execution complete.\n'');
}
```

Binary Output Multiple
BOUM

Purpose: BOUM outputs binary words from the adapter and device.

Format: boum (adapt, device, hndshk, mode, stor, andmsk, xormsk, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
mode	Execution mode
stor	Must be 0
andmsk	Value to be bit-wise logically ANDed with output value
xormsk	Value to be bit-wise logically XORed with output value
count	Number of times this function executes
rate	Execution rate in samples-per-second
data	First element of the sampled array
stat	Variable that receives the returning execution status.

Binary Output Multiple
BOUM

Remarks: BOUM ANDs the words with *andmsk* and XORs them with *xormsk*. The words are then output as specified in the *rate* argument. The data variable determines the first element of the array from which BOUM outputs variables. When the function is finished, execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

If you don't want masking, specify and AND mask of hex FFFF and an XOR mask of 0. Mask values may be in any number base supported by the language.

Example: The following program performs these tasks:

- Makes all required declarations (no masking).
- Fills integer array RAWDATA with 100 values in the range 0 to 99.
- Using BOUM, outputs the contents of RAWDATA at a rate of 250 samples-per-second from the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an Execution Complete message.

Binary Output Multiple
BOUM

```
/* BOUM example in C */
#include ''stdio.h''
main()
{
int rawdata[100];
int n, adapt, device, hndshk, mode, stor, andmsk,
    xormsk, stat;
long int count, rate;
adapt = 0;
device = 8;
hndshk = 0;
mode = 0;
stor = 0;
count = 100;
rate = 250;
andmsk = -1;
xormsk = 0;
stat = 0;
for (n = 0; n <= 99; n++)
      rawdata[n] = n;
boum (adapt, device, hndshk, mode, stor, andmsk,
      xormsk, count, rate, rawdata, &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''Execution complete.\n'');
}
```

Binary Output Simple
BOUS

Purpose: BOUS outputs the contents of the data variable as a 16-bit binary word.

Format: bous (adapt, device, hndshk, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
data	Variable from which data is retrieved
stat	Variable that receives the returning execution status.

Remarks: BOUS operates through the adapter and device. When the function is finished, execution status returns to the status variable. The most significant bit is 15 and the least significant bit is 0.

A data value of 9 (binary word 0000 0000 0000 1001) sets bits 0 and 3 of the binary output word. This latched value remains in effect until changed by another binary output function.

Binary Output Simple BOUS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BOUS, sets bits 0 and 3 of the binary output word of the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints Execution Complete message.

```
/* BOUS example in C */
#include ''stdio.h''
main()
{
int adapt,device, hndshk, rawvalue, stat;
adapt = 0;
device = 8;
hndshk = 0;
stat = 0;
rawvalue = 9;
bous (adapt, device, hndshk, &rawvalue,
      &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''Execution complete.\n'');
}
```

Counter Input Multiple
CINM

Purpose: CINM reads count values from chanlo of the adapter and device.

Format: cinm (adapt, device, chanlo, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel accessed
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: CINM collects values at *rate* values-per-second and stores them sequentially in memory. The data variable determines the first element of the storage array. When the function is finished, the execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

Counter Input Multiple
CINM

Use CSET to initialize the counter before anything is counted.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CINM, reads the on-board counter (device 10) of adapter 0; it performs 100 readings at a rate of 250 samples-per-second.
- Stores the values in integer array RAWDATA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDATA.

Counter Input Multiple CINM

```
/* CINM example in C */
#include ''stdio.h''
main()
{
int rawdata[100];
int adapt, device, chanlo, mode, stor, count,
    rate, rawdata, &stat);
long int count, rate;
adapt = 0;
device = 10;
chanlo = 0;
mode = 0;
stor = 0;
count = 100;
rate = 250;
stat = 0
cinm (adapt, device, chanlo, mode, stor,
      count, rate, rawdata, &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''%d\n'' ,rawdata[0]);
}
```

C

Counter Input Simple
CINS

Purpose: CINS reads the current number of counts input to *chanlo* of the adapter and device. It then stores the number in the data variable.

Format: cins (adapt, device, chanlo, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number being accessed
data	Variable that receives the returning data
stat	Variable that receives the returning execution status.

Remarks: The on-board counter device has one channel, 0. Use CSET to initialize the counter. When the function is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CINS, reads a value from the on-board counter device (10) of adapter 0.
- Stores the value in integer variable RAWVALUE.
- Checks execution status.

Counter Input Simple CINS

- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVALUE.

```
/* CINS example in C */
#include ''stdio.h''
main()
{
int adapt, device, chanlo, rawvalue, stat;
adapt = 0
device = 10;
chanlo = 0;
stat = 0;
cins (adapt, device, chanlo, &rawvalue,
      &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''%d\n'' ,rawvalue);
}
```

C

Counter Setup
CSET

Purpose: CSET initializes the counter of the adapter or device.

Format: cset (adapt, device, chanlo, setup, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
setup	Counter setup (must be 0)
data	Counter's initialized value
stat	Variable that receives the returning execution status.

Remarks: The *setup* value determines counter procedures. The data variable fixes the value at which the count begins (or resets). When the function is finished, execution status returns to the status variable.

You must assign *setup* a value of 0. This sets the counter to begin at the value of the data argument and, when the count reaches 0, roll over and begin counting at that value.

The on-board counter device has one channel, channel 0.

While applications can read the counter at any time, CSET provides the only way of initializing it to a specific value and mode of operation.

When CSET executes, the first count initializes the counter and is not itself counted. If you read the

counter immediately after executing CSET (and before any counts have occurred), the counter value will not reflect this initialization.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CSET, initializes the on-board counter device 10 of adapter 0. Counting begins at 65535 (the maximum count value).
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program.
- Otherwise, prints a "Counter set" message.

```
/* CSET example in C */
#include ''stdio.h''
main()
{
int adapt, device, chanlo, setup, rawvalue,
    stat;
adapt = 0;
device = 8;
chanlo = 0;
setup = 0;
rawvalue = -1;
stat = 0;
cset (adapt, device, chanlo, setup,
      &rawvalue, &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''Counter set.\n'');
}
```

Delay Execution
DELAY

Purpose: DELAY interrupts program execution.

Format: delay (adapt, count, stat)

adapt	Adapter number accessed.
count	Length of delay (milliseconds)
stat	Variable that receives the returning execution status.

Remarks: DELAY allows you to perform timed sampling at intervals longer than one second allowed by iterative I/O functions. Software overhead must be taken into account. The time required to execute a DELAY function and a subsequent I/O function increases the length of delay by several to several hundred milliseconds.

When the function is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Stops execution for 1 second (1000 milliseconds).
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an Execution complete message.

```
/* DELAY example in C */
#include ''stdio.h''
main()
{
int adapt, stat;
long int count;
adapt = 0;
stat = 0;
count = 1000;
delay (adapt, count, &stat);
if (stat != 0)
   printf(''Execution error %d\n'' ,stat);
else
   printf(''Execution complete.\n'');
}
```

Chapter 7. FORTRAN Functions

Contents

FORTRAN Function List

This chapter contains information on the following functions:

AINM	Analog Input Multiple
AINS	Analog Input Simple
AINSC	Analog Input Scan
AOUM	Analog Output Multiple
AOUS	Analog Output Simple
BINM	Binary Input Multiple
BINS	Binary Input Simple
BITINS	Binary Bit Input Simple
BITOUS	Binary Bit Output Simple
BOUM	Binary Output Multiple
BOUS	Binary Output Simple
CINM	Counter Input Multiple
CINS	Counter Input Simple
CSET	Counter Set
DELAY	Delay Execution

The FORTRAN bindings are supplied as an object module, DACF.OBJ and DACPF.OBJ. Include the module in the object modules list that the linker requires to make functions accessible to your FORTRAN program.

Editing, Compiling, and Linking

You can create source code for programs in compiled languages by using EDLIN or any other ASCII editor. Call functions just like any other external subroutine. You must observe the variable-declaration, parameter-passing, and array dimensioning conventions of the language.

After compiling the source code, link the resulting object modules with the proper object modules and libraries to form an executable (.EXE) file. Enter the correct one in response to the linker's prompt:

```
DACF.OBJ
```

for FORTRAN Version 2.00 and

```
DACPF.OBJ
```

for Professional FORTRAN.

Once the DAC.COM is loaded, the .EXE files execute in the normal way.

See the IBM Personal Computer *FORTRAN Compiler Version 2.00* or *Professional FORTRAN* for more information on compiling and linking your programs.

Analog Input Multiple
AINM

Purpose: AINM samples analog values from the specified adapter, device, and channel.

Format: CALL AINM (adapt, device, chanlo, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel accessed
ctrl	Expansion device control
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: See Appendix D for more information on programming options and performance considerations.

Analog Input Multiple
AINM

After sampling, AINM stores values sequentially in memory. The data element must be the first element of the storage array. AINM acquires count samples at *rate* samples-per-second. After sampling is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Reads 100 analog values at a rate of 1000 samples-per-second from channel 3 of the on-board analog input device. This is device 9 of adapter 0.
- Stores the values in integer array RAWDTA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDTA.

Analog Input Multiple
AINM

```
C       AINM example in FORTRAN
        INTEGER*2 RAWDTA(100)
        INTEGER*2 ADAPT,DEVICE,CHANLO,CTRL,MODE,
       * STOR,STAT
        INTEGER*4 COUNT,RATE
        ADAPT=0
        DEVICE=9
        CHANLO=3
        CTRL=0
        MODE=0
        STOR=0
        COUNT=100
        RATE=1000
        STAT=0
        CALL AINM (ADAPT,DEVICE,CHANLO,CTRL,
       *MODE,STOR,COUNT,RATE,RAWDTA(1),STAT)
        IF (STAT.NE.0) GOTO 300
        WRITE (*,101) RAWDTA(1)
101     FORMAT(1X,I6)
        GOTO 500
300     WRITE(*,100) STAT
100     FORMAT(1X,'Execution error ',I6)
500     END
```

Analog Input Simple
AINS

Purpose: AINS selects a single analog value from the adapter, device, and channel and stores it in the data variable.

Format: CALL AINS (adapt, device, chanlo, ctrl, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel accessed
ctrl	Expansion device control
data	Variable that receives the returning data
stat	Variable that receives the returning execution status.

Remarks: The *ctrl* argument should be set to 0 when accessing the on-board device.

Example: The following program performs these tasks:

- Makes all required declarations.
- Reads a single analog value from channel 3 of the on-board analog input device. This is device 9 of board 0.
- Stores the value in integer variable RAWVAL.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVAL.

Analog Input Simple
AINS

```
C       AINS example in FORTRAN
        INTEGER*2 ADAPT, DEVICE, CHANLO, CTRL,
       *RAWVAL,STAT
        ADAPT=0
        DEVICE=9
        CHANLO=3
        CTRL=0
        STAT=0
        CALL AINS (ADAPT, DEVICE, CHANLO,CTRL,
       *RAWVAL,STAT)
        IF (STAT.NE.0) GOTO 300
        WRITE(*,101) RAWVAL
101     FORMAT(1X,I6)
        GOTO 500
300     WRITE(*,100) STAT
100     FORMAT(1X,'Execution error',I6\)
500     END
```

Analog Input Scan
AINSC

Purpose: AINSC scans a range of channels on the input device.

Format: CALL AINSC (adapt, device, chanlo, chanhi, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	First channel that scan samples
chanhi	Last channel that scan samples
ctrl	Expansion device control
mode	Must be 0
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: After sampling, AINSC stores values sequentially in memory. The data variable determines the first element of the storage array. AINSC takes count scans at rate scans-per-second. When sampling is finished execution status returns to the status variable.

The maximum rate for this function is obtained if only one on-board channel is scanned. As the number of channels increases, the maximum rate decreases.

In the following example, AINSC takes 100 scans at the rate of 500 scans-per-second. Each scan inputs one sample from channels 0, 1, 2, and 3. Samples are stored sequentially in RAWDTA, as shown below:

Array Element	Channel
RAWDTA(1)	0
RAWDTA(2)	1
RAWDTA(3)	2
RAWDTA(4)	3
RAWDTA(5)	0
RAWDTA(6)	1
RAWDTA(7)	2
RAWDTA(8)	3
•	•
•	•
•	•
RAWDTA(397)	0
RAWDTA(398)	1
RAWDTA(399)	2
RAWDTA(400)	3

Note: Because FORTRAN does not have 0th elements, all array element numbers are offset by +1.

The hardware takes the samples as close to simultaneously as the analog input hardware allows. The nominal skew is approximately 250 microseconds.

Analog Input Scan
AINSC

Set the control (*ctrl*) argument to 0 when accessing the on-board device.

Example: The following program performs these tasks:

- Makes all required declarations.
- Performs 100 scans of channels 0 to 3 of the on-board analog input device. This is device 9 of adapter 0. The rate is 500 scans-per-second.
- Stores the values in the integer array RAWDTA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first four elements (that is, the first scan) of RAWDTA.

Analog Input Scan
AINSC

```
C       AINSC example in FORTRAN
        INTEGER*2 RAWDTA(100)
        INTEGER*2 ADAPT,DEVICE,CHANLO,CHANHI,CTRL,
       * MODE,STOR,STAT
        INTEGER*4 COUNT,RATE
        ADAPT=0
        DEVICE=9
        CHANLO=0
        CHANHI=3
        CTRL=0
        MODE=0
        STOR=0
        COUNT=100
        RATE=1000
        STAT=0
        CALL AINSC (ADAPT,DEVICE,CHANLO,CHANHI,CTRL,
       * MODE,STOR,COUNT,RATE,RAWDTA(1),STAT)
        IF (STAT.NE.0) GOTO 300
        DO 200 I=1,4
200     WRITE (*,101) RAWDTA(I)
101     FORMAT(1X,I6)
        GOTO 500
300     WRITE(*,100) STAT
100     FORMAT(1X,'Execution error ',I6\)
500     END
```

Analog Output Multiple AOUM

Purpose: AOUM outputs values to the adapter, device, and channel.

Format: CALL AOUM (adapt, device, chanlo, ctrl, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
ctrl	Expansion device control
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate in samples-per-second
data	First element of the sampled array
stat	Variable that receives the returning execution status.

Remarks: See Appendix D for more information on programming options and performance considerations.

The data variable determines the first element of the array. AOUM performs count samples at rate samples-per-second. When sampling is finished, execution status returns to the status variable.

Analog Output Multiple AOUM

The example iterates the on-board device through all of its possible output voltages, from the lowest to the highest.

Example: The following program performs these tasks:

- Makes all required declarations.
- Fills the integer array RAWDTA with 4096 values in the range 0 to 4095.
- Using AOUM, writes the contents of RAWDTA at a rate of 1000 samples-per-second to channel 1 of the on-board analog output device. This is device 9 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an Execution Complete message.

Analog Output Multiple
AOUM

```
C     AOUM example in FORTRAN
      INTEGER*2 RAWDTA(4096)
      INTEGER*2 ADAPT,DEVICE,CHANLO,CTRL,MODE,
     * STOR,STAT
      INTEGER*4 COUNT,RATE
      DO 200 I = 0,4095
200   RAWDTA(I+1) = I
      ADAPT=0
      DEVICE=9
      CHANLO=1
      CTRL=0
      MODE=0
      STOR=0
      COUNT=4096
      RATE=1000
      STAT=0
      CALL AOUM (ADAPT,DEVICE,CHANLO,CTRL,
     * MODE,STOR,COUNT,RATE,RAWDTA(1),STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101)
101   FORMAT(1X,'Execution complete.'\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Analog Output Simple
AOUS

Purpose: AOUS outputs the data variable to the adapter, device, and channel, where it converts it to voltage.

Format:

```
CALL AOUS (adapt, device, chanlo, ctrl, data, stat)
```

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
ctrl	Expansion device control
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: The data value is latched in the device and remains in effect until it is changed by another function. If output channel 1 is configured for a 0 to 10 volt range, then the example sets analog output to +5 volts. This remains constant until changed by AOUS or AOUM.

Analog Output Simple
AOUS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using AOUS, writes a value in the middle of the analog range to channel 1 of the on-board analog output device. This is device 9 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an Execution complete message.

```
C     AOUS example in FORTRAN
      INTEGER*2 ADAPT,DEVICE,CHANLO,CTRL,
     * RAWVAL,STAT
      ADAPT=0
      DEVICE=9
      CHANLO=1
      CTRL=0
      STAT=0
      RAWVAL=2048
      CALL AOUS (ADAPT,DEVICE,CHANLO,CTRL,
     * RAWVAL,STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101)
101   FORMAT (1X,'Execution Complete.'\)
      GOTO 500
300   WRITE (*,100)STAT
100   FORMAT (1X,'Execution error ',I6\)
500   END
```

Binary Input Multiple
BINM

Purpose: BINM inputs binary words from the adapter and device.

Format:

```
CALL BINM (adapt, device, hndshk, mode, stor,
andmsk, xormsk, count, rate, data, stat)
```

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
mode	Execution mode
stor	Must be 0
andmsk	Value that is bit-wise logically ANDed with input value
xormsk	Value that is bit-wise logically XORed with input value
count	Number of times to execute this function
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Binary Input Multiple
BINM

Remarks: BINM inputs words at rate words-per-second. It ANDs with *andmsk*, XORs them with *xormsk*, then stores them sequentially in memory. The data element must be the first element of the storage array. When the function is finished, execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

If you don't want to use masking, specify an AND mask of hex FFFF and an XOR mask of 0. Mask values may be in any number base supported by the language.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BINM, inputs 100 binary words at a rate of 250 samples-per-second from the on-board binary input device. This is device 8 of adapter 0.
- Stores the values in integer array RAWDTA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDTA.

Binary Input Multiple
BINM

```
C     BINM example in FORTRAN
      INTEGER*2 RAWDTA(100)
      INTEGER*2 ADAPT,DEVICE,HNDSHK,MODE,
     * STOR,ANDMSK,XORMSK
      INTEGER*4 COUNT,RATE
      ADAPT=0
      DEVICE=8
      HNDSHK=0
      MODE=0
      STOR=0
      ANDMSK=-1
      XORMSK=0
      COUNT=100
      RATE=250
      STAT=0
      CALL BINM (ADAPT,DEVICE,HNDSHK,MODE,
     * STOR,ANDMSK,XORMSK,COUNT,RATE,
     * RAWDTA(1),STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101) RAWDATA(1)
101   FORMAT(1X,I6)
      GOTO 500
300   WRITE (*,100) STAT
100   FORMAT (1X,'Execution error ',I6\)
500   END
```

Binary Input Simple
BINS

Purpose: BINS selects a 16-bit binary word from the adapter and device. Then it stores the word in the data variable.

Format: CALL BINS (adapt, device, hndshk, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
data	Variable that receives returning data
stat	Variable that receives the returning execution status.

Remarks: When the function is finished, execution status returns to the status variable. The most significant bit is 15 and the least significant bit is 0.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BINS, reads the value of the binary input word from the on-board binary device. This is device 8 of adapter 0.
- Stores the value in integer variable RAWVAL.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVAL.

Binary Input Simple
BINS

```
C     BINS example in FORTRAN
      INTEGER*2 ADAPTER,DEVICE,HNDSHK,RAWVAL,STAT
      ADAPT=0
      DEVICE=8
      HNDSHK=0
      STAT=0
      CALL BINS (ADAPT,DEVICE,HNDSHK,RAWVAL,
     * STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101) RAWVAL
101   FORMAT(1X,I6)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Binary BIT Input Simple
BITINS

Purpose: BITINS inputs the state of the bit in the binary input port located in the adapter and device.

Format: CALL BITINS (adapt, device, bit, data, stat)

adapt	Adapter number accessed
device	Device number accessed
bit	Bit number (15 to 0) read
data	Variable that receives the returning bit value
stat	Variable that receives the returning execution status.

Remarks: The bit value (1 or 0) is stored in the data variable. When the function is finished, execution status returns to the status variable.

BITINS does not affect handshaking lines on the specified input port. Bits are numbered 15 to 0, from most significant to least significant.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BITINS, reads the value of bit 15 of the binary input word on the on-board binary device. This is device 8 of adapter 0.
- Stores the bit value in the integer variable RAWVAL.

Binary BIT Input Simple
BITINS

- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVAL.

```
C     BITINS example in FORTRAN
      INTEGER*2 ADAPT,DEVICE,BIT,RAWVAL,STAT
      ADAPT=0
      DEVICE=8
      BIT=15
      STAT=0
      CALL BITINS (ADAPT,DEVICE,BIT,RAWVAL,
     * STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101) RAWVAL
101   FORMAT(1X,I6)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Binary BIT Output Simple BITOUS

Purpose: BITOUS sets the state of a bit in the binary output word of the adapter and device. The bit takes the value of the data variable.

Format: CALL BITOUS (adapt, device, bit, data, stat)

adapt	Adapter number accessed
device	Device number accessed
bit	The bit number (15 to 0) for output
data	Variable from which the bit value is retrieved (must be 1 or 0)
stat	Variable that receives the returning execution status.

Remarks: When the function is finished, execution status returns to the status variable.

BITOUS neither reads nor affects handshaking lines on the binary input port. The function acts on only the bit specified. It numbers bits from 15 to 0, beginning with the most significant. The single exception to this rule occurs when BITOUS is the first binary output function executed after system initialization. In that case, BITOUS sets or clears the specified bit and zeroes all other bits.

Binary BIT Output Simple
BITOUS

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BITOUS, sets to 1 the value of bit 15 of the binary output word in the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution Complete" message.

```
C     BITOUS example in FORTRAN
      INTEGER*2 ADAPT,DEVICE,BIT,RAWVAL,STAT
      ADAPT=0
      DEVICE=8
      BIT=15
      RAWVAL=1
      STAT=0
      CALL BITOUS (ADAPT,DEVICE,BIT,RAWVAL,
     * STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101)
101   FORMAT(1X,'Execution complete.'\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Binary Output Multiple
BOUM

Purpose: BOUM outputs binary words from the adapter and device.

Format: CALL BOUM (adapt, device, hndshk, mode, stor, andmsk, xormsk, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
mode	Execution mode
stor	Must be 0
andmsk	Value to be bit-wise logically ANDed with output value
xormsk	Value to be bit-wise logically XORed with output value
count	Number of times this function executes
rate	Execution rate in samples-per-second
data	First element of the sampled array
stat	Variable that receives the returning execution status.

Binary Output Multiple
BOUM

Remarks: BOUM ANDs the words with *andmsk* and XORs them with *xormsk*. It then outputs them at *rate* words-per-second. The data variable determines the first element of the array from which BOUM outputs variables. When the function is finished, execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

If you don't want masking, specify an AND mask of hex FFFF and an XOR mask of 0. Mask values may be in any number base supported by the language.

Example: The following program performs these tasks:

- Makes all required declarations (no masking).
- Fills integer array RAWDTA with 100 values in the range 0 to 99.
- Using BOUM, outputs the contents of RAWDTA at a rate of 250 samples-per-second from the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an Execution Complete message.

Binary Output Multiple
BOUM

```
C     BOUM example in FORTRAN
      INTEGER*2 RAWDTA(100)
      INTEGER*2 ADAPT,DEVICE,HNDSHK,
     * MODE,STOR,ANDMSK,XORMSK,STAT
      INTEGER*4 COUNT,RATE
      ADAPT=0
      DEVICE=8
      HNDSHK=0
      MODE=0
      STOR=0
      COUNT=100
      RATE=250
      ANDMSK=-1
      XORMSK=0
      STAT=0
      DO 200 I = 0,99
200   RAWDTA(I+1) = I
      CALL BOUM (ADAPT,DEVICE,HNDSHK,MODE,
     * STOR,ANDMSK,XORMSK,COUNT,RATE,RAWDTA(1),
     * STAT)
      IF(STAT.NE.0) GOTO 300
      WRITE (*,101) RAWDTA(1)
101   FORMAT(1X,'Execution complete.',\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Binary Output Simple
BOUS

Purpose: BOUS outputs the contents of the data variable as a 16-bit binary word.

Format: CALL BOUS (adapt, device, hndshk, data, stat)

adapt	Adapter number accessed
device	Device number accessed
hndshk	Handshake (must be 0)
data	Variable from which data is retrieved
stat	Variable that receives the returning execution status.

Remarks: BOUS operates through the adapter and device. When the function is finished, execution status returns to the status variable. The most significant bit is 15 and the least significant bit is 0.

A data value of 9 (binary word 0000 0000 0000 1001) sets bits 0 and 3 of the binary output word. This latched value remains in effect until changed by another binary output function.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using BOUS, sets bits 0 and 3 of the binary output word of the on-board binary device. This is device 8 of adapter 0.
- Checks execution status.

Binary Output Simple
BOUS

- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints "Execution Complete" message.

```
C     BOUS example in FORTRAN
      INTEGER*2 ADAPT,DEVICE,HNDSHK,RAWVAL,STAT
      ADAPT=0
      DEVICE=8
      HNDSHK=0
      RAWVAL=9
      STAT=0
      CALL BOUS (ADAPT,DEVICE,HNDSHK,RAWVAL,
     * STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101)
101   FORMAT(1X,'Execution complete.'\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Counter Input Multiple
CINM

Purpose: CINM reads count values from chanlo of the adapter and device.

Format: CALL CINM (adapt, device, chanlo, mode, stor, count, rate, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
mode	Execution mode
stor	Must be 0
count	Number of times the function executes
rate	Execution rate, in samples-per-second
data	First element of the array that receives returning data
stat	Variable that receives the returning execution status.

Remarks: CINM collects values at rate values-per-second and stores them sequentially in memory. The data variable determines the first element of the storage array. When the function is finished, the execution status returns to the status variable.

See Appendix D for more information on programming options and performance considerations.

Counter Input Multiple
CINM

Use CSET to initialize the counter before anything is counted.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CINM, reads the on-board counter device 10 of adapter 0; it performs 100 readings at a rate of 250 samples-per-second.
- Stores the values in integer array RAWDTA.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints the first element of RAWDTA.

```
C     CINM example in FORTRAN
      INTEGER*2 RAWDTA(100)
      INTEGER*2 ADAPT,DEVICE,CHANLO,MODE,STOR,STAT
      INTEGER*4 COUNT,RATE
      ADAPT=0
      DEVICE=10
      CHANLO=0
      MODE=0
      STOR=0
      COUNT=100
      RATE=250
      STAT=0
      CALL CINM (ADAPT,DEVICE,CHANLO,MODE,
     * STOR,COUNT,RATE,RAWDTA(1),STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101) RAWDTA(1)
101   FORMAT(1X,I6\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Counter Input Simple
CINS

Purpose: CINS reads the current number of counts input to chanlo of the adapter and device. It then stores the number in the data variable.

Format: CALL CINS (adapt, device, chanlo, data, stat)

adapt	Adapter number accessed
device	Device number accessed
chanlo	Channel number being accessed
data	Variable that receives the returning data
stat	Variable that receives the returning execution status.

Remarks: The on-board counter device has one channel, 0. Use CSET to initialize the counter. When the function is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CINS, reads a value from the on-board counter device (10) of adapter 0.
- Stores the value in integer variable RAWVAL.
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints RAWVAL.

Counter Input Simple
CINS

```
C     CINS example in FORTRAN
      INTEGER*2 ADAPT,DEVICE,CHANLO,RAWVAL,STAT
      ADAPT=0
      DEVICE=10
      CHANLO=0
      STAT=0
      CALL CINS (ADAPT,DEVICE,CHANLO,RAWVAL,
     *STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE(*,101) RAWVAL
101   FORMAT(1X,I6\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Counter Setup
CSET

Purpose: CSET initializes the counter of the adapter or device.

Format:

```
CALL CSET (adapt, device, chanlo, setup, data, stat)
```

adapter	Adapter number accessed
device	Device number accessed
chanlo	Channel number accessed
setup	Counter setup (must be 0)
data	Counter's initialized value
stat	Variable that receives the returning execution status.

Remarks: The setup value determines counter procedures. The data variable fixes the value at which the count begins (or resets). When the function is finished, execution status returns to the status variable.

You must assign the *setup* argument a value of 0. This sets the counter to begin at the value of the data argument. When the count reaches 0, it rolls over and begins counting at that value.

The on-board counter device has one channel, channel 0.

While applications can read the counter at any time, CSET provides the sole means of initializing it to a specific value and mode of operation.

Counter Setup
CSET

When CSET executes, the first count initializes the counter and is not itself counted. If you read the counter immediately after executing CSET (and before any counts have occurred), the counter value will not reflect this initialization.

Example: The following program performs these tasks:

- Makes all required declarations.
- Using CSET, initializes the on-board counter device (10) of adapter 0. Counting begins at 65535 (the maximum count value).
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints a "Counter Set" message.

```
C     CSET example in FORTRAN
      INTEGER*2 ADAPT,DEVICE,CHANLO,SETUP,
     * RAWVAL,STAT
      ADAPT=0
      DEVICE=10
      CHANLO=0
      SETUP=0
      RAWVAL=-1
      STAT=0
      CALL CSET (ADAPT,DEVICE,CHANLO,SETUP,
     * RAWVAL,STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101)
101   FORMAT(1X,'Counter set.'\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Delay Execution
DELAY

Purpose: DELAY interrupts program execution.

Format: CALL DELAY (adapt, count, stat)

adapt	Adapter number accessed
count	Length of delay (milliseconds)
stat	Variable that receives the returning execution status.

Remarks: DELAY allows you to perform timed sampling at intervals longer than one second allowed by iterative I/O functions. Software overhead must be taken into account. The time required to execute a DELAY function and a subsequent I/O function increases the length of delay by several to several hundred milliseconds.

When the function is finished, execution status returns to the status variable.

Example: The following program performs these tasks:

- Makes all required declarations.
- Stops execution for 1 second (1000 milliseconds).
- Checks execution status.
- If execution status is non-zero, prints an error message and ends the program. Otherwise, prints an "Execution Complete" message.

Delay Execution
DELAY

```
C     DELAY example in FORTRAN
      INTEGER*2 ADAPT,STAT
      INTEGER*4 COUNT
      ADAPT=0
      COUNT=1000
      STAT=0
      CALL DELAY (ADAPT,COUNT,STAT)
      IF (STAT.NE.0) GOTO 300
      WRITE (*,101)
101   FORMAT(1X,'Execution complete.'\)
      GOTO 500
300   WRITE(*,100) STAT
100   FORMAT(1X,'Execution error ',I6\)
500   END
```

Appendixes

Contents

Appendix A. Error Codes

Errors return in decimal to the status variable. Check this variable after an execution to see if an error has occurred. All errors return in the same way. There is, however, a difference between "hard" and "soft" errors, as explained below.

No Error

You see a No Error condition reported in the status variable when everything is working properly.

Error Code	Description
0	No Error: The function has executed normally and control returns to the caller.

Soft Error

A soft error allows the function to execute, but affects the integrity of the data.

Error Code	Description
1	Timer Overrun: The execution rate was faster at times than the device and function could manage. The function executed, but a small percentage of samples was taken at an irregular and slower rate than specified.
142	Excessive Timer Overrun: The execution rate was often faster than the device and function could manage. The function

executed, but a large percentage of samples was taken at an irregular and slower rate than specified.

Hard Errors

A hard error indicates a failure to execute. The function ends prematurely and control returns to the caller. No data is collected.

Error Code	Description
-2	No Device Driver: (Compiled languages only). The device driver DAC.COM was not found.
128	Unknown Adapter: The requested adapter is either not in the system (that is, no adapter is addressed to that adapter number), or the adapter is not working.
131	Unknown Device: Either the requested device is not known, or a presence test on the device has failed.
132	Unknown Storage Operation: An incorrect value is specified for the storage argument.
133	Unknown Execution Mode: An incorrect value is specified for the execution mode argument.
134	Invalid Channel Range: Values for the channel low and channel high arguments do not set a valid channel range (that is, *chanhi* has a lower value than *chanlo* or is greater than 255).

Error Code	Description
135	Invalid Count Range: An incorrect value is specified for the count argument.
136	Unknown Handshake Value: An incorrect value is specified for the handshake argument.
137	Unknown Bit Value: The value for the bit number argument is outside the range of valid bits.
138	Device Timeout: A device that does not exist was specified for the device argument or the external handshaking did not occur.
139	Invalid Rate Range: The value for the rate argument is outside the valid range.

Appendix B. Sample Programs

BASIC Sample Program

Each section of the BASIC sample program illustrates the use of one analog I/O function in Interpreted BASICA. All five sections appear in the file BEXAM.BAS. The program begins with a header (DACHDR.BAS) and concludes with brief error-handling and data-plotting routines. When you run BEXAM.BAS, the header executes first. The program pauses, displaying a prompt until you press the Enter key. Then Section 1 runs. Sections 2 through 5 work in the same way.

Requirements

The BASIC sample program requires the following:

- Analog input and analog output ranges set to -5 to +5 volts
- I/O address for the adapter set to 0
- Color monitor for Sections 4 and 5
- Test circuit
- Distribution Panel.

Wiring the Test Circuit

The sample program requires a test circuit. You use this to run the example program without connections to external devices. The circuit provides analog input signals which it derives from internal voltages on the adapter. It also allows you to see, by varying the intensity of an LED, the effect of changing analog output voltage. This requires a few components available at any electronic supply house; the only tool you need is a medium-sized flat-bladed screwdriver.

The components you need are:

- A 100 K ohm potentiometer.
- A standard Light Emitting Diode (LED) with a 10 K ohm resistor ballast.
- Three clip leads.
- 24 inches of 17 gauge insulated, stranded wire. Use this wire to make six 4- inch jumpers, stripped 1/2-inch at each end.

Wire the test circuit to the Distribution Panel as illustrated in the IBM Personal Computer *Distribution Panel Installation Instructions*. The following table contains the specifications:

From	To
D/A 0	Pot end
D/A 0	LED + (red)
D/A 1	Pot end
AGND	LED – (black)
AGND	A/D 0-
AGND	A/D 1-
A/D0	BO8
A/D1	Pot wiper

Adding the Header

BASICA programs require a header. Supplied on the master diskette in the file DACHDR.BAS, this header is a pre-written section of BASICA code. It must appear at the beginning of your program.

After the header executes, every function is accessible as a real variable.

> **Note:** The header is supplied as an ASCII file. You can incorporate it into your Interpreted BASICA program by using the Interpreted BASICA MERGE statement. For more information on using MERGE, refer to your BASIC manual.

To add the header to your program, first load Interpreted BASICA, then use the following sequence of statements to merge the PROGRAM with DACHDR.BAS:

```
LOAD ''BEXAM.BAS''
MERGE ''DACHDR.BAS''
SAVE ''BEXAM.BAS''
```

Use the ASCII option (,A) when saving files that you may want to merge later. Otherwise the program stores in the normal, tokenized form and will not be MERGE-able.

> **Note:** DACHDR.BAS reserves program lines 1-99 for its own use. If your program includes any lines numbered in this range, the MERGE operation overwrites them. You may, of course, renumber DACHDR.BAS lines any way you want, but remember that the header must precede the rest of the program code.

This header does two jobs. First, it checks to see whether or not the device driver is loaded. If it finds no device driver, it displays this message:

```
ERROR: DAAC DEVICE DRIVER NOT FOUND
```

If it finds the driver, the header executes a DEF SEG. This specifies the segment that contains the device driver and the Interpreted BASICA interface code. The base address of that segment is stored in DSEG. You can execute DEF SEG again if you need to use other external (to Interpreted BASICA) code.

The header reserves the first 99 program lines. It consists of a series of statements that calculate the absolute values for each function call; this makes the calls available to the program.

The header also initializes the variables used in the function calls (AINS, AOUS, and the others), in a series of lines like these:

```
27 AINM =    PEEK(&H13) * 256 + PEEK(&H12)
28 AINS =    PEEK(&H15) * 256 + PEEK(&H14)
```

The contents of these variables represents the offset from DSEG to the code's entry point for that function.

Once the header has executed, every function is accessible as an integer variable. Remember that all variables are global in BASICA; treat those to which the software assigns function addresses as reserved variables.

Global Variables

The example programs use global variables of their own. You must initialize them as shown here:

```
120 ADAPT% = 0
140 'use on-board analog I/O device
150 DEVICE% = 9
160 'and DIMension a 640-point array
170 'to be used in sections 3,4, and 5
180 DIM RAWDATA% (639)
```

For the purposes of these examples, assign address 0 to the adapter, as explained in Chapter 1.

All of the example programs deal with the on-board analog I/O device. Because of this, the programs set DEVICE%=9. They also DIMension an integer array that will hold 640 data points. The number 640 is somewhat arbitrary. The simple data plotting routine used by several of the examples has a 640-point horizontal resolution. In actual practice, you will probably need to dimension all types of arrays at various points in the program. To provide a clear example, these programs use a single array, named RAWDATA%. This array stores all data read and written by multiple analog I/O functions (AINM, AINSC, and AOUM).

Section 1: A Simple Analog Output from Two Channels

This section uses the function AOUS (Analog Output Simple) to:

- output voltages of +5 Volts to analog output channel 1
- output -5 Volts to analog output channel 0.

These voltages serve as the source of the analog signals that are used in the subsequent sections.

The first step is to assign values for the remaining arguments of AOUS (ADAPT% and DEVICE%, remember, have been assigned already).

```
1070 CTRL% = 0
1080 STAT% = 0
1090 LOVOLT% = 0
1100 HIVOLT% = 4095
1110 LOCHAN% = 0
1120 HICHAN% = 1
```

The first call to AOUS sets the output voltage at D/A channel 0 (LOCHAN%) to -5 Volts (LOVOLT%).

```
1140 CALL AOUS (ADAPT%, DEVICE%, LOCHAN%,
CTRL%, LOVOLT%, STAT%)
```

> **Note:** You must set CTRL% (the expansion device control argument) to 0 when accessing the on-board devices.

Following this call (and all calls to the adapter) functions take the time to execute a test and conditional jump to a brief error-handling routine.

The program sets the execution status variable (STAT%) to 0 before each call. Immediately after each call, it tests STAT%. If STAT% is non-zero, it

then sets the variable LNUM% equal to the line number in which the error occurred. The error handler then comes into play, as shown below:

```
1160 IF STAT% <> 0 THEN LNUM% = 1140 : GOTO 6000
```

The error handler looks like this:

```
6000 '****** Error handler begins here *******
6010 '
6020 'on error, print message, number, and exit
6030 PRINT ''Execution Error # '';STAT%; ''in line
number ''; LNUM%
6040 PRINT ''Program Terminated''
6050 END
```

If an error occurs, the line and error numbers print, enabling you to find out — by consulting Appendix A — what went wrong.

The program then continues, setting D/A channel 1 (HICHAN%) to +5 Volts (HIVOLT%).

```
1180 CALL AOUS (ADAPT%, DEVICE%, HICHAN%, CTRL%,
HIVOLT%, STAT%)
```

After testing STAT% again, the program prints a message on the screen and waits for you to press the Enter key.

```
1230 PRINT ''Output voltages set.''
1240 PRINT ''D/A 0 terminal is -5 Volts''
1250 PRINT ''D/A 1 terminal is +5 Volts''
1260 '
1270 '
1280 INPUT ''PRESS ENTER TO RUN SECTION 2'', C$
1290 CLS
1300 '
```

When you press the Enter key, the screen clears and Section 2 begins.

Section 2: A Simple Analog Input from a Single Channel

Section 2 uses AINS (Analog Input Simple) in a loop to continuously read channel 1 of the on-board analog input device of adapter 0. Channel 1 connects through the potentiometer to +/- 5 volts. By twisting the shaft of the potentiometer, you can modulate the amount of the +5 volt signal that reaches analog input channel 1.

As in the section 1, the first step is to assign values for the arguments of the function.

```
2060 CHANNEL% = 1
2070 CTRL% = 0
2080 STAT% = 0
```

> **Note:** Economical programming standards probably dictate a simple reuse of the variable HICHAN% for the channel number argument. However, the program explicitly re-specifies a CHANNEL% variable here; this emphasizes that all variables used by the CALL statement must be assigned before the program performs the call.

The actual sampling takes place under the control of a WHILE...WEND structure. This takes a sample, displays it as a voltage, and checks for a halt request from the keyboard. As long as no key has been pressed, the adapter samples channel 1 continuously.

```
2100 HALT$ = '' ''
2110 LOCATE 25,1
2120 PRINT ''Press any key to stop sampling.''
2130 WHILE HALT$ = '' ''
2140     'get a sample
2150     CALL AINS (ADAPT%, DEVICE%, CHANNEL%,
CTRL%, V%, STAT%)
2160     'if status non-zero, set line and go
to error handler
2170     IF STAT% <> 0 THEN LNUM% = 2150 : GOTO 6000
2180     'convert raw A/D value to volts
2190     VOLTS = V%/409.6-5
2200     'print time of day
2210     LOCATE 1,1,0
2220     PRINT TIME$
2230     'print voltage
2240     LOCATE 4,1,0
2250     PRINT USING ''Channel 1 input
voltage is ##.##''; VOLTS
2260     LOCATE 4,27,0
2270     'test for halt request
2280     HALT$ = INKEY$
2290 WEND
```

The conversion-to-volts equation in line 2190 converts the 12-bit values (in the range 0 to 4095) returned by the analog input device. It becomes a representation of input voltage in the range +/- 5 Volts. If the analog input system were configured to a different range, you would have to change this equation accordingly.

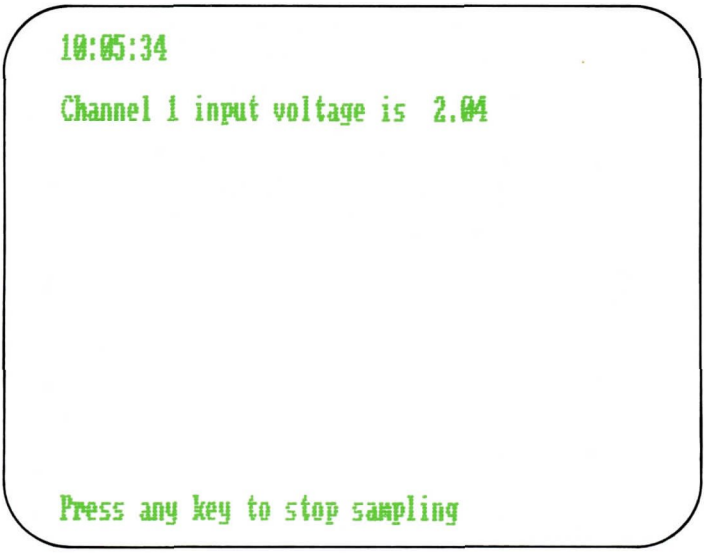
The general form of this equation is:

$$VOLTAGE = ((CODE * RANGE/2^n) - OFFSET)$$

where

- CODE is the raw A/D value.
- RANGE is the A/D.
- Reference voltage span is in volts.
- n is the number of bits of resolution of the A/D converter (exponent of 2).
- OFFSET is the negative offset of the A/D range. A 0 to 10V range, for example, uses the equation VOLTS = V%/409.6 (there is no negative offset). Similarly, if you were using an expansion device of different resolution, you would have to change n to reflect the resolution of that device.

Now run the program. As you turn the shaft of the potentiometer from one stop to the next, you vary the fraction of the +/-5V signal supplied to analog input channel 1. The changing voltage displays on the screen, as shown in the following figure:



Section 3: Multiple Analog Inputs From a Single Channel

This program uses AINM to input 640 analog values into a one-dimensional array. Unlike the single-sample (AINS) loop used in Section 2, AINM acquires signals at precisely-timed intervals. Since the input signal measured (the voltage modulated by the pot) does not fluctuate very rapidly, you can use a slow sampling rate of 200 samples-per-second. The first lines of this program assign values to the arguments of AINM. Remember, the program has already DIMensioned the RAWDATA% array to include 640 elements.

```
3060 CHANNEL% = 1
3070 CTRL% = 0
3080 MODE% = 0
3090 STOR% = 0
3100 COUNT = 640
3110 RATE = 200
3120 STAT% = 0
```

In this program, COUNT and RATE are real, rather than integer, variables. They are the only non-integer variables for arguments to the functions.

The variable COUNT must not exceed the amount of storage available in the target array. AINM acquires count sample, so a statement of the form:

```
DIM ARRAY% (COUNT - 1)
```

allocates the correct number of array elements.

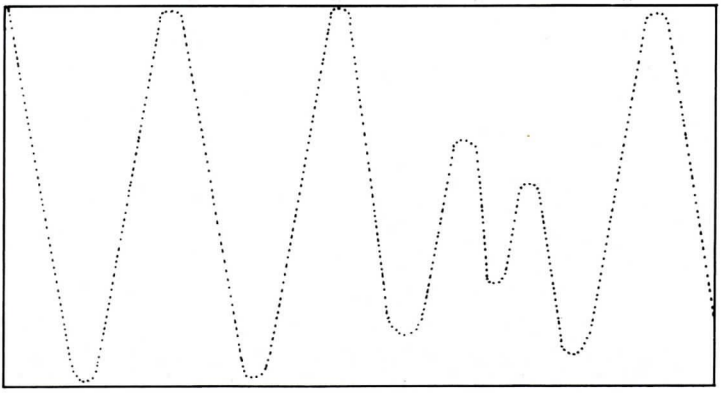
Often, timed sampling of this sort works in conjunction with a triggering structure. Otherwise, you run a risk that the function will sample while no data is being generated. In this example, a simple press of the Enter key triggers the function.

```
3170 INPUT ''Press ENTER to begin sampling.'', C$
3180 'beep and print ''sampling'' message, then do
AINM
3190 BEEP : CLS
3200 PRINT ''Sampling analog input channel 1...''
3210 CALL AINM (ADAPT%, DEVICE%, CHANLO%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
```

Press the Enter key; when you hear the beep, turn the shaft of the potentiometer back and forth for about three seconds (until you see the graph). Once the program collects 640 data points, it jumps to a graphing routine. The routine plots the data, then waits for you to press Enter before moving to the next section.

```
3240 '
3250 'graph points in RAWDATA%
3260 GOSUB 10000
3270 '
```

The figure below shows a typical plot produced by Section 3. This plot was generated on an IBM graphics printer, using the DOS GRAPHICS command and the Shift and PrtSC keys.



The data plotting routine (SIMPLOT) uses the WINDOW, VIEW, and PSET statements of BASICA Version 2.0. You can examine this routine in the listing included in this appendix.

Section 4: An Analog Input Scan of Multiple Channels

Section 4 uses the function AINSC (Analog Input Scan) to input 320 analog values from each of two channels into a two-dimensional array. This section is in most respects identical to Section 3. It first executes a brief subroutine that outputs a binary low voltage (around 0 volts) to analog input channel 0. This simply provides a second data value for the scan to record. If connected to analog input channel 0, it "floats" to an unpredictable value. By driving it low with a binary output, you guarantee that it returns a constant value near 0 volts.

A simple GOSUB statement in line 4050 jumps to the following subroutine.

```
7040 BDEVICE% = 8
7050 'use bit 8
7060 BIT% = 8
7070 'set bit output low
7080 LO% = 0
7090 STAT% = 0
7100 CALL BITOUS (ADAPT%, BDEVICE%, BIT%, LO%,
STAT%)
7110 IF STAT% <> 0 THEN LNUM% = 7037 : GOTO 6000
7120 RETURN
```

This subroutine sets up the required arguments for BITOUS (binary Bit Output Simple); it sets the value of bit 8 to 0. Note that you must use a different variable name, as well as a different number, to specify the device. Simply re-assigning DEVICE%, causes subsequent analog I/O calls to attempt to access device 8. This hangs the system.

Next, the program assigns values to the arguments of AINSC. Note that there are two channel arguments: CL%, specifying the low channel (the first channel in the scan) and CH%, specifying the high channel (the last channel in the scan). For each scan specified by the COUNT, AINSC inputs a sample from every channel in the range CL% to CH%, starting with

CL% and ending after CH%. In this case, each scan consists of an input from channel 0 followed by an input from channel 1.

```
4080 CL% = 0
4090 CH% = 1
4100 CTRL% = 0
4110 MODE% = 0
4120 STOR% = 0
```

Since each scan can acquire multiple samples, be sure to reserve enough array space for the data by a scanning input function such as AINSC. When DIMensioning arrays filled from scans, you may wish to use a statement of the type:

```
DIM ARRAY% (COUNT*((CH - CL)+1))
```

In the case of this section, you'll find it most convenient to simply reduce the count.

```
4150 COUNT = 320
4160 RATE = 200
4170 STAT% = 0
4180 '
```

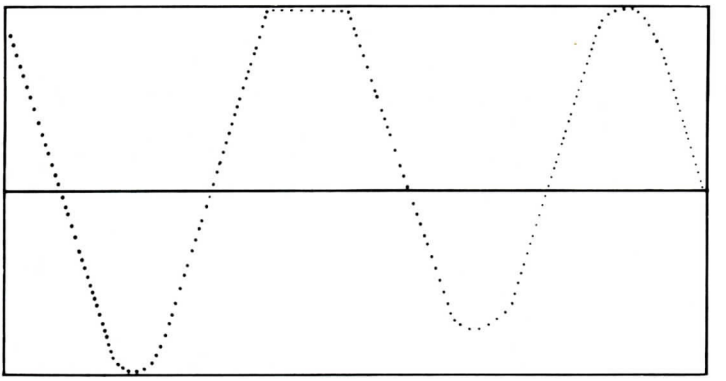
This section uses a triggering structure identical to the one used in Section 3. The only difference is in the data acquisition function itself.

```
4210 INPUT ''Press ENTER to begin scan.'', A$
4220 'beep and print ''scanning'' message, then
     do AINSC
4230 BEEP: CLS
4240 PRINT ''Scanning analog inputs 0 and 1...''
4250 CALL AINSC (ADAPT%, DEVICE%, CL%, CH%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
```

As in Section 3, press Enter and turn the shaft of the potentiometer. After AINSC has input 320 scans, the program jumps to the plotting subroutine and displays the data.

```
4280 '
4290 'display data
4300 GOSUB 10000
4310 '
```

The following figure is a typical plot generated by Section 4.



Section 5: Multiple Analog Outputs From a Single Channel

The final section uses the function AOUM (Analog Output Multiple) to reverse the data acquisition process. This outputs the contents of an array (RAWDATA%), via D/A channel 1, as a varying voltage — an analog signal. When you run this section, the contents of RAWDATA% are output to analog output channel 0. The LED, connected from D/A channel 0 to analog ground (AGND), glows with a varying intensity as the output voltage changes.

The arguments of AOUM first assume these values:

```
5070 CHANNEL% = 1
5080 CTRL% = 0
5090 MODE% = 0
5100 STOR% = 0
5110 COUNT = 640
5120 RATE = 200
5130 STAT% = 0
```

Now print a message and call AOUM.

```
5160 PRINT ''Outputting data acquired in Section 4.''
5170 '
5180 'and do AOUM
5190 CALL AOUM (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
```

The glow of the LED should be directly proportional to the curve plotted in Section 4. If the LED does not glow, re-run BEXAM.BAS; when running section 4, make sure to turn the shaft of the potentiometer all the way from one stop to the other. The curve on the previous page was generated by a RAWDATA% array. When output in Section 5, this made the LED dim and brighten twice.

Run BEXAM.BAS as many times as you want. However, attempts to run parts of the program without first running the header (that is, RUN 1000) may not succeed.

BEXAM.BAS Listing

```
1 ' PROGRAM: BASIC Sample Program
2 ' for the Data Acquisition And Control Adapter
3 '
4 ' FILENAME: BEXAM.BAS
5 '
6 ' WRITTEN BY: Cyborg Corporation (1984)
7 '
8 ' NOTE: See Appendix B for instructions.
9 '
100 'initialize global variables used in all sections
110 'adapter number 0
120 ADAPT% = 0
140 'use on-board analog I/O device
150 DEVICE% = 9
160 'and DIMension a 640-point array
170 'to be used in sections 3,4, and 5
180 DIM RAWDATA% (639)
190 '
200 ' clear the screen and run the first section
210 CLS
215 KEY OFF
220 INPUT ''PRESS ENTER TO RUN SECTION 1'', C$
300 '
1000 '***** Section 1 *******
1010 'this routine sets analog output channel #1
1020 'to +5 Volts and analog output channel #2
1030 'to -5 Volts.
1040 '
1050 'assign values to the arguments of AOUS
1060 '(ADAPT% and DEVICE% were initialized above)
1070 CTRL% = 0
1080 STAT% = 0
```

```
1090 LOVOLT% = 0
1100 HIVOLT% = 4095
1110 LOCHAN% = 0
1120 HICHAN% = 1
1130 'set voltage to -5 at channel 0
1140 CALL AOUS (ADAPT%, DEVICE%, LOCHAN%, CTRL%,
LOVOLT%, STAT%)
1150 'if status non-zero, set line and go to error handler
1160 IF STAT% <> 0 THEN LNUM% = 1140 : GOTO 6000
1170 'set voltage to +5 at channel 1
1180 CALL AOUS (ADAPT%, DEVICE%, HICHAN%, CTRL%,
HIVOLT%, STAT%)
1190 'if status non-zero, set line and go to error handler
1200 IF STAT% <> 0 THEN LNUM% = 1180 : GOTO 6000
1210 '
1220 'print message
1230 PRINT ''Output voltages set.''
1240 PRINT ''D/A 0 terminal is -5 Volts''
1250 PRINT ''D/A 1 terminal is +5 Volts''
1260 '
1270 '
1280 INPUT ''PRESS ENTER TO RUN SECTION 2'', C$
1290 CLS
1300 '
2000 '******* Section 2 **************
2010 'this routine continuously inputs and
2020 'displays as a voltage the analog value
2030 'from channel 1 of the on-board device
2040 '
2050 'assign values for the arguments of AINS
2060 CHANNEL% = 1
2070 CTRL% = 0
2080 STAT% = 0
2090 ' then set up for the loop
```

```
2100 HALT$ = '' ''
2110 LOCATE 25,1
2120 PRINT ''Press any key to stop sampling.''
2130 WHILE HALT$ = '' ''
2140     'get a sample
2150     CALL AINS (ADAPT%, DEVICE%, HANNEL%, CTRL%,
V%, STAT%)
2160     'if status non-zero, set line and go to error handler
2170     IF STAT% <> 0 THEN LNUM% = 2150 : GOTO 6000
2180     'convert raw A/D value to volts
2190     VOLTS = V%/409.6-5
2200     'print time of day
2210     LOCATE 1,1,0
2220     PRINT TIME$
2230     'print voltage
2240     LOCATE 4,1,0
2250     PRINT USING ''Channel 1 input
voltage is ##.##''; VOLTS
2260     LOCATE 4,27,0
2270     'test for halt request
2280     HALT$ = INKEY$
2290 WEND
2300 CLS
2310 '
2320 INPUT ''PRESS ENTER TO RUN SECTION 3'', C$
2330 CLS
2340 '
3000 '********** Section 3 **************
3010 'this program takes 640 samples from
3020 'channel 1 of the on-board analog input device
3030 'and stores them in a one-dimensional array
3040 '
3050 'assign values to the arguments of AINM
3060 CHANNEL% = 1
```

```
3070 CTRL% = 0
3080 MODE% = 0
3090 STOR% = 0
3100 COUNT = 640
3110 RATE = 200
3120 STAT% = 0
3130 '
3140 '
3150 'set up a structure that allows the user to
3160 'start sampling by pressing the ENTER key
3170 INPUT ''Press ENTER to begin sampling.'', C$
3180 'beep and print ''sampling'' message, then do AINM
3190 BEEP : CLS
3200 PRINT ''Sampling analog input channel 1...''
3210 CALL AINM (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
3220 'if status non-zero, set line and go to error handler
3230 IF STAT% <> 0 THEN LNUM% = 3210 : GOTO 6000
3240 '
3250 'graph points in RAWDATA%
3260 GOSUB 10000
3270 '
3280 '
3290 '
3300 '
3310 INPUT ''PRESS ENTER TO RUN SECTION 4'', C$
3320 SCREEN 0, 0, 0
3330 WIDTH 80
3340 CLS
3400 '
4000 '***** Section 4 *****
4010 'this routine takes 50 samples each from
4020 'channels 0 and 1 of the analog input device
4030 'and stores them in a two-dimensional array
```

```
4040 'Start by driving analog input 0 low
4050 GOSUB 7000
4060 '
4070 'assign values to the arguments of AINSC
4080 CL% = 0
4090 CH% = 1
4100 CTRL% = 0
4110 MODE% = 0
4120 STOR% = 0
4130 'note that the count must be 320, rather than 640,
4140 'since each scan acquires two samples
4150 COUNT = 320
4160 RATE = 200
4170 STAT% = 0
4180 '
4190 'set up a structure that allows the user to
4200 'start sampling by pressing the ENTER key
4210 INPUT ''Press ENTER to begin scan.'', A$
4220 'beep and print ''scanning'' message, then do AINSC
4230 BEEP: CLS
4240 PRINT ''Scanning analog inputs 0 and 1...''
4250 CALL AINSC (ADAPT%, DEVICE%, CL%, CH%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
4260 'if status non-zero, set line and go to error handler
4270 IF STAT% <> 0 THEN LNUM% = 4250 : GOTO 6000
4280 '
4290 'display data
4300 GOSUB 10000
4310 '
4320 INPUT ''PRESS ENTER KEY TO RUN SECTION 5'', C$
4330 SCREEN 0, 0, 0
4340 WIDTH 80
4350 CLS
4360 '
5000 '********Section 5*************
5010 ' This example outputs the contents of
5020 ' RAWDATA% to the analog output channel -1
5030 ' An LED connected from channel 1 to channel 0
5040 ' changes intensity as the output voltage changes
5050 '
5060 'assign values to the arguments of AOUM
5070 CHANNEL% = 0
5080 CTRL% = 0
```

```
5090 MODE% = 0
5100 STOR% = 0
5110 COUNT = 640
5120 RATE = 200
5130 STAT% = 0
5140 '
5150 'print message
5160 PRINT ''Outputting data acquired in Section 4.''
5170 '
5180 'and do AOUM
5190 CALL AOUM (ADAPT%, DEVICE%, CHANNEL%, CTRL%,
MODE%, STOR%, COUNT, RATE, RAWDATA%(0), STAT%)
5200 'if status non-zero, set line and go to error handler
5210 IF STAT% <> 0 THEN LNUM% = 5190 : GOTO 6000
5220 LOCATE 10,1,0
5230 PRINT ''End of the Sample Program.''
5240 END
5250 '
5260 '
5500 '
6000 '****** Error handler begins here *******
6010 '
6020 'on status error, print message, error number, and exit
6030 PRINT ''Execution Error # '';STAT%; ''in line
number ''; LNUM%
6040 PRINT ''Program Terminated''
6050 END
6100 '
7000 '******* Binary output subroutine begins here ********
7010 'Set up to drive analog input 0 low using BITOUS
7020 'initialize variables
7030 'use binary device 8
7040 BDEVICE% = 8
7050 'use bit 8
7060 BIT% = 8
7070 'set bit output low
7080 LO = 0
7090 STAT% = 0
7100 CALL BITOUS (ADAPT%, BDEVICE%, BIT%, LO%, STAT%)
7105 'if status non-zero, set line and go to error handler
7110 IF STAT% <> 0 THEN LNUM% = 7037 : GOTO 6000
7120 RETURN
8000 '
```

```
9000 ' ********* Plotting Routine begins here ************
9999 '
10000 ' Name: SIMPLOT
10010 '
10020 ' Arguments: LENGTH% = number of points to plot
10030 ' RAWDATA%(.) = array to be plotted
10040 ' Affects: The screen is left in mode 2. All
10050 ' variables ending in .GRF are reserved.
10060 '
10070 ' This subroutine plots the elements of the array
10080 ' RAWDATA%(.) on a color display. The elements are
10090 ' windowed to the range specified by MAXVAL.GRF and
10100 ' MINVAL.GRF. THe number of points plotted must be
10110 ' passed as an argument in the variable LENGTH%.
10120 ' The screen clears upon each execution of SIMPLOT.
10130 '
20000 MAXVAL.GRF = 4095 ' maximum value to be plotted
20005 LENGTH% = 640
20010 MINVAL.GRF = 0 ' minimum value to be plotted
20020 '
20030 SCREEN 2
20040 CLS
20050 '
20060 VIEW (100,50) - (600,150),,1
20070 WINDOW (0, MINVAL.GRF) - (LENGTH% - 1, MAXVAL.GRF)
20080 '
20090 FOR I.GRF% = 0 TO LENGTH% - 1
20100    PSET (I.GRF%, RAWDATA%(I.GRF%))
20110 NEXT I.GRF%
20120 '
20130 RETURN
```

BASICA Header Listing

```
10 'NAME: Data Acquisition And Control (DAAC)
11 '      HEADER for BASICA
12 '
13 'FILE NAME:  DACHDR.BAS
14 '
15 'DOS DEVICE NAME:  DAAC
16 '
17 'RESERVED FUNCTION NAMES:
18 '           AINM, AINS, AINSC, AOUM, AOUS,
19 '           BINM, BINS, BITINS, BITOUS, BOUM, BOUS,
20 '           CINM, CINS, CSET, DELAY
21 'RESERVED DEF SEG VALUE NAME: DSEG
22 '
23 'NAMES DEFINED AND USED BY HEADER:
24 '           ADAPT%, AI, COUNT, FOUND%,
25 '           HNAME$, SG%, STAT%
26 '
27 '
28 'When using the BASICA Interpreter, this header
29 'must be executed before any function calls are
30 'made that access the DAAC adapter. It initializes
31 'a number of variables for each function call. These
32 'variables are reserved and should not be used except
33 'to access the DAAC adapter. This routine also does a
34 'DEF SEG to the segment where the DAAC Device Driver
35 '(DAC.COM) is loaded. If you execute a DEF SEG to
36 'access other hardware, you must DEF SEG to the segment
37 'of the DAAC Device Driver before any subsequent
38 'calls to access the DAAC adapter.
39 '
40 '
41 FOUND% = 0
42 SG% = &H2E
43 'Start searching the interrupt vectors until you find
44 'one that points to the DAAC device driver.
45 'Do a DEF SEG to that segment.
```

```
46 WHILE ((SG% <= &H3E) AND (FOUND% = 0))
47        DEF SEG = 0
48        DSEG = PEEK(SG%) + PEEK(SG% + 1) * 256
49        DEF SEG = DSEG
50        HNAME$='' ''
51        FOR AI=10 TO 17
52                HNAME$ = HNAME$ + CHR$(PEEK(AI))
53        NEXT AI
54        IF HNAME$ = ''DAAC     '' AND PEEK(18) +
PEEK(19) <> 0 THEN FOUND% = 1
55        SG% = SG% + 4
56 WEND
57 IF FOUND% = 0 THEN PRINT ''ERROR: DEVICE
DRIVER DAC.COM NOT FOUND'' : END
58 'Now initialize all function name variables for calls
59 'to access the device driver.
60 AINM          = PEEK(&H13) * 256 + PEEK(&H12)
61 AINS          = PEEK(&H15) * 256 + PEEK(&H14)
62 AINSC         = PEEK(&H17) * 256 + PEEK(&H16)
63 AOUM          = PEEK(&H19) * 256 + PEEK(&H18)
64 AOUS          = PEEK(&H1B) * 256 + PEEK(&H1A)
65 BINM          = PEEK(&H1D) * 256 + PEEK(&H1C)
66 BINS          = PEEK(&H1F) * 256 + PEEK(&H1E)
67 BITINS        = PEEK(&H21) * 256 + PEEK(&H20)
68 BITOUS        = PEEK(&H23) * 256 + PEEK(&H22)
69 BOUM          = PEEK(&H25) * 256 + PEEK(&H24)
70 BOUS          = PEEK(&H27) * 256 + PEEK(&H26)
71 CINM          = PEEK(&H29) * 256 + PEEK(&H28)
72 CINS          = PEEK(&H2B) * 256 + PEEK(&H2A)
73 CSET          = PEEK(&H2D) * 256 + PEEK(&H2C)
74 DELAY         = PEEK(&H2F) * 256 + PEEK(&H2E)
75 'Finally, execute any call to re-initialize the
76 'device driver from any former invocation of BASIC.
77 ADAPT% = 0
78 COUNT = 1
79 STAT% = 0
80 CALL DELAY (ADAPT%, COUNT, STAT%)
81 '
82 'End of DAAC BASICA Header
83 '
```

FORTRAN and C Sample Programs

The following sample programs are similar programs written in FORTRAN and C.

The programs consist of two loops, one nested within the other. The outer loop iterates the analog output channel 0 (VDACH0) from 0 to 4090. The inner loop iterates the analog output channel 1 (VDACH1) over the same range. Thus, for each setting of channel 0, channel 1 varies over its full possible range. Within the inner loop, the analog channels 0 to 3 are scanned and their values printed.

You must run these examples with the diagnostic wrap connector installed. The wrap connector is supplied with your adapter.

The analog input channels correlate to the following analog output channels:

Analog Input	**Analog output**
0	0
1	1
2	0 minus 1
3	1 minus 0

The displayed values for the four output channels show this correlation.

FORTRAN Sample Program

```
C     PROGRAM FEXAM.FOR
C
C     SAMPLE PROGRAM IN FORTRAN TO DEMONSTRATE USE OF
C     ANALOG INPUT AND OUTPUT FUNCTIONS.
C
C     PROGRAM ASSUMES THE FOLLOWING ADAPTER PARAMETERS:
C          ADAPT = 0, DEVICE = 9 (ANALOG I/O)
C
C     SET FUNCTION PARAMETERS TO INTEGER TYPE
      INTERGER*2 ADAPT,DEVICE,CHANLO,CHANHI
      INTERGER*2 CTRL,STAT,STOR,MODE
      INTERGER*4 COUNT, RATE
C
C     SET ''PROGRAM'' VARIABLES TO INTEGER TYPE
C     VALUES FOR D/A CHANNELS 0 AND 1, RESPECTIVELY
      INTEGER*2 VDACHO, VDACH1
C
C     SET UP ARRAY TO CONTAIN CONVERTED DATA
      INTEGER*2 VADAT(4)
C
C     INITIALIZE THESE VARIABLES TO ZEROES
      DATA VDACHO, VDACH1, VADAT /6*0/
C
C     INITIALIZE FOR PROGRAM USE
      ADAPT = 0
      DEVICE = 9
      CHANLO = 0
      CHANHI = 3
      CRTL = 0
      STAT = 0
      MODE = 0
      STOR = 0
      COUNT = 1
      RATE = 600
C
```

```
C       REMIND USER TO INSTALL WRAP CONNECTOR
C
        WRITE(*,100)
100     FORMAT(1X,'WRAP CONNECTER SHOULD BE INSTALLED'/)
        WRITE(*,200)
200     FORMAT(1X,'BEGINNING EXECUTION'/)
        CALL DELAY (ADAPT,3000,STAT)
C
C       LOOP PROGRAM THRU FULL 12-BIT RANGE OF DEVICE
C       (I.E., 0 - 4096)
C
        DO 400 II = 0, 4096, 256
             VDACH0 = II
             IF (VDACH0.EQ.4096) VDACH0=4095
C
C        CALL ANALOG OUTPUT FUNCTION AND CHECK STATUS
C
             CALL AOUS(ADAPT,DEVICE,0,CTRL,VDACH0,STAT)
             IF (STAT.NE.0) GO TO 500
             DO 300 JJ = 0, 4096, 256
                VDACH1 = JJ
                IF (VDACH1.EQ.4096) VDACH1=4095
                CALL AOUS (ADAPT,DEVICE,1,CTRL,VDACH1,STAT)
                IF (STAT.NE.0) GO TO 500
C
C       USE LOOP-BACK TO ANALOG INPUT SCANNING COMMAND
C       CALL AND USE STATUS CHECK AS BEFORE
C
                CALL AINSC (ADAPT,DEVICE,CHANLO,CHANHI,CTRL,
      *         MODE,STOR,COUNT,RATE,VADAT(1),STAT)
                IF (STAT.NE.0) GO TO 500
C
```

```
C       DISPLAY SECTION
C
                WRITE(*,225)VDACH0,VDACH1
225             FORMAT(1X,'D/A OUTPUTS: C0=',I4,' C1=',I4)
                WRITE (*,250) VADAT
250             FORMAT(1X,'A/D INPUTS: C0=',I4,' C1=',
     *                                  I4,' C2=',I4,' C3=',I4/)
                CALL DELAY (ADAPT,500,STAT)
300          CONTINUE
400     CONTINUE
C
C       JUMP TO END AFTER SUCCESSFUL EXECUTION OF LOOP
C
        GO TO 600
C
C       ERROR-HANDLING SECTION - ALSO TERMINATES EXECUTION
C
500     WRITE(*,550)STAT
550     FORMAT(1X,' ERROR DETECTED - STATUS NUMBER =',I4/)
C
600     WRITE(*,650)
650     FORMAT(1X,'SAMPLE PROGRAM COMPLETE'/)
        END
```

C Sample Program

```
/*
   CEXAM.C - C test program for a/d to d/a operations
         assumes the diag. wrap connector is installed.
*/

main()
{
/*
   parameters required by all functions.
*/

   int      adapt, device, chanlo, chanhi, ctrl;
   int      hndshk, mode, stor;
   long int       count, rate;
   int       stat;

/*
   local variables
*/

   int       vdach0, vdach1, vadat[4];
   int       lower, upper, step0, step1;
   int       n;

  lower = 0;
  upper = 4096;
  step0 = 256;
  step1 = 256;

  adapt = 0 ;
  device = 9 ;
  chanlo = 0 ;
  chanhi = 3 ;
  ctrl = 0 ;
  hndshk = 0 ;
  mode = 0 ;
  stor = 0 ;
  count = 1 ;
  rate = 100 ;
```

```
  vdach0 = lower ;

  while ( vdach0 <= upper ){
    if (vdach0 == 4096) vdach0 = 4095;
    aous( adapt, device, chanlo, ctrl, &vdach0, &stat ) ;
    if (stat != 0)
      printf('' AOUS0 error %d\n'', stat ) ;

    vdach1 = lower ;

    while ( vdach1 <= upper ){
      if (vdach1 == 4096) vdach1 = 4095;
      chanlo = 1 ;
      aous( adapt, device, chanlo, ctrl,
                &vdach1, &stat ) ;
      chanlo = 0 ;
      if (stat != 0)
           printf('' AOUS1 error %d\n'', stat ) ;
      ainsc( adapt, device, chanlo, chanhi, ctrl, mode,
                stor, count, rate, &vadat[0], &stat ) ;
      if (stat != 0)
           printf('' AINSC error %d\n'', stat ) ;
         else {
           printf(''CHAN0 %d CHAN1 %d\n'', vdach0, vdach1) ;
           for (n=0; n<=3; n++)
                printf(''INPUT%d %d\n'',n,vadat[n]);
              }
      delay ( adapt, (long)400, &stat ) ;
      vdach1 = vdach1 + step1 ;
      }
  vdach0 = vdach0 + step0 ;
  }
}
```

Appendix C. Expanding Data Acquisition and Control Capabilities

The Data Acquisition and Control Adapter contains three on-board I/O devices. As described in Chapter 2, "Data Acquisition Concepts," these devices provide the following I/O capabilities:

- Four channels of analog input.
- Two channels of analog output.
- A 16-bit binary input port.
- A 16-bit binary output port.
- A one-channel, 16-bit counter.

This combination of input and output devices proves sufficient for many applications. You can, however, easily expand your adapter to include up to 256 devices. This appendix answers some of the questions that can arise as you consider expanding the I/O capabilities of your adapter.

This appendix should be used in conjunction with the IBM Personal Computer *Technical Reference*.

Adding Expansion Devices

Your adapter has an expansion bus interface that allows you to connect one or more expansion chassis. The expansion bus interface port is a pair of transition headers (J1 and J2).

Expansion Bus Interface

The expansion bus interface provides a full 16-bit parallel data path and addressing for up to 253 expansion devices. The expansion chassis contains the expansion devices.

Expansion Chassis

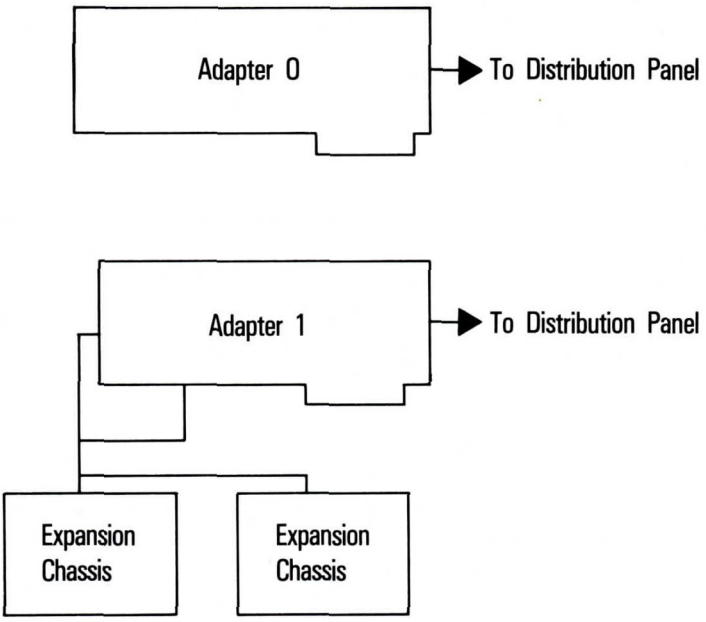
An expansion chassis is a protective housing. It encases a printed circuit board containing expansion slots and, in some cases, on-board devices similar to those on the adapter in the computer. You can install expansion adapters in the expansion slots in much the same way as you install adapters in your IBM Personal Computer.

> **Note:** Do not use the Data Acquisition and Control Adapter in an expansion chassis, and do not use an expansion chassis adapter in the computer. Any attempt to do so may result in damage to the adapter, the chassis, or both.

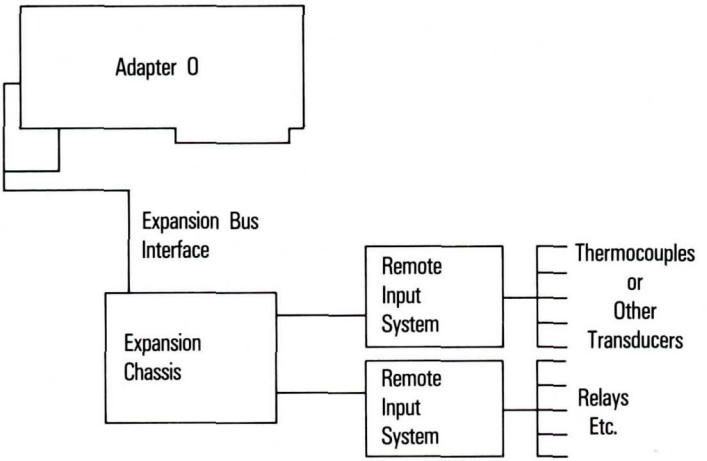
With few exceptions, an expansion chassis should be supported with an auxiliary power supply. While the IBM Personal Computer's power supply adequately meets the needs of the adapter's on-board devices, the addition of several expansion devices can put a severe load on it. By equipping your expansion chassis with an auxiliary power supply, you can ensure adequate power to support your expansion devices.

The diagram below shows some typical system configurations that use adapters and expansion devices.

Configuration 1



Configuration 2



Expansion Devices

Expansion devices are plug-in modules that increase the capabilities of the internal adapter. Functions provided for expansion devices include analog input, analog output, binary I/O, and counter I/O. Some expansion devices are functionally equivalent to the internal on-board devices. Others provide such enhancements as additional channels, higher speed, greater resolution, and special-purpose signal conditioning.

Adding More Binary Ports

In addition to its on-board binary I/O device, your system can access up to 253 additional binary I/O devices through the expansion interface.

Adding More Analog Channels

A variety of expansion interface analog input modules allow access to more channels of analog input. These modules can include enhanced capability for input dividing and input gain; this allows you to input signals over a wide range. The software supports enhanced analog input devices by using the AI function control (ctrl) argument. For more information on this support, see "Accessing Enhanced AI Devices" in this appendix.

Analog Output (AO) expansion modules, which provide extra channels of output, are also supported on the expansion bus. However, no enhanced device support is provided for AO functions. The ctrl argument is not defined for these functions.

Thermocouples and Other Transducers

Thermocouples and other transducers, such as strain gauges, and Resistance Temperature Detectors (RTD), often require special interface hardware. This hardware provides such things as cold-junction compensation, galvanic isolation, excitation voltage, impedance transformation, and other types of signal conditioning. An expansion bus interface remote input system provides all of the necessary interface hardware for several popular types of devices. Among these are J, K, and T thermocouples, as well as the devices mentioned above. The system places the functions of this hardware under program control.

One of the more useful results of this arrangement is that data can be acquired from large numbers of dissimilar transducers. Because the interface hardware converts each transducer's output into a signal in the +/−5 Volt range, the programmer does not need to take the electrical characteristics of each transducer into consideration.

Improving A/D Resolution

With the 12-bit resolution of the adapter's analog input or output device, you can resolve to better than one part in 4000. This equates to an effective resolution of 0.0025 Volts over a 10-Volt range. If you need better resolution, consider using an expansion bus interface analog device that has a 14 or 16-bit A/D converter. A 16-bit A/D converter, for example, resolves to better than one part in 65 000, equivalent to 0.0001 Volts over a 10-Volt range.

Accessing Expansion Devices

You can access any expansion device using the same functions that access the on-board devices. By changing the device number (device) argument, you can tell any function to access any device on the expansion bus interface of a particular adapter.

The Data Acquisition and Control System software provides for the easy incorporation of expansion devices. For example, it recognizes up to 253 expansion devices with a capability of 256 channels for each device.

Accessing Enhanced AI Devices

Enhanced analog input (AI) devices have specific features that you control through generalized function registers. These specific features can include:

- Programmable gain
- J, K, and T thermocouple support
- Resistance temperature detector (RTD) support
- Strain gauge support.

The software supports additional input function capability by using the control (ctrl) argument described in Chapter 4. The value in the ctrl argument is used to program the function registers.

Enhanced AI Device Control

The enhanced AI device is the same as the analog input device described in the IBM Personal Computer *Technical Reference* with the addition of two registers. These registers are:

- Function 1 register
- Function 2 register.

The enhanced AI devices use the same register format as the on-board devices. The device number and the Function 1 and 2 registers are the only difference. For more information on register formats, see "Programming Considerations" in the Technical Reference manual.

Enhanced AI Device Registers

The enhanced AI device has the following registers:

Register	Read/Write	Name	Function
0	Write	AI Control	Setup and control register
0	Read	AI Status	Returns status of the hardware
1	Write	Function 1	Enhanced Features Support
2	Write	Function 2	Enhanced Features Support
2	Read	AI Data	Returns current analog status

Function 1 Register Description

The following table describes the Function 1 register.

Bits	Byte	Function
0 to 7	Low	General register to provide support for enhanced features. The value is derived from non-zero ctrl argument low byte minus 1. (0 to 254 range)
8 to 15	High	Not used

Function 2 Register Description

The following table describes the Function 2 register.

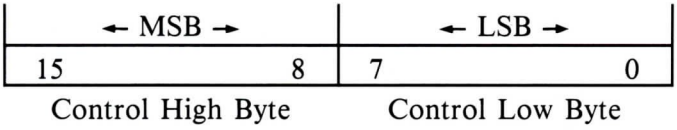
Bit	Byte	Function
0 to 7	Low	General register to provide support for enhanced features. Value derived from non-zero ctrl argument high byte minus 1. (0 to 254 range)
8 to 15	High	Not used

Control Argument (ctrl) Processing

The control argument processing is designed to support capabilities of expansion devices. The software processing of the control argument (ctrl) and the software analog input function strategy is described on the following pages.

Processing Description

The following describes the operations performed when the value of the control argument is read:



Low Byte Processing

When the low byte equals zero, no additional write operation is performed. When the low byte does not equal zero, the value of the low byte minus 1 is written to the Function 1 register.

High Byte Processing

When the high byte equals zero, no additional write operation is performed. When the high byte does not equal zero, the value of the high byte minus 1 is written to the Function 2 register.

Programming Strategy

The following is the strategy that the device driver uses when accessing an enhanced analog input device.

START: ANALOG INPUT STRATEGY

- Set the device number in the device-number register.
- If ctrl = 0, jump to NormalAin. If ctrl low byte > 0, set the Function 1 register with the ctrl low byte value minus 1. If ctrl high byte > 0, set the Function 2 register with the ctrl high byte value minus 1.
- Delay 1.

NormalAin:

- Set the channel number without setting the CONVERT START bit in the AI control register.
- Delay 2.
- Set the same channel number now with the CONVERT START bit set.
- Delay 3.
- Read the BUSY BIT in the AI Status register and wait for a complete A/D conversion.
- If the conversion is not complete within a required time, exit the routine with a timeout error.
- When conversion is complete, set the channel number without setting the CONVERT START bit.
- Input the data.

END: ANALOG INPUT STRATEGY

Appendix D. Performance Considerations

The Data Acquisition and Control System gives you the flexibility to select I/O data rates to support a wide range of applications. The software can process multiple I/O functions in optional configurations that provide flexibility and data integrity to meet your performance requirements.

You can extend the device driver software capabilities by using the special execution mode options. The following data rates are possible:

- Multiple analog input function in excess of 16 000 samples per second
- Multiple analog output and binary I/O function in excess of 17 000 samples per second.
- Multiple counter input function in excess of 12 000 samples per second.

The mode argument is used in conjunction with the rate argument to provide performance alternatives for the applications developer.

The performance of multiple I/O functions are shown in the following pages as *nominal* and *extended*. The configuration tables give you the options and results expected depending on what values you choose. Note that the functions AOUM and CINM are not supported in the Extended Performance Configurations.

You have the option to specify a mode of zero for any range value with the understanding that timer overruns are likely to occur for values in excess of 1500 samples per second. If mode equals zero and you are sampling above 1500 samples per second, the value of the status variable may be invalid. This is due to sampling-iteration processing of system interrupts or interrupts supporting other system devices and adapters.

If mode equals 128 and the multiple I/O function processing is complete, all system interrupts are enabled and normal system processing continues.

Nominal Performance Configurations for the AINM, AOUM, BINM, BOUM, and CINM Functions

Rate Argument Range	1 to 1499 (samples/sec)		1500 and above for AOUM, CINM 1500 to 7999 for AINM, BINM, BOUM	
Mode Argument Value	0	128	0	128
AINM AOUM BINM BOUM CINM	All system interrupts area enabled. Iterative function sampling supported by adapter/timer interrupt.	All system interrupts are disabled except adapter level. Iterative function sampling supported by adapter/timer interrupt.	All system interrupts are enabled. Adapter/timer is poled to support iterative function sampling.	All system interrupts are disabled. Adapter/timer is poled to support iterative function sampling.

Extended Performance Configurations for the AINM Functions

Rate Argument Range	8000 to 12499 (samples/sec)		12500 and above (samples/sec)	
Mode Argument Value	0	128	0	128
AINM	All system interrupts area enabled. Adapter/timer is poled to support iterative function sampling.	All system interrupts are disabled. Iterative function sampling supported by adapter/timer interrupt. Following the initial read, the device response timeout check is disabled. Status variable updating is disabled. (status = 0 is returned)	All system interrupts are enabled. Adapter/timer is poled to support iterative function sampling.	All system interrupts are disabled. A/D device is read at the maximum rate possible for the specified value in the count argument. Following the initial read, the device response timeout check is disabled. Status variable updating is disabled. (status = 0 is returned)

Extended Performance Configurations for the BINM and BOUM Functions

Rate Argument Range	8000 and above (samples/sec)	
Mode Argument Value	0	128
BINM BOUM	All system interrupts area enabled. Adapter/timer is poled to support iterative function sampling.	All system interrupts are disabled. Adapter/timer is poled to support iterative function sampling. Following the initial read, the device response timeout check is disabled. Status variable updating is disabled. (status = 0 is returned) Masking of the I/O values is disabled. Handshaking using the STROBE and CTS lines is disabled.

Glossary

A/D. Analog-to-Digital. The conversion of data from analog form to digital form (A/D Conversion).

analog signal. A signal whose measured value is a continuously variable physical quantity.

device, I/O. Any input or output hardware. The adapter has three I/O devices: the Analog I/O device, the Digital I/O device, and the Counter device. An expansion chassis connected to the expansion bus interface allows the addition of expansion devices.

D/A. Digital-to-Analog. The conversion of data from digital form to analog form (D/A Conversion).

digital signal. A discrete or discontinuous signal; one whose various states are discrete intervals apart.

expansion bus interface. An expansion bus for the adapter allowing communication with external I/O devices.

handshaking. A scheme for signaling readiness of an I/O device to input or output data.

iterative functions. Those functions that execute more than once. They are also called *multiple* functions.

jitter. Inconsistency of spacing between samples.

masks. A pattern of characters that is used to control the retention or elimination of portions of another pattern of characters.

module, I/O. An external I/O device.

noise. Any extraneous component present in a signal. It attaches itself to the signal either at point of origin or during transmission. It is most often caused by proximity of high-voltage alternating current.

non-iterative functions. Those functions that execute only once. They are also called *simple* functions.

range. The difference between the highest and lowest value that a quantity or function can assume.

sample. A single piece of input data that can be stored or used by the Personal Computer.

sensor-based. Pertaining to the use of sensing devices, such as transducers or sensors, to monitor a physical process.

skew. The true length of time between a number of apparently simultaneous samples.

TTL. Transistor-Transistor Logic. This indicates the level of a binary signal. The two levels are TTL low, which is less than or equal to 0.4 volts, and TTL high, which is greater than or equal to 2.4 volts.

word. A 16-bit binary value.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Continued from inside front cover

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassette(s) on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

LIMITATIONS OF REMEDIES

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette(s) or cassette(s) not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized IBM PERSONAL COMPUTER dealer with a copy of your receipt, or
2. if IBM or the dealer is unable to deliver a replacement diskette(s) or cassette(s) which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT WILL IBM BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF IBM OR AN AUTHORIZED IBM PERSONAL COMPUTER DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of Florida.

Should you have any questions concerning this Agreement, you may contact IBM by writing to IBM Personal Computer, Sales and Service, P.O. Box 1328-W, Boca Raton, Florida 33432.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

International Business
Machines Corporation
P.O. Box 1328-S
Boca Raton, Florida 33432

Printed in the
United States of America

6137681