9010A Language Compiler

P/N 661504
December 1983

FLUKE®

NOTICE

This manual describes unpublished Software which contains the trade secrets and confidential proprietary information of John Fluke Mfg. Co., Inc. and which embodies substantial creative effort, ideas, and expressions. THE SOFTWARE IS PROVIDED UNDER LICENSE FROM FLUKE. Fluke grants Licensee a perpetual non-exclusive license to use this material and make up to three copies for backup purposes without written permission from Fluke.

THIS SOFTWARE IS LICENSED FOR USE ON A SINGLE COMPUTER SYSTEM.

LIMITED WARRANTY

Fluke warrants that the Software has been properly recorded on non-defective diskettes. Fluke does not warrant the Software to be error free. Fluke will replace such diskettes without charge if Fluke in good faith determines that such diskettes were not subject to misuse and if returned to a Fluke Technical Service Center, within ninety (90) days of shipment. Refer to your 9010A Operator Manual for a listing of locations. Fluke reserves the right to change the specifications and operating characteristics of the Software it produces, over a period of time, without notice.

FLUKE GRANTS NO OTHER WARRANTIES, EITHER EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL FLUKE BE LIABLE FOR ANY LOSS OF DATA, USE, PROFITS OR GOODWILL, OR FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR OTHER SIMILAR DAMAGES AS A RESULT OF ANY MATTER RELATED TO THIS AGREEMENT, REGARDLESS OF THE FORM OF THE ACTION.

Contents

(continued on page ii)

(continued on page iii)

(continued on page iv)

(continued on page v)

APPENDICES

INDEX

Section 1
Introduction

CONTENTS

INTRODUCTION TO THE 9010A LANGUAGE COMPILER

The 9010A Language Compiler package is used to create test and troubleshooting programs for the Fluke 9010A Micro-System Troubleshooter.

The 9010A is an excellent tool for interactive troubleshooting, and many users may want to take advantage of its power by writing extensive test programs. While the 9010A itself is very convenient for entering relatively short programs, it may be advantageous to create and maintain large, elaborate, or complex programs using a host computer's editing and file management facilities. The 9010A Language Compiler allows 9010A programs to be developed conveniently on a host computer system and then transferred to the 9010A for execution.

THE HOST COMPUTER SYSTEM

The illustration shows the 9010A connected by an RS-232-C serial interface to a host computer system. Fluke currently supports the 9010A Language Compiler on the following computer systems:

- Fluke 1720A and 1722A Instrument Controllers
- Most Z80 CP/M systems with 8-inch disk drives
- Kaypro II
- IBM Personal Computers (PC and XT)

Registered Trademarks:

Z-80: Zilog

CP/M: Digital Research Inc.

Kaypro: Kaypro Corp.

IBM: International Business Machines

HOW THE COMPILER WORKS

The 9010A is able to read and write test programs via its auxiliary RS-232-C interface. The entire contents of the 9010A program memory, including setup parameters and address space descriptors, are transferred through the serial interface in a special hex data format. The 9010A Language Compiler takes advantage of this ability of the 9010A to read programs in hex format.

The test programmer develops the test programs on the host computer system in an ASCII source program form using the full power of the editing and file storage capabilities of the host system. In this sense, a 9010A program on the host system is much like a program written in any other programming language, such as BASIC, FORTRAN, or Pascal.

Once the program has been written in source form, the 9010A Language Compiler program converts the source program into the hex format required for transfer to the 9010A.

The program is then transferred to the 9010A using a transfer program that is supplied with the compiler package, and the hex format program is read into Troubleshooter memory by pressing the AUX I/F and READ keys on the 9010A.

LANGUAGE EXTENSIONS

The 9010A Language Compiler accepts any program that can be entered through the 9010A keyboard. In fact, the syntax of the 9010A Language is compatible with program listings obtained from the 9010A using the AUX I/F SETUP, AUX I/F LEARN, and AUX I/F PROGM commands described in the 9010A Operator Manual.

In addition to the standard 9010A commands, however, the 9010A Language Compiler provides some powerful extensions. These additional features are designed to make it much easier to develop and maintain large 9010A programs. Some of the key features are:

• Program Comments	Allows the test programmer to incorporate documentation into the program itself
• Keyword Abbreviations, Optional Command Keywords, and Shorthand Notation for Unary Operators	Minimizes the typing required to enter a test program on the host system
• File Inclusion	Permits common programs to be conveniently shared by many source files, reducing the time required to develop test programs for new applications
• Symbolic Names for Programs, Labels, and Registers	Allow programs to be written more clearly, making them easier to understand and maintain

THE 9010A LANGUAGE COMPILER PACKAGE

The 9010A Language Compiler package consists of this manual and a diskette that contains several programs and data files. The key software elements of the package are as follows:

Disk Verification Program

The Disk Verification Program is a utility program that verifies the integrity of compiler package files. This program is used to assure that there are no files missing, that the files are not corrupted, and that they are compatible versions.

Compiler

The compiler is a program that accepts the source file representation of 9010A programs, including setup parameters and address descriptors, and produces a corresponding hex format file that can be read into the 9010A.

The compiler checks for coding errors in the source file and displays an error message whenever an error is detected. If the source file contains errors, then a hex file is not created.

In addition to the hex format output file, the compiler can produce a listing file containing a modified copy of the source file. The listing file can be requested in several optional formats that make the processing performed by the compiler more visible to the test programmer.

File Transfer Program

The compiler package contains a utility program that is used to transfer 9010A programs between the host system and the 9010A. The primary purpose of the file transfer program is to transfer hex files produced by the compiler to the 9010A for execution, but it can also be used to transfer programs from the 9010A to the host system.

Pod Data Files

Some of the Setup commands of the 9010A Language refer to information that is specific to particular 9010A interface pods. Pod-specific information includes the enableable forcing lines, bus test address, and RUN UUT address.

The 9010A Language Compiler package contains a pod data file for each interface pod currently available from Fluke. The pod data files provide the information required by the compiler to process the pod-specific Setup commands.

By simply creating new pod data files, the compiler can be updated to accommodate new pods which are developed in the future.

USE WITH THE 9005A

Hex files that are produced by the 9010A Language Compiler are compatible with the 9005A as well as the 9010A. However, programs that are transferred from the host system to a 9005A cannot be edited on the 9005A, nor can they be written to a cassette tape as they can with a 9010A.

Section 2
How to Use This Manual

CONTENTS

INTRODUCTION

This manual is the reference source for the 9010A Language Compiler and the 9010A Language. It is written with the assumption that the reader is already familiar with the operation of both the 9010A Micro-System Troubleshooter and the host computer system.

If you are not familiar with the 9010A, you should refer to the 9010A Operator Manual and the 9010A Programming Manual and learn how to use the 9010A before proceeding in this manual. Of course, if you are not familiar with the host computer system, you should read the instruction manuals provided with your system.

ORGANIZATION

The 9010A Language Compiler User Manual is divided into the following sections:

1. INTRODUCTION	Introduces the 9010A Language Compiler and the 9010A Language and describes basic features.
2. HOW TO USE THIS MANUAL	Describes the sections of the manual and recommends how each section should be used.
3. GETTING STARTED	Describes what you need to get started using the 9010A Language Compiler with your particular computer.
4. WRITING PROGRAMS	Gives an overview of the 9010A Language and describes how to create 9010A source files.
5. USING THE COMPILER	Describes how to use the compiler and the file transfer program.
6. LANGUAGE REFERENCE	Provides detailed information on the 9010A Language syntax in a quick-reference format.
APPENDICES A-F	Provides other information about the 9010A Language.

SUGGESTED USE

The sections in this manual appear in the order in which they are intended to be read by a first-time user of the 9010A Language Compiler. Section 1, Introduction, provides an overview of the features of the 9010A Language Compiler. If you are a first-time user of the compiler, the introduction gives you an idea of what to expect.

This section, How to Use this Manual, provides guidance in using the manual so that you can quickly and correctly begin to use the 9010A Language Compiler.

Section 3, Getting Started, provides you with the information you need to get your new compiler running. Before attempting to use the compiler, it is essential that you read this section thoroughly so that you can avoid problems. Getting Started shows you how to set up your host computer system and how to connect it to the 9010A.

Section 4, Writing Programs, uses explanations and examples to introduce you to the 9010A Language and demonstrates how to create 9010A program source files. Everyone should read this section at least once. When you become more familiar with the 9010A Language, you will rely less on Section 4 and more on Section 6.

Section 5, Using the Compiler, provides information on how to use the compiler and the file transfer program. This will enable you to create hex files and transfer them to the 9010A for execution.

Section 6, Language Reference, contains much of the same information as Section 4, but the information is more detailed, and it is organized to enable quick reference. This section is designed for use when you are in the middle of a program and need specific syntax information in a hurry.

Appendices A through F provide detailed information about the 9010A Language. You will probably use the appendices for quick reference after you have learned how to use the language.

Section 3
Getting Started

CONTENTS

INTRODUCTION

This section provides the information needed to set up your host computer system to work with the 9010A Language Compiler. For each version of the compiler, the following information is presented:

- What You Need — Describes the hardware configuration required to use the compiler package
- Backing Up the Program Diskette — Provides the information needed to create a working copy of the program diskette
- Hooking Up the System — Describes how to connect the 9010A to the host system and how to set the RS-232-C serial interface parameters
- System Dependencies — Presents other information that is unique to a particular host system

You should carefully read the instructions that apply to your host system. It is not necessary for you to read the material that relates to other host systems.

FLUKE 1720A INSTRUMENT CONTROLLER

Introduction

The following information applies to the 1720A version of the 9010A Language Compiler.

What You Need

The following equipment is needed in order to use the compiler package:

1. Fluke 9005A or 9010A Micro-System Troubleshooter with Option 9010A-001, RS-232-C Interface
2. Fluke 1720A Instrument Controller (Option 1720A-001, 128K-Byte E-Disk is recommended.)
3. Fluke Y1705 RS-232-C Null Modem Cable and Y1707 RS-232-C Interface Cable
4. 9010A-920 9010A Language Compiler, 1720A/1722A Version

Backing Up the Program Diskette

The 9010A Language Compiler package consists of this manual and a write-protected program diskette containing the compiler itself and various other programs and data files.

Before using the compiler, you should make a copy of the program diskette. This copy should be used for normal day-to-day operations, while the original program diskette should be kept in a safe place as a backup so that the working copy can be restored if it is ever damaged.

Complete instructions on how to copy diskettes can be found in the 1720A File Utility User Manual.

Verifying the Working Diskette

Once you have created a working copy of the program diskette, you should verify the integrity of its files by running VERIFY, one of the programs included in the compiler package. To run the VERIFY program, type

VERIFY⟨RETURN⟩

in response to the 1720A Console Monitor program prompt.

The VERIFY program checks the contents of the 1720A System Device (SY0:) to verify the integrity of the Compiler package files. It calculates a checksum for each of the files and compares it to the checksum contained in the VERIFY.DAT file. VERIFY.DAT is an ASCII file that contains a list of filenames and checksums for each of the files in the compiler package.

Results from the VERIFY program are printed in tabular form as each file is checked. Missing files or checksum errors (that could indicate either corrupted files or incorrect version numbers) are reported. If such problems occur, recopy the diskette and run the VERIFY program again. If problems persist and you are unable to run any of the programs, contact a Fluke Technical Service Center.

Hooking Up the System

The 1720A must be connected to the 9010A whenever you want to transfer the hex files produced by the compiler to the 9010A for execution.

1. Use an RS-232-C interface cable and an RS-232-C null modem cable to connect the auxiliary interface of the 9010A to one of the serial ports on the 1720A.

   KB1: or KB2: can be chosen as the serial port on the 1720A. XFER, the file transfer program described in Section 5, Compiler Usage, allows you to specify the port name to be used when transferring files to the 9010A.

   Since XFER defaults to KB1:, it is more convenient to connect the 9010A to KB1: if KB1: is not already being used for some other purpose.

2. Set the RS-232-C auxiliary interface parameters on the rear panel of the 9010A. Suggested settings are:

   9600 baud (switch setting 7)
   Parity: even
   Data bits: 8
   Stop bits: 1
   Parity: on

   9010A Setup parameter NEWLINE must be set to 00000D0A (the 9010A default value) for transferring files.

3. Set the parameters of the serial port on the 1720A to correspond to those of the 9010A. SET, a 1720A system program, is included on the program diskette for this purpose. Refer to the 1720A Set RS-232-C Utility User Manual for a complete description of how to use the SET utility.

NOTE

The STALL option must be enabled on the 1720A if any files are to be transferred from the 9010A to the 1720A. This option is not required if files are only transferred from the 1720A to the 9010A.

Some early versions of the 1720A Set RS-232-C Utility program do not implement the STALL option. Be sure to use the Set RS-232-C Utility program that is contained in the 9010A Language Compiler package.

The End of Line character should be set to 10 and the End of File character should be set to 26 (the 1720A default values).

The following example demonstrates how the SET utility can be used to select the parameters that correspond to the above 9010A settings.

```
#SET
*KB1: BR 9600 DB 8 PB E SB 1 SI E SO E
*EX
```

Since the 1720A serial port parameters must be reestablished every time the 1720A is turned on, you will probably want to incorporate the necessary commands into a system command file. The 1720A Floppy Disk Operating System User Manual contains information on how this is done.

System Dependencies

Text Editor

In order to create and maintain source files on the host system, a general-purpose text editor is required. The Editor Accessory program (filename ESX) is the recommended editor for use with the 1720A.

A copy of the Editor Accessory program is included on the program diskette, and a copy of the Editor User Manual is included with the compiler package.

Disk Space

After using the Editor or Compiler programs, it may be advantageous to pack the disk contents, using the /P option in the 1720A File Utility program, to provide as much free disk space as possible. Refer to the 1720A File Utility User Manual if you need help with packing the disk.

The message

```
?Read/write past physical end of file
```

means that there was not enough contiguous disk space to create the output files. Delete any unnecessary files, pack the disk, and try again.

Compiler Organization

The Compiler program is constructed of overlayed program segments, some of which must be loaded during program execution. Therefore, if the Compiler program is being used from a floppy disk, the disk must remain in the disk drive while the program is running. Do not remove the disk until the program is finished.

If the overlays are not available when needed, the fatal error message

```
!Unable to load overlay
```

will be displayed.

FLUKE 1722A INSTRUMENT CONTROLLER

Introduction

The following information applies to the 1722A version of the 9010A Language Compiler.

What You Need

The following equipment is needed in order to use the compiler package:

1. Fluke 9005A or 9010A Micro-System Troubleshooter with Option 9010A-001, RS-232-C Interface.

2. Fluke 1722A Instrument Controller.

3. Fluke Y1705 RS-232-C Null Modem Cable and Y1707 RS-232-C Interface Cable.

4. Fluke 9010A-920 9010A Language Compiler, 1720A/1722A Version.

Backing Up the Program Diskette

The 9010A Language Compiler package consists of this manual and a write-protected program diskette containing the compiler itself and various other programs and data files.

Before using the compiler, you should make a copy of the program diskette. This copy should be used for normal day-to-day operations, while the original program diskette should be kept in a safe place as a backup so that the working copy can be restored if it is ever damaged.

Complete instructions on how to copy diskettes can be found in the 1722A System Manual.

Verifying the Working Diskette

Once you have created a working copy of the program diskette, you should verify the integrity of its files by running VERIFY, one of the programs included in the compiler package. To run the VERIFY program, type

VERIFY ⟨RETURN⟩

in response to the 1722A FDOS prompt.

The VERIFY program checks the contents of the 1722A System Device (SY0:) to verify the integrity of the compiler package files. It calculates a checksum for each of the files and compares it to the checksum contained in the VERIFY.DAT file. VERIFY.DAT is an ASCII file that contains a list of filenames and checksums for each of the files in the compiler package.

Results from the VERIFY program are printed in tabular form as each file is checked. Missing files or checksum errors (that could indicate either corrupted files or incorrect version numbers) are reported. If such problems occur, recopy the diskette and run the VERIFY program again. If problems persist and you are unable to run any of the programs, contact a Fluke Technical Service Center.

Hooking Up the System

The 1722A must be connected to the 9010A whenever you want to transfer the hex files produced by the compiler to the 9010A for execution.

1. Use an RS-232-C interface cable and an RS-232-C null modem cable to connect the auxiliary interface of the 9010A to the serial port on the 1722A.

2. Set the RS-232-C auxiliary interface parameters on the rear panel of the 9010A. Suggested settings are:

   9600 baud (switch setting 7)
   Parity: even
   Data bits: 8
   Stop bits: 1
   Parity: on

The 9010A Setup parameter NEWLINE must be set to 00000D0A (the 9010A default value) for transferring files.

3. Set the parameters of the serial port on the 1722A to correspond to those of the 9010A. The Set Utility program (SET), a 1722A system program, is included on the program diskette for this purpose. Refer to the 1722A System Manual for a complete description of how to use the SET utility.

NOTE

The STALL option must be enabled on the 1722A if any files are to be transferred from the 9010A to the 1722A. This option is not required if files are only transferred from the 1722A to the 9010A.

The End of Line character should be set to 10 and the End of File character should be set to 26 (the 1722A default values).

The following example demonstrates how the SET utility can be used to select the parameters that correspond to the above 9010A settings.

```
#SET
*KB1: BR 9600 DB 8 PB E SB 1 SI E SO E
*EX
```

Since the 1722A serial port parameters must be reestablished every time the 1722A is turned on, you will probably want to incorporate the necessary commands into a system command file. The 1722A System Manual contains information on how this is done.

System Dependencies

Text Editor

In order to create and maintain source files on the host system, a general-purpose text editor is required. The Editor Accessory program (filename EDIT) is the recommended editor for use with the 1722A.

A copy of the Editor Accessory program is included on the program diskette, and instructions for using the editor are included as an Addendum to this manual.

Disk Space

After using the Editor or Compiler programs, it may be advantageous to pack the disk contents, using the /P option in the 1722A File Utility program, to provide as much free disk space as possible. Refer to the 1722A System Manual if you need help with packing the disk.

The message

```
?Read/write past physical end of file
```

means that there was not enough contiguous disk space to create the output files. Delete any unnecessary files, pack the disk, and try again.

IBM PERSONAL COMPUTER

Introduction

The following information applies to the IBM Personal Computer (PC) version of the 9010A Language Compiler.

What You Need

The following equipment is needed in order to use the compiler package:

1. 9005A or 9010A Micro-System Troubleshooter with Option 9010A-001 RS-232 Interface.
2. IBM Personal Computer (model PC or XT) with:
   a. A monochrome or color display.
   b. Version 1.1 or 2.0 of the IBM DOS Operating System.
   c. At least 128 K bytes of RAM.
   d. A disk drive. We recommend using two disk drives or a fixed Winchester technology disk drive.
   e. An RS-232-C interface.
3. Fluke Y1705 RS-232-C Null Modem Cable and Fluke Y1707 RS-232-C Interface Cable.
4. Fluke 9010A-923 9010A Language Compiler (IBM PC version).

Backing Up the Program Diskette

The 9010A Language Compiler package consists of this manual and a write-protected program diskette containing the compiler itself and various other programs and data files.

Before using the compiler, you should make a copy of the write-protected program diskette. This copy is used for normal day-to-day operations, while the original program diskette should be kept in a safe place as a backup so that the working copy can be restored if it is ever damaged.

Complete instructions on how to copy diskettes can be found in the IBM Disk Operating System (DOS) User Manual.

Verifying the Working Diskette

Once you have created a working copy of the program diskette, you should verify the integrity of its files by running VERIFY, one of the programs included in the compiler package. To run the VERIFY program, put the working diskette in drive a: and then type

a:VERIFY ⟨RETURN⟩

in response to the IBM system prompt.

The VERIFY program checks the contents of the copy to verify the integrity of the compiler package files. It calculates a checksum for each of the files and compares it to the checksum contained in the VERIFY.DAT file. VERIFY.DAT is an ASCII file that contains a list of filenames and checksums for each of the files in the compiler package.

Results from the VERIFY program are printed in tabular form as each file is checked. Missing files or checksum errors (that could indicate either corrupted files or incorrect version numbers) are reported. If such problems occur, recopy the diskette and run the VERIFY program again. If problems persist and you are unable to run any of the programs, contact a Fluke Technical Service Center.

Hooking Up the System

The IBM PC must be connected to the 9010A whenever you want to transfer the hex files produced by the compiler to the 9010A for execution.

1. Use an RS-232 interface cable and an RS-232 null modem cable to connect the auxiliary interface of the 9010A to a serial port on the IBM PC.

2. Set the RS-232 auxiliary interface parameters on the rear panel of the 9010A. Suggested settings are:

   2400 baud (switch setting 5)
   Parity: On
   Data bits: 8
   Stop bits: 1
   Parity: Even

3. Set the parameters of the serial port on the IBM PC to correspond to those of the 9010A.

   You may use the IBM MODE command to configure the serial port.

   Refer to the IBM instruction manuals for help.

4. The NEWLINE setup parameter should be set to 10000D0A for transferring files. If transmission errors occur, it may be necessary to change the timing delay to a larger value. See the 9010A Operator Manual for more information.

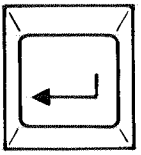
5. The 9010A setup parameters STALL and UNSTALL should be set to 13 and 11 respectively (the 9010A default values) when transferring files.

System Editor

In order to create and maintain source files on the host system, a general-purpose text editor is required. Any general-purpose editor may be used with 9010A language source files.

RETURN Key

References to the RETURN key in this manual refers to the



key on the IBM Personal Computers.

CP/M OPERATING SYSTEMS

Introduction

The following information applies to the version of the 9010A Language Compiler for CP/M systems.

CP/M (Control Program for Microcomputers) is a product of Digital Research, Inc. It is a general-purpose operating system that runs on a wide variety of host computers.

What You Need

The following equipment is needed in order to use the compiler package with a host computer running the CP/M operating system:

1. 9005A or 9010A Micro-System Troubleshooter with Option 9010A-001 RS-232 Interface

2. CP/M compatible Z80 based host computer system with:

   a. At least one eight-inch IBM 3740 format disk drive. We recommend using two disk drives.

   b. Standard CP/M Operating System software (version 2.2).

   c. An RS-232-C interface.

3. An RS-232-C Interface Cable suitable for connecting your host computer system to the 9010A. For example, use a Fluke Y1709 RS-232-C Interface Cable to connect a Kaypro II Personal Computer to a 9010A.

4. Fluke 9010A-921, the version of the 9010A Language Compiler package for CP/M on eight inch disks, or 9010A-922, the version for the Kaypro II Personal Computer with CP/M on a 5-1/4 inch disk.

Backing Up the Program Diskette

The 9010A Language Compiler package consists of this manual and a write-protected program diskette containing the compiler itself and various other programs and data files.

Before using the compiler, you should make a copy of the write-protected program diskette. This copy is used for normal day-to-day operations, while the original program diskette should be kept in a safe place as a backup so that the working copy can be restored if it is ever damaged.

Verifying the Working Diskette

Once you have created a working copy of the program diskette, you should verify the integrity of its files by running VERIFY, one of the programs included in the compiler package. To run the VERIFY program, type

⟨VERIFY RETURN⟩

in response to the CP/M system prompt.

The VERIFY program checks the contents of the copy to verify the integrity of the compiler package files. It calculates a checksum for each of the files and compares it to the checksum contained in the VERIFY.DAT file. VERIFY.DAT is an ASCII file that contains a list of filenames and checksums for each of the files in the compiler package.

Results from the VERIFY program are printed in tabular form as each file is checked. Missing files or checksum errors (that could indicate either corrupted files or incorrect version numbers) are reported. If such problems occur, recopy the diskette and run the VERIFY program again. If problems persist and you are unable to run any of the programs, contact a Fluke Technical Service Center.

Hooking Up the System

The host computer must be connected to the 9010A whenever you want to transfer the hex files produced by the compiler to the 9010A for execution.

1. Use an RS-232 interface cable to connect the auxiliary interface of the 9010A to a serial port on the host computer.

2. Set the RS-232 auxiliary interface parameters on the rear panel of the 9010A. Suggested settings are:

   9600 baud (switch setting 7)
   Parity: Even
   Data bits: 8
   Stop bits: 1
   Parity: on

3. The NEWLINE setup parameter should be set to 10000D0A for transferring files. If transmission errors occur, it may be necessary to change the timing delay to a larger value. See the 9010A Operator Manual for more information.

4. The 9010A setup parameters STALL and UNSTALL should be set to 13 and 11 respectively (the 9010A default values) when transferring files.

5. Set the parameters of the serial port on the host computer to correspond to those of the 9010A.

   Refer to Installing Software in this section for further information on setting the RS-232 parameters.

Installing Software

On CP/M systems, the File Transfer utility program (XFER) uses information from a data file for configuring RS-232-C transfers. This file, CONFIG.PRT, is automatically created for each system the first time that the File Transfer program is used.

The program will prompt for information about RS-232-C port parameters, and use the information that you enter to create the data file on the system default disk.

Refer to the host computer's instruction manuals if you need further information to answer the prompts.

Once the CONFIG.PRT data file is available on the disk, it will automatically be used for subsequent file transfers with the XFER program. This file contains port status and data addresses, an optional baud rate address, and SIO initialization bytes.

To change the RS-232-C configuration in the CONFIG.PRT file, use the Configure option (C) in the File Transfer program. The prompts will be repeated to allow you to redefine the configuration.

Note that the CONFIG.PRT file will be created on the system default device. The system disk must not be write-protected at this time.

Editor

In order to create and maintain source files on the host system, a general-purpose text editor is required. Any general-purpose editor may be used with 9010A Language source files.

Section 4
Writing Programs

CONTENTS

INTRODUCTION

This section provides the information you need to write programs for the 9010A Language Compiler. The section is divided into three parts. Each part is self-contained and describes increasingly more advanced features of the 9010A Language.

The three parts cover the following topics:

PART 1: GENERAL PROGRAM FORMAT	Describes how to write simple programs using the standard features of the 9010A Language
PART 2: CODING SHORTCUTS	Introduces some extended features of the 9010A Language which reduce the amount of typing required to enter programs
PART 3: SYMBOLIC NAME	Allows programs to be made more readable and easier to maintain by using mnemonic names for programs, labels, and registers

The best way to learn the 9010A Language is to start by reading through Part 1 of this section, and then skip directly to Section 5, Using the Compiler. You should use the compiler to compile the example programs provided in Part 1, and then try writing some simple programs of your own.

Once you feel comfortable with the concepts covered in Part 1, you can return at any time to this section and proceed with the more advanced concepts covered in the remaining parts. The compiler can be used productively at any of the three levels.

PART 1: GENERAL PROGRAM FORMAT

Introduction

The 9010A Language Compiler allows you to create source files identical to those that the 9010A AUX I/F functions send via the RS-232-C auxiliary interface. These files can contain the entire contents of the 9010A memory – not only 9010A programs but also any available address space and setup information.

In source files for the 9010A Language Compiler, address space information, setup information, and programs are described in separate blocks. These blocks are identified with compiler keywords, such as SETUP INFORMATION. This section provides information about using the various blocks and shows some sample source files.

In the 9010A language, program statements use an expanded syntax to take advantage of the flexibility of the host system text editor and to provide enhanced readability. Program lines may contain comments and symbolic names. More information about source files and program lines is found throughout this section. Section 6, Language Reference, contains detailed information on the syntax and usage of each 9010A program statement.

The following is an example of a short source file containing two valid 9010A programs and no address space or setup information:

```
PROGRAM 0

   DPY-THIS IS AN EXAMPLE
   EXECUTE PROGRAM 10
   DPY-OF A VALID 9010A PROGRAM

PROGRAM 10

   REG1 = 40
0: LABEL 0
   DEC REG1
   IF REG1 > 0 GOTO 0
```

Important Details

When writing programs for the 9010A Language Compiler on your computer, you will find that it is necessary to pay attention to some details that you could ignore when entering programs using the 9010A keyboard. These important rules are:

- Each 9010A statement must be on a separate line. Continuation lines are not allowed.
- A statement may begin in any column.
- Spaces and tabs are ignored, except when they occur in DPY or AUX statements.
- Blank lines are ignored.
- Adjacent keywords, symbolic names (described in Part 3 of this section), and numbers must be separated by at least one space.

  EXAMPLES:

VALID	INVALID
READ PROBE	READPROBE
DTOG @ 100F9 = 80 BIT 7	DTOG @ 100F9 = 80 BIT7

- Uppercase and lowercase characters can be used interchangeably.

  EXAMPLE:

  The following program statements are all equivalent:

```
WRITE @ 100FA = 1

write @ 100fa = 1

Write @ 100FA = 1
```

- In a few cases, the 9010A Language does not correspond exactly to the keys that would be pressed if the program were being entered on the 9010A keyboard.

  For example, INC REG5 is a legal statement accepted by the compiler. However, the keystrokes used to create this statement on the 9010A are INC 5, which would not be accepted by the compiler.

  As another example, REGA = REGA INC is a legal statement accepted by the compiler, but the keystrokes used to create this statement on the 9010A are REG A INC, which would not be accepted.

- In general, the keywords of the 9010A Language are not identical to the wording that appears on the 9010A keyboard.

  For example:

KEYBOARD	9010A LANGUAGE
DISPL	DPY
COMPL	CPL
RPEAT	REPT
TOGGL DATA	DTOG

  In all cases, however, the keywords accepted by the compiler are compatible with listings produced by the 9010A through the RS-232-C auxiliary interface.

Program Comments

The 9010A Language Compiler allows you to add comments to your programs, making the programs more readable and easier to maintain.

The rules for using comments are:

- Comments start with an exclamation point (!), and they extend to the end of the line.
- A comment can be on the same line as a 9010A statement, or it can be on a separate line.
- If a comment extends over several lines, each line must begin with an exclamation point.
- A comment cannot be placed in the middle of a 9010A statement.

EXAMPLE:

```
! This example demonstrates the use of comments.

PROGRAM 0                          ! Main program

   DPY-THIS IS AN EXAMPLE          ! DPY statements can have comments
   EXECUTE PROGRAM 10              ! Execute the delay routine
   DPY-OF A VALID 9010A PROGRAM

PROGRAM 10                         ! Delay routine

   REG1 = 40                       ! Initialize REG1 with delay count
0: LABEL 0
   DEC REG1                        ! Count down to zero
   IF REG1 > 0 GOTO 0
```

9010A Programs

The 9010A Language allows programs to be specified in the same form that would be produced by the 9010A AUX I/F PROGM keys. By connecting a printer to the auxiliary interface of the 9010A, you can obtain formatted listings of your 9010A programs. These listings can serve as examples of acceptable syntax.

Address Space Information

The 9010A Language allows address space information to be specified in the same form that would be produced by the 9010A AUX I/F LEARN keys.

The following rules apply to address space information:

- The address space information must appear at the beginning of the source file, preceding all 9010A programs (i.e., before the first PROGRAM statement).

  Up to 100 address descriptors may be specified.

  EXAMPLE:

```
! This is an example of a source file containing
! UUT memory map information

ADDRESS SPACE INFORMATION

   RAM @ C000-FFFF

   ROM @ 0000-1FFF SIG 0295
   ROM @ 2000-3FFF SIG C262

PROGRAM 0

   RAM SHORT
   ROM TEST
```

Setup Information

The 9010A Language allows any or all of the setup parameters to be specified in the same form produced by the 9010A AUX I/F SETUP keys.

The 9010A setup functions allow the operator to control the reporting of UUT errors, enable microprocessor lines, and specify operating parameters. The 9010A Operator Manual contains complete information on the various setup parameters that can be specified.

The following rules apply to setup information:

- Setup information must appear at the beginning of the source file, preceding all 9010A programs (i.e., before the first PROGRAM statement). The setup information may appear either before or after any address space information.

- You can specify all of the setup parameters, some of them, or none of them. Setup parameters that are not explicitly set assume default values contained in the pod data file (if a pod data file is specified), or to the power-up values supplied by the 9010A.

- Some setup information is pod-dependent. The pod-unique information includes enableable forcing lines, the default bus test address, and the RUN UUT address. If any of your 9010A programs depend upon the pod-unique features (i.e., a forcing line needs to be disabled or a RUN UUT must be performed at the pod's default address, then the appropriate Pod Data file needs to be included in the source file. To do this, an INCLUDE statement is used to specify the correct Pod Data file:

  INCLUDE "podname.POD"

  This statement must appear before the setup information in the source file.

  EXAMPLE:

  ```
  INCLUDE "8086.POD"
  ```

The INCLUDE statement is described in Part 2 of this section. Pod data files are described below.

- A POD statement should be placed in the setup section if any of the programs depend upon pod-unique features.

  EXAMPLE:

```
INCLUDE "8086.POD"

SETUP INFORMATION

POD - 8086
TRAP ACTIVE FORCE LINE/NO
TRAP ACTIVE INTERRUPT-YES
```

Pod Data Files

The 9010A Language Compiler program diskette contains a collection of files with names like 8086.POD, 68000.POD, etc. These files contain pod-specific definitions for enableable forcing lines, bus test address (BUSADR), and RUN UUT address (UUTADR). If you want to specify any of the pod-specific setup parameters, you should merge the appropriate pod data file into your source file by using an INCLUDE statement. The INCLUDE statement must appear before the SETUP INFORMATION section.

Pod-specific forcing lines are defined in the pod data file. The pods equate each of the forcing lines to a bit in an enable mask.

For example, the definitions for an 8086 pod are shown below:

```
!
! Each of the enableable forcing lines must be defined as the
! appropriate bit in the enable mask.
!
FORCELN READY = 0          ! READY is bit 0 in the enable mask
FORCELN HOLD = 1           ! HOLD is bit 1 in the enable mask
FORCELN INTR = 3           ! INTR is bit 3 in the enable mask
BUSADR = 0000              ! BUSADR is the pod's default BUS TEST
                           !    address
UUTADR = FFFF0             ! UUTADR is the pod's default RUN UUT
                           !    address
                           ! Other definitions can follow
```

9010A/Pod Interaction

Setup information takes effect immediately upon loading a new hex file into the 9010A (whether through READ TAPE or AUX I/F READ). An interaction takes place between the 9010A and the interface pod when the hex file is loaded and setup information may be changed to the default setting of the pod if:

1. The pod name was not specified with a POD statement in the setup section, or

2. A different pod is connected to the 9010A while the hex file is being loaded into the 9010A.

To avoid changing the parameters in pod-dependent programs:

1. An INCLUDE statement must be used in the setup section of the source program to include information from the appropriate Pod Data file.

2. A POD statement must be used in the setup section of the source program to identify which pod is being used.

3. The correct pod (or no pod) must be connected to the 9010A when downloading a compiled hex file.

Sample Program

The following sample source file illustrates the concepts introduced in Part 1. This example can be used as a basis for writing your own 9010A programs. Before continuing to Part 2, you may wish to copy this program using your host computer and transfer it to your 9010A as an exercise.

Section 5, Using the Compiler, shows how to run the compiler and transfer the generated hex files to the 9010A.

Once you feel comfortable using the compiler at this level, you should proceed with Part 2, which introduces some extended features that simplify the task of writing larger test programs.

```
! This program tests the U52 flip-flop on the output side
! of the 8255 PIA on the NEC TK-80A single-board computer.

INCLUDE "8080.POD"

SETUP INFORMATION
   POD - 8080
   TRAP ACTIVE FORCE LINE-NO
   TRAP ACTIVE INTERRUPT-YES

ADDRESS SPACE INFORMATION        ! Note: this address space information
   RAM @ 8C00-8FFF               ! is not actually used by the
   ROM @ 0000-07FF SIG F77C      ! program, but the descriptors
   I/O @ 100F8-100FA BITS FF     ! will be loaded into the 9010A

PROGRAM 0                        ! Main program

   WRITE @ 100FB = 80            ! Configure PIA for output
0: LABEL 0                       ! Set up entry
   REG2 = A                      ! Set up 10 stimulus loops
   REG8 = 1A09                   ! Store U52 - pin9 for prompt

   EXECUTE PROGRAM 3             ! Prompt for probe placement
   EXECUTE PROGRAM 1             ! Detect probe placement
   IF REG8 = 1 GOTO 1            ! Branch on open node
   DPY-TESTING U52#              ! Display - Testing U52
   READ PROBE                    ! Clear probe data register
```

```
2: LABEL 2                          ! Stimulus loop
   WRITE @ 100FA = 1                ! Set flip flop D input high
   DTOG @ 100F9 = 80 BIT 7          ! Toggle flip flop
   WRITE @ 100FA = 0                ! Set flip flop D input low
   DTOG @ 100F9 = 80 BIT 7          ! Toggle again
   DEC REG2                         !
   IF REG2 > 0 GOTO 2               ! Loop for 10 tries
   EXECUTE PROGRAM 2                ! Extract probe data
   IF REG8 = A GOTO 3               ! Branch on probe count = 10
   DPY-U52 TOGGLING IMPROPERLY#     ! Display bad toggle
   GOTO 4                           ! Exit

1: LABEL 1                          ! Open node loop
   DPY-WAS PROBE IN PLACE# ?1       ! Query
   IF REG1 = 0 GOTO 0               ! Branch if probe not ready
   DPY-U52 OPEN#                    ! Display - U52 bad
   GOTO 4                           ! Exit

3: LABEL 3                          ! Device passed
   DPY-U52 TEST PASSED#
4: LABEL 4                          ! End

PROGRAM 1                           ! Program to detect probe placement

   SYNC FREE-RUN
0: LABEL 0                          ! Set counts
   REG1 = 50                        ! Open count = 50
   REG2 = 20                        ! Debounce count =20

1: LABEL 1                          ! Open loop
   READ PROBE                       ! Gather level information
   IF REG0 AND 5000000 > 0 GOTO 2   ! Branch on bounce level
   DEC REG1                         ! Decrement open count
   IF REG1 > 0 GOTO 1               ! Loop if count > 0
   REG8 = 1                         ! Set Open Node flag
   GOTO 3                           ! Exit

2: LABEL 2                          ! Debounce loop
   DEC REG2                         ! Decrement debounce count
   READ PROBE                       ! Gather level information again
   IF REG0 AND 5000000 = 0 GOTO 0   ! Branch on open level
   IF REG2 > 0 GOTO 2               ! Loop if count > 0
   REG8 = 0                         ! Set Begin Test flag
3: LABEL 3                          ! End
```

```
PROGRAM 2                          ! Program to extract the probe data

   READ PROBE                      ! Gather probe information
   REG8 = REG0 AND 7F              ! Extract count
   REG9 = REG0 SHR SHR SHR SHR SHR SHR SHR SHR AND FFFF ! Extract Sig.
   REGA = REG0 SHR SHR SHR SHR SHR SHR SHR SHR
   REGA = REGA SHR SHR SHR SHR SHR SHR SHR SHR
   REGA = REGA SHR SHR SHR SHR SHR SHR SHR SHR AND 7 ! Extract level

PROGRAM 3                          ! Program to prompt the operator

   REG2 = REG8 AND 7F              ! Register 2 = pin number
   REG1 = REG8 SHR SHR SHR SHR SHR SHR SHR AND 7F ! R1 = Device num.
   DPY-PROBE U@1 PIN @2#           ! Prompt for probe placement
```

PART 2: CODING SHORTCUTS

Introduction

The 9010A Language is designed to be compatible with the formatted listings produced by the AUX I/F keys on the 9010A. In this format, you may find that some statements require much more typing than would be required to enter the same statement through the 9010A keyboard.

To make it easier to enter large programs on the host system, the 9010A Language Compiler provides several features which reduce the amount of typing required. These features are:

- Optional Keywords and Keyword Abbreviations
- Unary Operator Shorthand
- Default Entries
- File Inclusion

Optional Keywords and Keyword Abbreviations

The 9010A Language provides the option of abbreviating certain keywords or leaving them out entirely. Appendix C, Optional Keywords and Keyword Abbreviations, contains a complete list of the optional keywords and valid abbreviations. Furthermore, the syntax diagrams in Section 6, Language Reference, indicate the abbreviated form of each statement in the language.

EXAMPLES:

STATEMENT	ABBREVIATED FORM
EXECUTE PROGRAM 5	EXECUTE 5
	or
	EX 5
WRITE @ 100FF = 25	WRITE 100FF = 25
	or
	WR 100FF = 25
3: LABEL 3	3:
SYNC ADDRESS	SYNC A
	or
	3: SYNC A

Unary Operator Shorthand

For multiple applications of a unary operator (INC, DEC, CPL, SHL, or SHR), you may specify the unary operator followed by a decimal number indicating how many times it is to be applied.

EXAMPLE:

statement

```
REG1 = REG0 SHR SHR SHR SHR SHR SHR SHR AND 7F
```

be abbreviated to

```
REG1 = REG0 SHR 7 AND 7F
```

Default Entries

When programs are created through the 9010A keyboard, many of the entries in a program step can be defaulted to the appropriate dedicated register by pressing the ENTER key.

For example, to create the statement READ @ REGF, you need only press the READ and ENTER keys on the 9010A. The read address automatically defaults to REGF.

Similarly, if the WRITE and ENTER keys are pressed on the 9010A, the write address automatically defaults to REGF, and the data to be written defaults to REGE.

The 9010A Language provides a similar default capability. You can use an asterisk (*) to indicate that an entry should default to a dedicated register. The syntax diagrams in Section 6, Language Reference, indicate which entries can be defaulted in this way.

EXAMPLES:

STATEMENT	DEFAULT FORM
READ REGF	READ *
WRITE REGF = REGE	WRITE * = *

File Inclusion

To facilitate handling large collections of source code which can be shared by several programs, the 9010A compiler provides a file inclusion feature. This feature allows you to create a library of useful 9010A programs and use the file inclusion facility to merge them into a particular source file.

A line of the form

INCLUDE "filename"

in the source file will be replaced by the contents of the file "filename" when the program is compiled. The effect is equivalent to manually entering the contents of the included file at that point in the source file.

EXAMPLE:

Assuming that the file PROMPT.S contains

```
PROGRAM 3

    REG2 = REG8 AND 7F
    REG1 = REG8 SHR 7 AND 7F
    DPY-PROBE U@1 PIN @2#
```

then the source file

```
PROGRAM 0

    REG8 = 1A09
    EXECUTE PROGRAM 3

INCLUDE "PROMPT.S"
```

have exactly the same effect as the source file

```
PROGRAM 0

    REG8 = 1A09
    EXECUTE PROGRAM 3

PROGRAM 3

    REG2 = REG8 AND 7F
    REG1 = REG8 SHR 7 AND 7F
    DPY-PROBE U@1 PIN @2#
```

Sample Program

The following example is similar to the one given at the end of Part 1, but it takes full advantage of the abbreviation features. The example assumes that the file PROBE1.S contains the code for PROGRAM 1, PROBE2.S contains PROGRAM 2, and PROMPT.S contains PROGRAM 3.

```
! This program tests the U52 flip-flop on the output side
! of the 8255 PIA on the NEC TK-80A single-board computer.

INCLUDE "8080.POD"

SETUP                                  ! Equivalent to SETUP INFORMATION
   POD - 8080
   TRAP ACTIVE FORCE LINE NO           ! - in SETUP statements is optional
   TRAP ACTIVE INTERRUPT YES

ADDRESS SPACE
   RAM 8C00-8FFF                       ! @ omitted
   ROM 0000-07FF SIG F77C
   I/O 100F8-100FA BITS FF

PROGRAM 0

   WR 100FB = 80                       ! WRITE abbreviated to WR
0: REG2 = A
   REG8 = 1A09
   0EX 3                               ! Same as EXECUTE PROGRAM 3
   EX 1
   IF REG8 = 1 GOTO 1
   DPY TESTING U52#
   PROBE
2: WR 100FA = 1                        ! Short form of LABEL statement
   DTOG 100F9 = 80 BIT 7
   WR 100FA = 0
   DTOG 100F9 = 80 BIT 7
   DEC REG2
   IF REG2 > 0 GOTO 2
   EX 2
   IF REG8 = A GOTO 3
   DPY U52 TOGGLING IMPROPERLY#        ! - is optional in DPY statement
   GOTO 4
```

```
1: DPY WAS PROBE IN PLACE# ?1
   IF REG1 = 0 GOTO 0
   DPY U52 OPEN#
   GOTO 4
3: DPY U52 TEST PASSED#
4:

INCLUDE "PROBE1.S"          ! Code for PROGRAM 1 is inserted here

INCLUDE "PROBE2.S"          ! PROGRAM 2

INCLUDE "PROMPT.S"          ! PROGRAM 3
```

PART 3: SYMBOLIC NAMES

Introduction

The 9010A Language allows programs, labels, and registers to be referred to by symbolic names. For example, the statement

```
EXECUTE PROGRAM 5
```

could be replaced by something more meaningful, such as

```
EXECUTE PROGRAM DELAY
```

Symbolic names can contribute greatly to the readability of programs, allowing the programs to be self-documenting to a large degree.

The following rules apply to symbolic names:

- Symbolic names must begin with a letter, and they can contain any number of letters, digits, and underscore characters (_).
- Only the first eight characters of a name are significant. For example, TESTMENU1 AND TESTMENU2 are treated as identical names.
- 9010A Language keywords, such as LOOP, READ and PROGRAM, cannot be used as symbolic names. For example, although LOOP cannot be used as a symbolic label name, LOOP1 is acceptable.
- Appendix A contains a complete list of the 9010A Language keywords. Using a keyword as a symbolic name causes the compiler to issue a SYNTAX ERROR message.
- Symbolic names must contain at least one letter other than A, B, C, D, E, or F so that they can be distinguished from hexadecimal constants. This means that words like BAD, ACE, or FADE cannot be used as symbolic names because the compiler will interpret them as hex constants. Using a hex constant as a symbolic name causes the compiler to issue a SYNTAX ERROR message.

- Symbolic names can be used anywhere that the corresponding actual program number, register number, or label number can occur in a 9010A program.

  Forward references are permissable for program names and label names. In other words, an EXECUTE or GOTO statement using a symbolic name is allowed to appear either before or after corresponding PROGRAM or LABEL statements.

  Register names may appear in DPY and AUX statements.

- Symbolic names are case-insensitive. For example, a name can be declared in uppercase and referenced in lowercase, and names can be a mixture of uppercase and lowercase letters.

Symbolic Program Names

9010A Language allows programs to be referred to by name as well as by number. By choosing descriptive program names, you can make your programs much more readable and maintainable.

Symbolic program names do not need to be declared explicitly. Simply using a name in a PROGRAM statement or in an EXECUTE statement is sufficient to define that symbolic program name.

The compiler assigns sequential program numbers to symbolically-named programs, starting with PROGRAM 0 for the first program in the source file. Each time a symbolic PROGRAM statement is encountered, the next sequential program number is assigned to it. A source file can contain any combination of programs with actual program numbers and programs with symbolic names.

NOTE

EXECUTE statements can appear either before or after the PROGRAM statement. They do not have any effect on the sequence of program numbers assigned to symbolic program names.

Whenever the compiler encounters a program in the source file with an actual program number rather than a symbolic name, then subsequent symbolic program names are assigned program numbers that follow sequentially from the given program number.

EXAMPLE:

```
PROGRAM 5       ! Compiled as PROGRAM 5
  :
  :
PROGRAM PA      ! Compiled as PROGRAM 6
  :
  :
PROGRAM PB      ! Compiled as PROGRAM 7
  :
  :
PROGRAM 20      ! Compiled as PROGRAM 20
  :
  :
PROGRAM PC      ! Compiled as PROGRAM 21
  :
  :
```

The following rules apply whenever a source file contains programs with actual program numbers:

- Programs with actual program numbers must be in numeric order in the source file. For example, PROGRAM 5 must precede PROGRAM 20.

- There must be a large enough gap between two programs with actual program numbers for any intervening programs with symbolic names. For example, if the source file contains a PROGRAM 8 and a PROGRAM 11, PROGRAM 8 must precede PROGRAM 11 and there can be at most two symbolically-named programs between them.

  EXAMPLE:

```
! This example demonstrates the use of symbolic program names
!
! The compiler will assign PROGRAM 0 to MAIN and
! PROGRAM 1 to DELAY

PROGRAM MAIN

   DPY-THIS IS AN EXAMPLE
   EXECUTE DELAY
   DPY-OF A VALID 9010A PROGRAM

PROGRAM DELAY

   REG1 = 40
0: DEC REG1
   IF REG1 > 0 GOTO 0
```

Symbolic Labels

The symbolic label feature allows you to refer to a branching location with a mnemonic name, providing the same advantages as symbolic program names.

The following rules apply to symbolic labels:

- Symbolic label names are not declared explicitly. Simply using a name as the target of a GOTO or in a LABEL statement is sufficient to define a symbolic label name.
- Within a single program, symbolic label names cannot be mixed with hexadecimal label numbers. A source file may contain a mixture of hexadecimal and symbolic labels, but within a given program all labels must be either hexadecimal or symbolic.
- Symbolic labels are local to the program in which they appear. This means that it is possible to have duplicate label names in different programs without conflict.
- Each program is limited to 16 label definitions, even if the labels are referred to symbolically.
- Within a given program, the compiler assigns hexadecimal labels to symbolic label names sequentially, starting at 0. The assignment is made upon the first appearance of the label, whether it is a LABEL statement or the target of a GOTO statement.

EXAMPLE:

```
PROGRAM FIND
:
:
SEARCH:                              ! Label 0 will be assigned to SEARCH
   READ @ REG1
   INC REG1
   IF REGE = REG3 GOTO FOUND         ! Label 1 will be assigned to FOUND
   IF REG1 > REG2 GOTO NOTFOUND      ! Label 2 will be assigned to
                                     NOTFOUND
                                     GOTO SEARCH
NOTFOUND:
:
:
FOUND:
:
:
```

Symbolic Register Names

Another way to enhance program readability is to use symbolic names for registers. The usage of the various 9010A registers can be made clear by choosing appropriate symbolic names.

Symbolic register names are a bit more complex than program or label names. For example, register names must be explicitly declared in a DECLARATIONS section. Another difference is that register names can be either local to a single program or global to the entire source file, depending on how they are declared.

Symbolic register names must be declared in an ASSIGN statement of the form

ASSIGN REGn TO name

ASSIGN statements are collected together into a DECLARATIONS section.

EXAMPLE:

```
DECLARATIONS
  ASSIGN REG1 TO ERRCNT
  ASSIGN REG2 TO FREQ
```

If the register names are to be used only within a particular program, then the DECLARATIONS section should appear between the PROGRAM statement and the body of the program itself. If the declarations are for global registers that are shared among several programs, then the DECLARATIONS section must appear at the beginning of the source file before the first PROGRAM statement.

It is possible to assign several symbolic names to the same register within a program. This can be done by specifying a list of names in a single ASSIGN statement or by using multiple ASSIGN statements. It should be noted, however, that using multiple names for the same register (implying multiple uses for a register) can lead to programming errors. It is the programmer's responsibility to ensure the integrity of the register contents.

EXAMPLE:

```
PROGRAM UUTTEST

DECLARATIONS
  ASSIGN REG1 TO ERRCNT          ! REG1 will be used when ERRCNT is
                                 ! referred to
  ASSIGN REG2 TO PINCNT,SETBIT   ! Both PINCNT and SETBIT will be
                                 ! allocated to REG2
                                 ! TEMP will also be allocated to REG2
  ASSIGN REG6 TO MASK            ! MASK will be allocated to REG6

  ERRCNT = 0                     ! Actually sets REG1 = 0
  SETBIT = 4                     ! Since PINCNT, TEMP, and SETBIT all
                                 ! share the same register, this
                                 ! statement has the effect of also
                                 ! setting PINCNT and TEMP

  MASK = SETBIT CPL AND FF
  WRITE @ REG3 = MASK
```

Predefined Register Names

Symbolic names have been predefined for each of the dedicated registers. These names can be used anywhere in a program that a register reference can be made. It is not necessary to declare these symbolic register names.

The predefined register names and their functions are as follows:

DEDICATED REGISTER	SYMBOLIC NAME	FUNCTION
A	BITMASK	Bit Mask
B	ROMSIG	ROM Signature
C	STSCTL	STS/CTL Information
D	BITNUM	Bit Number
E	DAT	Data
F	ADR	Address
0	PBDAT	Read Probe Data

Sample Program

The following example is similar to the one given at the end of Part 2. The example assumes that PROGRAM 1 contained in the file PROBE1.S has been renamed to PRBPLACE, PROGRAM 2 (in PROBE2.S) has been renamed to PROGRAM UNPACK, and PROGRAM 3 (in PROMPT.S) has been renamed to PROGRAM PROMPT.

```
! This program tests the U52 flip-flop on the output side
! of the 8255 PIA on the NEC TK-80A single-board computer.
!
! This version of the program demonstrates
! the use of symbolic names.

INCLUDE "8080.POD"

SETUP
   POD - 8080
   TRAP ACTIVE FORCE LINE NO
   TRAP ACTIVE INTERRUPT YES

ADDRESS SPACE
   RAM 8C00-8FFF
   ROM 0000-07FF SIG F77C
   I/O 100F8-100FA BITS FF

DECLARATIONS                        ! Global register declarations
   ASSIGN REG8 TO LOAD              ! Used in display message
   ASSIGN REG8 TO FLAG              ! Flag an output from probe placement
   ASSIGN REG8 TO COUNT             ! Count an output from unpacker

PROGRAM U52TEST

DECLARATIONS                        ! Local declarations
   ASSIGN REG2 TO CNT

   WR @ 100FB = 80                  ! Beginning of program body
```

```
START:
   CNT = A
   LOAD = 1A09
   EX PROMPT                    ! Symbolic program reference
   EX PRBPLACE
   IF FLAG = 1 GOTO OPEN
   DPY TESTING U52#
   PROBE

STIMULUS:                       ! Symbolic label definition
   WR @ 100FA = 1
   DTOG @ 100F9 = 80 BIT 7
   WR @ 100FA = 0
   DTOG @ 100F9 = 80 BIT 7
   DEC CNT
   IF CNT > 0 GOTO STIMULUS     ! Symbolic label reference

   EX UNPACK
   IF COUNT = A GOTO DONE
   DPY U52 TOGGLING IMPROPERLY#
   GOTO EXIT

OPEN:
   DPY WAS PROBE IN PLACE# ?1
   IF REG1 = 0 GOTO START
   DPY U52 OPEN#
   GOTO EXIT

DONE:
   DPY U52 TEST PASSED#

EXIT:                           ! End of main program

INCLUDE "PROBE1.S"              ! PROGRAM 1 must be renamed to
                                PROGRAM
                                ! PRBPLACE in file PROBE1.S
INCLUDE "PROBE2.S"              ! PROGRAM 2 renamed to PROGRAM
                                UNPACK
                                ! in file PROBE2.S
INCLUDE "PROMPT.S"              ! PROGRAM 3 renamed to PROGRAM
                                PROMPT
                                ! in file PROMPT.S
```

Section 5
Using the Compiler

CONTENTS

INTRODUCTION

This section provides the information needed to use the 9010A Language Compiler (9LC) and the File Transfer Utility program (XFER). The following topics are covered in this section:

- Preparing the Source File
- Compiling
- Transferring Programs

PREPARING THE SOURCE FILE

The first step in using the compiler is to create a source file containing the desired 9010A programs. The source file may use all the language features introduced in Section 4, Writing Programs. For detailed information on specific statements, see Section 6, Language Reference.

To edit and modify the source files, you should use the text editor that you normally use on your host computer system.

By convention, the names of source files are usually given a filename extension of .S, but this is not required by the compiler. PIA.S is an example of a typical source file name.

If the source file is not contained on a system default device the filename may also require a device name. For example:

```
MF1:DEMO.S
```

might specify a source file named DEMO.S on an optional floppy disk (MF1). Consult your host computer user's manuals for information about complete filename specifications.

The program diskette contains a sample source file named DEMO.S. This file is used as an example in the following procedures for using the compiler.

If you already have 9010A programs stored on 9010A cassette tapes, it is possible to transfer them to your host system and use them with the compiler. The procedure for doing this is described later in this section under the heading Transferring Programs from the 9010A.

COMPILING

Once you have created a source file, you are ready to run the compiler. The compiler reads the source file and creates an equivalent hex file which can then be transferred to the 9010A through the RS-232-C serial interface.

You have the option of running the compiler in either of two modes: the interactive mode or the command line mode.

NOTE

The following examples require the file DEMO.S to be on a non write-protected disk. If your working copy of the system disk (as described in Section 3) is write-protected, you will need to use two disk drives, with a copy of the demo program DEMO.S on a non write-protected disk in the second drive.

Interactive Mode

If you run the compiler in the interactive mode, it prompts you for the names of the source and hex files. The compiler asks you whether you want a listing file produced. If you answer yes, the compiler asks for the name of the listing file and the specific listing file options desired.

To run the compiler in the interactive mode, simply enter the command

[device]9LC ⟨RETURN⟩

NOTE

The use of [device] in the examples in this section refers to an optional device name specification that may be required for files that are not on a system default device.

⟨RETURN⟩ indicates the key that is pressed to terminate the command line.

After you have entered the filename command 9LC, the compiler responds by displaying its version number and copyright notice. The compiler then asks for the name of the source file. You now enter the name of the source file, for example:

[device]DEMO.S ⟨RETURN⟩

Next, you are prompted for the name of the hex file to be created by the compiler. Enter the name of the hex file followed by RETURN. If you simply press RETURN, the compiler generates a hex file with the same name as the source file, but with a .H extension appended to the root of the source file name. In this example, the hex file name becomes DEMO.H on the same device as DEMO.S.

⟨RETURN⟩

The compiler then asks you whether you want a listing of the source program. You should respond by entering Y (yes) or N (no). For this example, enter

Y ⟨RETURN⟩

If you request a listing file, the compiler prompts you for the listing file name. You should enter the required name, or simply press RETURN to get the same name as the source file with a .L extension, in this case DEMO.L (also on the same device as DEMO.S).

⟨RETURN⟩

After you have specified the listing file name, the compiler displays the listing file options. These options are described later in this section. If you simply press RETURN, the compiler produces a copy of the source file with line numbers added.

⟨RETURN⟩

At this point, you have specified the compiler options. The compiler displays the equivalent command line (the significance of which is explained below) and then proceeds to compile the source file.

While it is processing the source file, the compiler displays the name of each program, its program number, and the number of bytes of 9010A program memory required. After the compiler has processed the source files, it displays the total number of bytes required and then returns to the host operating system.

If the compiler detects any errors in the source file, it displays an appropriate error message along with the source line containing the error. The error message also appears in the listing file if a listing file has been requested. If the source file contains any errors, then a hex file will not be created.

Command Line Mode

An alternative way of running the compiler is to specify all the desired options directly on the command line. If any options are specified on the command line, then the prompting described above is completely bypassed.

To run the compiler in the command line mode, you enter a command in the following format:

[device] 9LC [-listoptions] [-H hexfile] [-L [listfile]] srcfile ⟨RETURN⟩

In the above notation, items within brackets [] are optional.

Srcfile is the name of the source file to be processed by the compiler. It may require an optional device name specification.

The -H option is used to override the default hex file name (.H extension). Hexfile is the desired name of the hex file produced by the compiler.

The -L option is used to override the default listing file name (.L extension). Listfile is the desired name of the listing file produced by the compiler.

The -L option without a listing file name can be used to produce a listing file in the case where no listing options are specified. The listing file is generated with the .L extension.

The -listoptions allow you to specify the form of the listing file. The listing file options and their functions are:

I Expand Include Files

S Replace Symbolic Names

D Replace Default Entries

A Expand Keyword Abbreviations

Specifying any of the options I,S,D, or A causes a listing file to be produced. The paragraphs following the next heading, Listing File Options, contain more information regarding these options.

The following examples illustrate the use of the command line mode.

The command

```
9LC  -L [device]DEMO.S ⟨RETURN⟩
```

produces exactly the same results as the sequence of options described above under the heading Interactive Mode.

To compile the source file DEMO.S and produce a hex file named DEMO.H but not produce a listing file, use the following command:

```
9LC  [device]DEMO.S ⟨RETURN⟩
```

To produce a listing file with include files expanded, use the following command:

```
9LC  -I [device]DEMO.S ⟨RETURN⟩
```

Listing File Options

The compiler provides a number of different listing file options. These options are described below.

- I Expand Include Files

  If the source file contains an INCLUDE statement, such as

  INCLUDE "6802.POD"

  the listing file normally just copies this statement. However, if the -I option is specified, then the listing file also shows the contents of the file 6802.POD.

- S Replace Symbolic Names

  If the source file contains symbolic names for registers, programs, or labels, they are normally copied to the listing file as they appear in the source file. However, if the -S option is specified, then the symbolic names are replaced by the actual program number, register number, or label number.

  EXAMPLE:

lines from source file:	EXECUTE DELAY INC ERRCNT
normal listing file:	EXECUTE DELAY INC ERRCNT
listing file with -S option:	EXECUTE 7 INC REG2

- D Replace Default Entries

  If the source file contains any default entries (indicated by *), the listing file normally copies the statement as it appears in the source file with the * in place. However, if the -D option is specified, then the listing file substitutes the appropriate default register for the *.

  EXAMPLE:

line from source file:	WRITE @ * = *
normal listing file:	WRITE @ * = *
listing file with -D option:	WRITE @ REGF = REGE

- A Expand Keyword Abbreviations

  If the source file contains the abbreviations RD, WR, or EX, they are normally copied to the listing file in their abbreviated form, just as they appear in the source file. However, if the -A option is specified, then the listing file replaces all occurrences of these abbreviated keywords with the full keyword.

  EXAMPLE:

line from source file:	EX PROGRAM 5
normal listing file:	EX PROGRAM 5
listing file with -A option:	EXECUTE PROGRAM 5

Syntax Errors

All programmers eventually have an elusive syntax error to track down. The compiler provides some help by pinpointing the location of the syntax error in the listing file, especially if the listing option has been selected to expand any include files.

Even after you have found the location of the syntax error, the exact cause of the problem may not be obvious. Appendix F, Error Messages, contains a list of common syntax errors that can be used as a time-saving checklist.

TRANSFERRING PROGRAMS

Once you have successfully compiled your programs, you are ready to transfer the generated hex file to the 9010A through the RS-232-C serial interface. XFER, the File Transfer Utility program, is provided on the program diskette for this purpose.

Before running XFER, the 9010A must be connected to the serial port of the host system as described in Section 3, Getting Started.

To run XFER, simply enter the command

[device]XFER ⟨RETURN⟩

After you have entered the command XFER, the program responds by displaying its version number and copyright notice, followed by a main menu of file transfer options:

T Transfer hex file from host to 9010A
S Transfer source files from 9010A to host
H Transfer hex files from 9010A to host
C Configure host system
Q Quit

Whenever this menu is displayed, you can return to the host operating system by entering

Q ⟨RETURN⟩

You can also use the Q command to return to this main menu when prompted for a filename in any of the other options in this menu.

You should select the C option if you want to change the default setting for the RS-232-C serial port. Refer to Section 3, Getting Started, for further information on configuring the serial port.

Transferring Programs to the 9010A

To transfer a file from the host system to the 9010A, you should select the T option. Since this is the default option, you may simply press the RETURN key.

⟨RETURN⟩

You are then asked to enter the name of the hex file to be transferred. To transfer DEMO.H, the file produced by compiling DEMO.S in the previous examples, enter

[device]DEMO.H ‹RETURN›

XFER then instructs you to prepare the 9010A for reading by pressing the AUX I/F, READ, and YES keys on the 9010A. As soon as you have pressed the YES key, the host system starts transferring the hex file to the 9010A.

NOTE

Pressing the AUX I/F and READ keys causes the 9010A to clear its program memory and reset all the setup parameters to their default values. Any programs currently in the 9010A memory are lost.

When the file transfer is complete, the 9010A displays the message AUX-RECEIVING - COMPLETE, and the host system again displays the file transfer options menu. To exit from XFER and return to the host operating system, enter

Q ‹RETURN›

The test programs can now be executed on the 9010A just like any other 9010A programs. Once the transfer is complete, the 9010A may be disconnected from the host system.

If you have followed the example above to compile DEMO.S and transfer DEMO.H to the 9010A, you can execute the program on the 9010A by pressing the following keys:

EXECUTE 0 ENTER

If your source file contains symbolic program names, you must determine which actual program numbers were assigned by the compiler to the symbolic program names. For this reason, the compiler displays the program names and their corresponding program numbers as it processes the source file.

Transferring Programs from the 9010A

Programs that are transferred from the 9010A to the host system can be stored either in source format or in hex format. If you have programs saved on 9010A cassettes and you want to modify them on the host system and take advantage of the features of the 9010A Language Compiler, then the programs must be stored in source format.

Hex format is useful if you simply want to store the 9010A programs on the host system and load them back into the 9010A at a later time without any modifications.

Source Format

To save programs from the 9010A on the host system in source format, select the S option from the file transfer options menu by entering

S ⟨RETURN⟩

NOTE

The following examples assume that you have transferred the programs in DEMO.H from the host system to the 9010A, as previously described under Transferring Programs to the 9010A.

XFER asks you for the name of the source file to be created on the host system. Respond by entering the source file name, in this case,

[device]DEMO1.S ⟨RETURN⟩

XFER then instructs you to prepare the 9010A for writing by pressing the AUX I/F and WRITE keys on the 9010A. When the transfer is complete, the 9010A displays the message

AUX-SENDING - COMPLETE

A menu of source options will now be displayed:

E - Save the entire file
S - Save the setup information
A - Save the address descriptors
P - Save all programs
0-99 - Save the specified program
R - Return to the main menu

If you choose to save the entire file, then setup information, address descriptors, and all programs will be saved.

If you choose to save the setup information or the entire file, XFER then prompts for the name of a pod data file, since the 9010A Language Compiler requires that a pod data file be included before any pod-dependent setup information. For the present example, enter

[device]Z80.POD ⟨RETURN⟩

In this case, XFER inserts a statement of the form INCLUDE "Z80.POD" immediately before the SETUP INFORMATION statement in the source file on the host system.

If, for some reason, you do not want to specify a pod data file, simply enter ⟨RETURN⟩ when prompted for the name of a pod data file. No INCLUDE statement will be inserted into the source file.

If you choose to save the address descriptors and none exist, a warning message will be displayed.

You have the option of saving individual programs or all of the 9010A programs in a single operation. If you attempt to save a program that does not exist, a warning message will be issued.

NOTE

The compiler requires setup and address space information to appear before any programs. Therefore, setup or address space information should be saved before any programs. If you attempt to save setup information or address descriptors after programs, the XFER program will print an error message.

At the end of the entire file transfer process, the new source file (in this case, DEMO1.S) exists on the host system. You can use the R option to return to the file transfer options menu.

The source file created by the file transfer utility can be modified using a text editor on the host system. For example, you may want to add comments or change the program numbers to symbolic names. The modified source file can be compiled, and the resulting hex file can be transferred back to the 9010A.

Hex Format

Hex format files are not generally modified on the host system, and they cannot be processed by the 9010A Language Compiler. The only reason for transferring files in hex format is to store the programs so that they can be loaded back into the 9010A at a later time.

To select the hex format, enter

H ⟨RETURN⟩

in response to the file transfer option menu.

XFER prompts you for the name of the hex file to be created. For example, you could enter

[device]DEMO1.H ⟨RETURN⟩

You are then instructed to press the AUX I/F and WRITE keys on the 9010A. When the transfer is complete, the 9010A displays the message AUX-SENDING - COMPLETE, and the file transfer utility returns to the file transfer option menu.

Section 6
Language Reference

CONTENTS

INTRODUCTION

This section provides a quick reference for 9010A Language syntax. As an aid to quick reference, the information contained here is concise. For an introduction to the language as a whole, see Section 4, Writing Programs.

This section is organized as follows:

- General Information
- Source File Syntax
- Setup Parameters
- 9010A Program Statements

Program statements are introduced with a syntax diagram that illustrates the legitimate construction. A complete definition of the various forms of the statement follow the syntax diagram. The statement definitions use the format shown in the following example page.

STATEMENT NAME

Syntax

Function

A description of the function(s) performed by the statement appears here.

- Characteristics, implications, and limitations of the statement are defined here.

Example

A programming example is shown here.

See Also

Any related statements or information are listed here.

SYNTAX DIAGRAM NOTATION

Syntax diagrams define correct spelling, punctuation, sequences of words, symbols, and expressions. The syntax diagrams used here conform to the following guidelines:

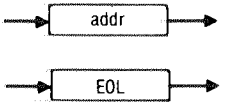
- Any path through a diagram starting from the left that does not run contrary to an arrowhead forms a legitimate statement.

- Words in a circular enclosure are to be entered as shown. Words can be typed in lowercase, uppercase, or a combination of lowercase and uppercase letters.

  Example:

- Words in a rectangular enclosure represent other information that is described either in the General Information section, in another syntax diagram, as a note on the same page, or that is in general use.

  Example:



- An asterisk in a circular enclosure above bracketed words indicates a default register entry. Only the asterisk should appear in the source file; the compiler substitutes the information in the brackets.

  Example:

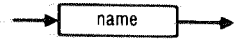
SPECIAL SYMBOLS

The following symbols are used in the syntax diagrams:

SYMBOL	FUNCTION
,	Separates a list of symbolic names (i.e., register name declarations)
EOL	Indicates end of line
–	Indicates range (i.e., addr to addr), used as a delimiter in AUX and DPY commands
>	Relational operator
>=	Relational operator
=	Relational operator
@	At
:	Separates the label name from the statement to be executed

SYMBOLIC NAMES

Symbolic names appear in the syntax diagrams as



The following rules apply to symbolic names:

- Symbolic names must begin with a letter, and they can contain any number of letters, digits, and underscore characters (_).
- Only the first eight characters of a name are significant. For example, TESTMENU1 and TESTMENU2 are treated as identical names.
- 9010A Language keywords, such as READ and PROGRAM, cannot be used as symbolic names. For example, LOOP cannot be used as a symbolic label name, although LOOP1 is acceptable.
- Appendix A contains a complete list of the 9010A Language keywords. Using a keyword as a symbolic name causes the compiler to issue a SYNTAX ERROR message.
- Symbolic names must contain at least one letter other than A, B, C, D, E, or F so that they can be distinguished from hexadecimal constants. This means that words like BAD, ACE, or FADE cannot be used as symbolic names because the compiler will interpret them as hex constants. Using a hex constant as a symbolic name causes the compiler to issue a SYNTAX ERROR message.

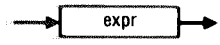
- Symbolic names can be used anywhere that the corresponding actual program number, register number, or label number can occur in a 9010A program.

- Forward references are permissable for program names and label names. In other words, an EXECUTE or GOTO statement using a symbolic name is allowed to appear either before or after the corresponding PROGRAM or LABEL statement.

- Symbolic names are case-insensitive. For example, a name can be declared in uppercase and referenced in lowercase, and names can be a mixture of uppercase and lowercase letters.

EXPRESSIONS

The syntax element



is used to designate a 9010A expression. Expressions consist of combinations of the following:

- Hexadecimal Constants (e.g., 10FC)
- Register References (e.g., REG3)
- Unary Operators (CPL, DEC, INC, SHL, SHR)
- Binary Operators (AND, OR)

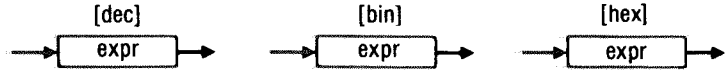
Unary operators specify operations that may be performed on only one register at a time. The five unary operators function as follows:

- CPL Replaces the value stored in the register with its binary ones complement.
- DEC Decrements the binary value of a register by 1.
- INC Increments the binary value of a register by 1.
- SHL Shifts the binary contents of the register one bit to the left. The farthest left bit is discarded. The farthest right bit becomes 0.
- SHR Shifts the binary contents of the register one bit to the right. The farthest right bit is discarded. The farthest left bit becomes 0.

Binary operators perform an operation with two registers or with a register and a hexadecimal value, or two hexadecimal values. The two binary operators function as follows:

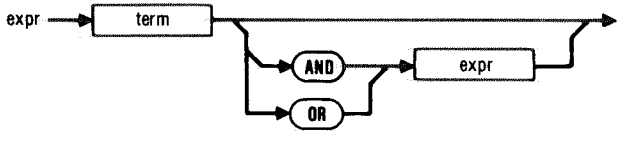
- AND Performs the logical bit-wise AND operation between two values.
- OR Performs the logical bit-wise OR operation between two values.

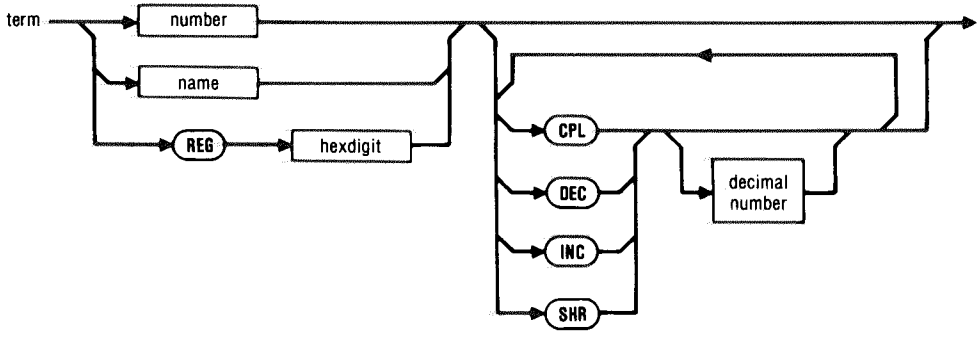
In certain contexts, expressions are interpreted as decimal, binary or hexadecimal numbers. These cases are indicated in the syntax diagram as follows:



Numeric constants in decimal expressions may contain only the digits 0 through 9. Similarly, numeric constants in binary expressions may contain only the digits 0 and 1, and hexadecimal expressions may contain only the digits 0-9, A-F.

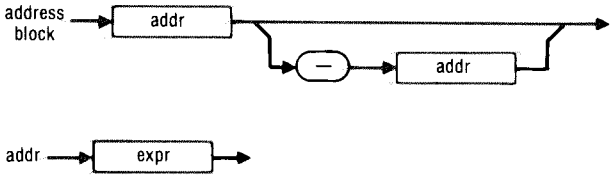
A unary operator followed by a decimal number is the unary operator shorthand feature described in Section 4, Part 2.





ADDRESSES

The following syntax diagrams apply to statements that require an address or an address range to be specified.



GENERAL INFORMATION

Statement Format

Follow these guidelines when constructing statements:

- Each 9010A statement must be on a separate line. Continuation lines are not allowed.
- A statement may begin in any column.
- Blanks and tabs are ignored, except when they occur in DPY or AUX statements.
- Blank lines are ignored.
- Adjacent keywords, symbolic names, and numbers must be separated by at least one blank.

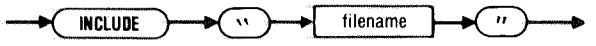
Program Comments

The rules for using comments are as follows:

- Comments start with an exclamation point (!), and they extend to the end of the line.
- A comment can be on the same line as a 9010A statement, or it can be on a separate line.
- If a comment extends over several lines, each line must begin with an exclamation point.
- A comment cannot be placed in the middle of a 9010A statement.

File Inclusion

The form of the INCLUDE statement is



The compiler replaces the INCLUDE statement with the contents of the specified file. The effect is equivalent to manually typing the contents of the included file in the source file at that point.

The following rules apply:

- The filename must be the name of an existing file.
- If the host operating system is case-sensitive regarding filenames, then the filename must be properly capitalized.
- A source file may include a file which in turn includes another file.
- INCLUDE statements must be on a line by themselves but can occur anywhere in the source file. INCLUDE statements may even appear as a statement in a 9010A program.
- The programmer is responsible for ensuring that the contents of the indicated file can legally be inserted at that point in the source file.
- A standard use of the INCLUDE statement is to include a pod data file.

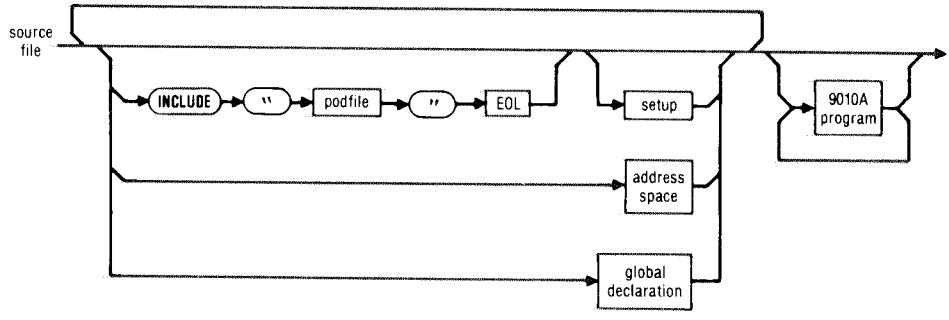
SOURCE FILE SYNTAX

The following pages contain reference information on source file syntax. For more explanation about a specific topic, refer to Section 4, Writing Programs.

Source File Syntax contains the following syntax diagrams:

SOURCE FILE
SETUP
ADDRESS SPACE
ADDRESS DESCRIPTOR
GLOBAL DECLARATION
SYMBOLIC REGISTER NAME DECLARATION
9010A PROGRAM
PROGRAM BODY
LOCAL DECLARATION
BINARY PROGRAM
INCLUDE DIRECTIVE

Syntax



Function

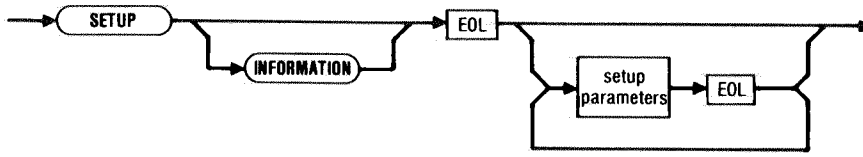
This syntax diagram defines the overall structure of the source file.

- The appropriate pod data file must be included if your programs have any pod dependencies.
- At this time, the pod data file must be one of the following (more files will be added as new interface pods are implemented):

1802.POD	8080.POD
6502.POD	8085.POD
6800.POD	8086.POD
68000.POD	8086MX.POD
6802.POD	8088.POD
6809.POD	8088MX.POD
6809E.POD	9900.POD
8041.POD	Z80.POD
8048.POD	

- The setup information, address space information, and global declarations are all optional. They may appear more than once, and they may appear in any order, providing that they appear before the first 9010A program.

Syntax



Function

Allows the user to control the reporting of UUT errors, enable microprocessor lines, and specify operating parameters.

- All setup parameters must be declared at the beginning of the source file preceding all programs.
- Setup parameters establish initial setup conditions only.
- Setup parameters are divided into the following categories:

1. Reporting UUT errors or enabling microprocessor lines:
   POD
   TRAP
   ENABLE
   EXERCISE ERRORS
   BEEP ON ERR TRANSITION

2. Specifying operating parameters:
   BUS TEST
   RUN UUT
   TIMEOUT

3. Relating to operation of the AUX I/F:
   STALL
   UNSTALL
   NEWLINE
   LINESIZE

Detailed information about setup parameters is contained in the next part of this section, Setup Parameters.

- The compiler supplies default values (as listed in Appendix D) for any setup parameters that do not explicitly appear in the source file.

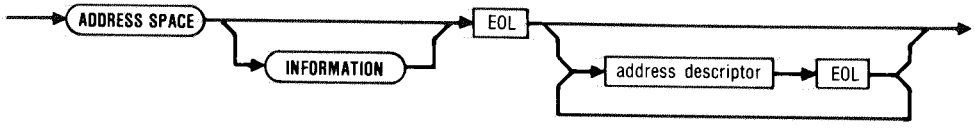
- The compiler default values for setup parameters can be overridden by the pod-specific values by including the appropriate pod data file.

See Also

Default Setup Parameters (Appendix D), Setup Parameter Limits (Appendix E), Pod Data Files (Section 4, Part 1)

ADDRESS SPACE

Syntax



Function

Forms the UUT memory map; identifies address blocks of RAM, ROM, and I/O.

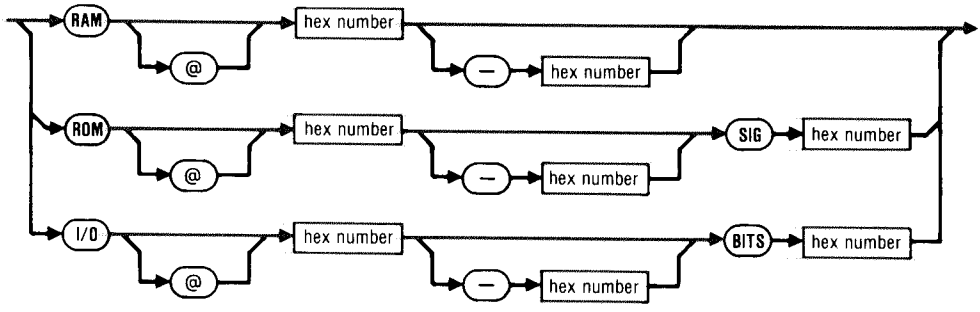
- All address descriptors must be declared at the beginning of the source file, preceding all programs.
- Up to 100 address descriptors can be specified in the source file.

Example

```
ADDRESS SPACE INFORMATION
  RAM @ 5000-50FF

  ROM @ 0000-0FFF SIG 0D47
  ROM @ 3000-4FFF SIG 2B60
  ROM @ 7000-70FF SIG 08AA
  ROM @ A000-AFFF SIG 44C9

  I/O @ 1A00-1A01 BITS 7F
```

ADDRESS DESCRIPTOR

Syntax



Function

Forms the UUT memory map; identifies address block of RAM, ROM and I/O.

- In a 9010A program statement, if a RAM, ROM, or I/O test is specified but the address range to be tested is not specified, the 9010A performs the specified test over all blocks of the appropriate memory type described by the address descriptors.

- Parameters and limits are as follows:

PARAMETER	LIMIT
signature (ROM)	0-FFFF
bit mask (I/O)	1-FFFFFFFF

Example

RAM @ 5000-50FF

ROM @ 0000-0FFF SIG 0D47
ROM @ 3000-4FFF SIG 2B60
ROM @ 7000-70FF SIG 08AA
ROM @ A000-AFFF SIG 44C9

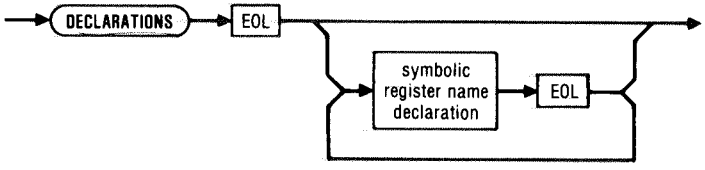
I/O @ 1A00-1A01 BITS 7F

See Also

LEARN, RAM TEST, ROM TEST, IO TEST (in 9010A Program Statements part of this section)

GLOBAL DECLARATION

Syntax



Function

Allows the programmer to define symbolic register names with global scope.

- Names with global scope are known throughout the entire source file and all files that are included after the global declarations.
- If a register name is redefined locally (inside a 9010A program), the local definition overrides the global definition and the program has no knowledge of the global declaration.
- Global declarations must appear at the beginning of the source file, before the first 9010A program is encountered.
- Global symbolic register declarations are restricted to the global registers (8-F).

Example

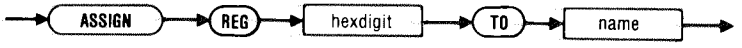
```
DECLARATIONS
  ASSIGN REG8 TO LOAD
  ASSIGN REG9 TO FLAG
PROGRAM U10
  .
  .
  .
```

See Also

SOURCE FILE, SYMBOLIC NAMES, SYMBOLIC REGISTER NAME DECLARATION, LOCAL DECLARATION

SYMBOLIC REGISTER NAME DECLARATION

Syntax



Function

Declares a symbolic name that the programmer uses in programs to refer to the indicated register.

- Symbolic register names must be declared in the global or local declarations section of the source file prior to being used in a program.
- Symbolic register names can be used wherever a register reference can be made (including AUX and DPY statements).
- Several symbolic names can be assigned to the same register.

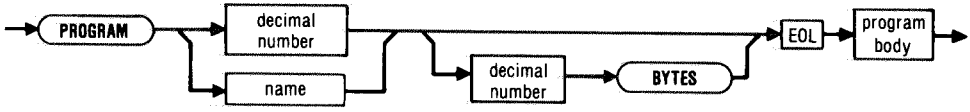
Example

```
DECLARATIONS
  ASSIGN REG1 TO TEMP, FLAG
  ASSIGN REGA TO PINNO
```

See Also

GLOBAL DECLARATION, LOCAL DECLARATION, SYMBOLIC NAMES, Predefined Register Names (in Section 4, Part 3)

Syntax



Function

This syntax diagram defines the overall structure for a 9010A program.

- Program numbers must be decimal numbers in the range 0-99.
- If a byte count appears in the program statement, the compiler compares it to the actual byte count and issues a warning message if the byte counts differ.
- Symbolic program names can be used in this statement.
- The source file can contain no more than one hundred 9010A programs.
- Numbered programs must appear in the correct order. If programs with symbolic names are combined with numbered programs, there must be a correct number of symbolically named programs between numbered programs. For example, if there are two numbered programs, program 4 and program 7, then there is room for only two symbolically named programs between them.

Example

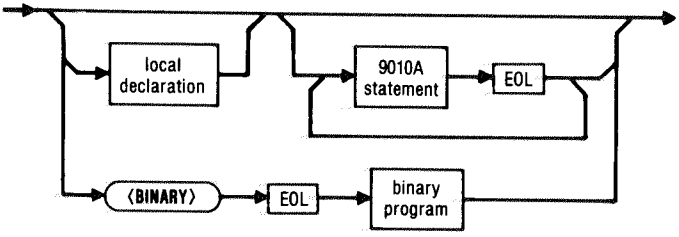
```
PROGRAM 35 728 BYTES
PROGRAM GETSIG
PROGRAM KEYBD TST
```

See Also

EXECUTE, Symbolic Program Names (Section 4, Part 3)

PROGRAM BODY

Syntax



Function

This syntax diagram defines the body of a 9010A program.

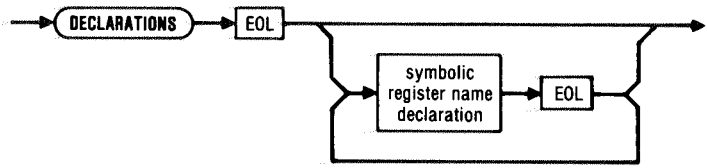
The details of the 9010A statements are provided in the 9010A Program Statements portion of this section.

See Also

LOCAL DECLARATION, 9010A PROGRAM STATEMENTS, BINARY PROGRAM

LOCAL DECLARATION

Syntax



Function

Allows the programmer to define symbolic register names with local scope.

- Names with local scope are known only within the program in which they are declared.
- Duplicate local names in different programs are unrelated.
- Local declarations must appear between the program statement and the first statement of the 9010A program body.
- No local declarations may appear inside a binary program.
- Symbolic names may be declared locally for all registers (0-F).

Example

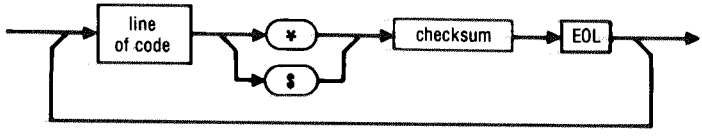
```
PROGRAM UUTTEST

DECLARATIONS
  ASSIGN REG1 TO ERRCNT
  ASSIGN REG2 TO PINCNT, SETBIT

  ERRCNT = 0
  SETBIT = 4
```

See Also

SOURCE FILE, SYMBOLIC REGISTER NAME DECLARATION, GLOBAL DECLARATION

BINARY PROGRAM

Syntax



Function

The 9000A Utility Program tape contains binary programs.

- Binary programs are introduced by the standard program statement (PROGRAM xx), followed on a separate line by ⟨BINARY⟩, followed by the binary program.
- A binary program contains lines of hex code. Each line is terminated by a one-byte checksum.
- A "*" is used to delimit a line of code from the checksum, except for the last line of the program where a "$" is used.
- The file transfer program (XFER) automatically reformats binary programs into the required format when they are transferred from the 9010A to the host system in source form.

Example

PROGRAM 10
⟨BINARY⟩

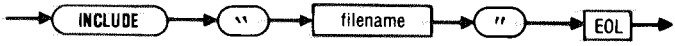
*514F50DDE5DD2A2B00DD562FDD5E2E7BE60F87874F0600FD2A2B00FD097BE6F0*28*
*CB3FCB3F4FDD097AE60F87874F2A2B0009DD7E02FD86025FDD7E03FD8E0357DD*D4*
*7E00FD8E0077DD7E01FD8E01237723732372DDE1010000C9284329464C554B65*C0*
205645522031SN$5E

See Also

9010A PROGRAM

INCLUDE DIRECTIVE

Syntax



Function

Replaces the INCLUDE "filename" statement with the contents of the indicated file. Equivalent to manually typing the contents of the included file in the source file at that point.

- The filename must be the name of an existing file.
- If the host computer system is case-sensitive regarding filenames, then the filename must be properly capitalized.
- A source file may include a file which in turn includes another file. Attempting to nest include files too deeply will result in a 9010A error message.
- Include directives must be on a line by themselves but can occur anywhere in the source file. Include directives may even appear as a statement in a 9010A program.
- The programmer is responsible for ensuring that the contents of the indicated file can legally be inserted at that point in the source file.
- A standard use of the INCLUDE statement is to include a pod data file.

Example

include "1802.POD"

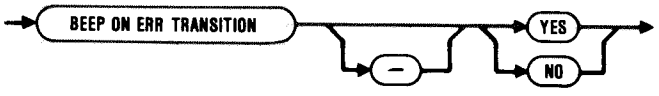
See Also

Pod Data Files (in Section 4, Part 1)

SETUP PARAMETERS

CONTENTS

Syntax



Function

Allows the programmer to control whether or not the 9010A should beep on ERR TRANSITIONS.

- YES enables the audible beep that sounds whenever an error is detected and reported. The beep also sounds whenever the error is removed.
- The 9010A's default value is YES.

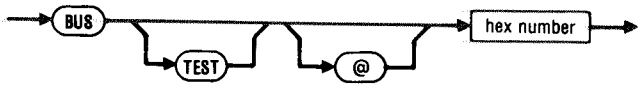
Example

BEEP ON ERR TRANSITION - NO

See Also

EXERCISE ERRORS, TRAP

Syntax



Function

When the Bus Test is performed in a 9010A program, testing of data lines occurs at the address listed.

- Setup parameter limits for Bus Test are 0-FFFFFFFF. Refer to the pod instruction manuals for legal addresses.
- If the Bus Test statement appears in the Setup Parameters section of the source file, then the default Bus Test address is as indicated.
- If this statement was not present and a pod data file was included at the beginning of the source file, the compiler supplies the definition for BUSADR.
- If a pod data file was not included at the beginning of the source file, the default Bus Test address is 0000.

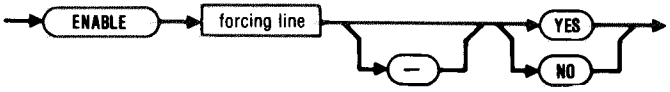
Example

BUS TEST @ 1C00

See Also

Pod Data Files (in Section 4, Part 1), BUS TEST (in 9010A Program Statements part of this section), and Appendix D (Pod-Specific Setup Parameters)

Syntax



Function

Allows an operator to individually enable or disable pod forcing lines.

- If YES is selected, the forcing line is enabled.
- If NO is selected, the forcing line is disabled.
- Forcing lines are pod-specific and include lines such as the following:

WAIT	BR/ACK	READY
RDY	INTR	BUSRQ
TSC	MR	HOLD
DBE	DMA	RQGT0
HALT	UNUSED	RQGT1

- There are a maximum of eight enableable forcing lines. Refer to the pod instruction manuals for specific information.
- The appropriate pod data file must be included prior to the appearance of any ENABLE statements. In addition, a POD statement identifying the pod should appear in the Setup Parameters section of the source file.
- If a pod data file was included at the beginning of the source file, the forcing lines listed in the definition for FORCELNS will all have default values of YES.

Example

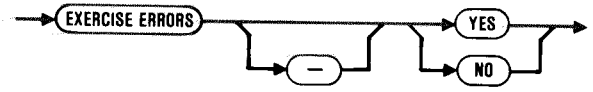
ENABLE HALT - NO

See Also

Pod Data Files (in Section 4, Part 1) and Appendix D (Pod-Specific Setup Parameters), POD

EXERCISE ERRORS

Syntax



Function

Allows the operator control over 9010A error reporting and interactive handling of errors.

- If YES is selected, the 9010A displays detected error messages and prompts the operator to loop on the errors.
- If NO is selected, the errors are not reported to the operator, but error messages are transmitted to the RS-232 if it is connected (without the -LOOP? portion of the message).
- The 9010A's default value is YES.

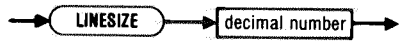
Example

EXERCISE ERRORS - NO

See Also

BEEP, TRAP

Syntax



Function

Allows the programmer to specify the maximum number of characters transmitted per line when the 9010A is sending data through the AUX I/F.

- Setup parameter limits for LINESIZE are 10-255.
- The LINESIZE used is determined by the line size of your remote device.
- The 9010A's default value is 79.

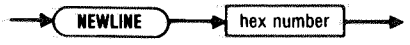
Example

LINESIZE 120

See Also

NEWLINE, STALL/UNSTALL

Syntax



Function

When the 9010A is sending data through the AUX I/F, a terminator sequence is sent at the end of each line. This statement allows the programmer to specify both the ASCII terminator characters to be sent and the delay between lines.

- Setup parameter limits for NEWLINE are eight hexadecimal digits.
- The 9010A default value is 00000D0A.
- The selection of the terminator sequence allows the operator to meet the needs of a wide variety of remote devices. For example, if the remote device provides its own Linefeed at the end of each line, the terminator sequence would consist of only the Carriage Return (00000D). Or, if a double space is needed between lines, the terminator sequence would be a Carriage Return and two Linefeeds (0D0A0A).
- The eight hexadecimal digits have the following meaning:

  First two digits: These may have any hexadecimal value between 0 and FF. They must be followed by six digits as described below. The two digits represent a count that corresponds to a timing delay between the transmission of lines. For 9010A versions prior to 2C, the timing delay is approximately 2.4 ms/count, providing a total timing delay range of 0 to .6 seconds. The delay is 6 ms/count, for maximum delay of approximately 1.5 seconds with 9010A versions 2C and later.

  Last six digits: These are the ASCII terminator characters which are sent at the end of each line when the 9010A is sending data. The characters are also sent once as the initial trigger when the AUX I/F READ operation is selected. The characters, which have two digits each, are sent left to right. Zeros are not sent.

NEWLINE

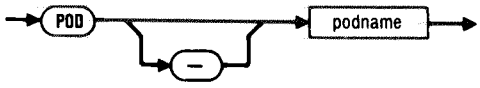
Example

```
NEWLINE 000D0A0A  ! terminator sequence of a carriage
                    return and 2 linefeeds
NEWLINE 00000D0A  ! terminator sequence of a carriage
                    return and I linefeed
NEWLINE 1A00000D  ! terminator sequence of a time delay
                    and carriage return
```

See Also

LINESIZE, STALL/UNSTALL

Syntax



Function

Identifies the pod to be used when executing the 9010A programs in the source file. The POD statement allows the 9010A to use the data in the pod data file to configure its setup parameters to match the specified pod.

- At this time podname is one of the following (more files will be added as new interface pods are implemented):

1802	6809	'40/50	8088
6502	6809E	8080	8088MX
6800	8041	8085	9900
68000	'35/48	8086	Z80
6802	'39/49	8086MX	

- When using the 8048 pod, the podname must be listed in this statement as '35/48, '39/49, or '40/50, as appropriate.

Example

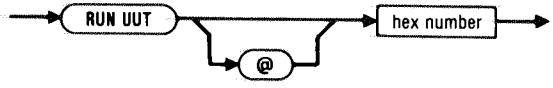
POD - 8080

POD '39/49

See Also

Pod Data Files and 9010A Pod Interaction (in Section 4, Part 1)

Syntax



Function

Used when the address for a RUN UUT operation is allowed to default in a 9010A program.

- Setup parameter limits for RUN UUT are 0-FFFFFFFF.
- If the RUN UUT statement appears in the setup parameters section of the source file, then the RUN UUT address will be as indicated.
- If this statement was not present and a pod data file was included at the beginning of the source file, the compiler supplies the definition for UUTADR.
- If a pod data file was not included at the beginning of the source file, the default RUN UUT address is 0000.

Example

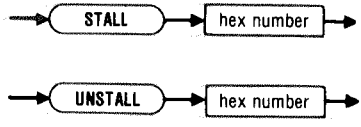
RUN UUT @ C000

See Also

Pod Data Files (in Section 4, Part 1), RUN UUT (in 9010A Program Statements part of this section), and Appendix D (Pod-Specific Setup Parameters)

STALL
UNSTALL

Syntax



Function

Allows the programmer to specify the Stall and Unstall characters (X-ON and X-OFF) to which the 9010A responds when it is sending data through the AUX I/F.

- Setup parameter limits for Stall and Unstall are 0-FF.
- Any ASCII character may be selected for the Stall and Unstall characters. The characters are specified with their hexadecimal ASCII values. The characters used are those that are required by your remote device.
- The 9010A's default values are as follows:

STALL 13	(CTRL S)
UNSTALL 11	(CTRL Q)

Example

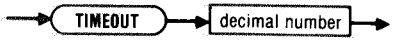
STALL 13

UNSTALL 11

See Also

LINESIZE, NEWLINE

Syntax



Function

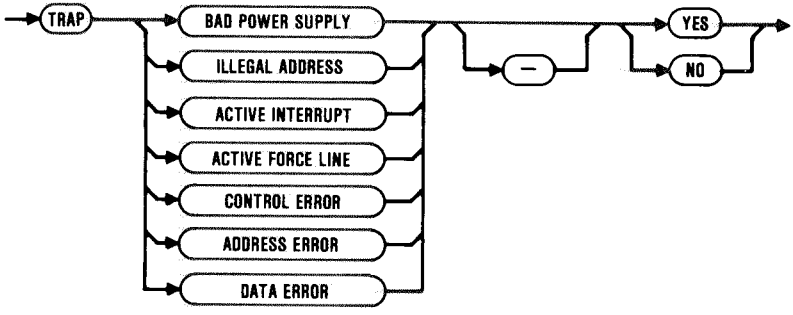
Represents a count of how long the 9010A waits before timing out on an interface pod operation.

- Setup parameter limits for TIMEOUT are 0-60000.
- The 9010A's default value is 200.

Example

TIMEOUT - 200

Syntax



Function

Allows the operator to individually enable or disable traps on UUT system errors.

- If YES is selected, the UUT system error is reported to the operator as it occurs.
- IF NO is selected, the UUT system error is not reported to the operator as it occurs.
- Any error types not explicitly specified are set to the 9010A default values.
- The 9010A's default values are as follows:

TRAP BAD POWER SUPPLY	YES
TRAP ILLEGAL ADDRESS	YES
TRAP ACTIVE INTERRUPT	NO
TRAP ACTIVE FORCE LINE	YES
TRAP CONTROL ERROR	YES
TRAP ADDRESS ERROR	YES
TRAP DATA ERROR	YES

Example

TRAP BAD POWER SUPPLY - NO

TRAP ACTIVE INTERRUPT - NO

See Also

EXERCISE ERRORS, BEEP

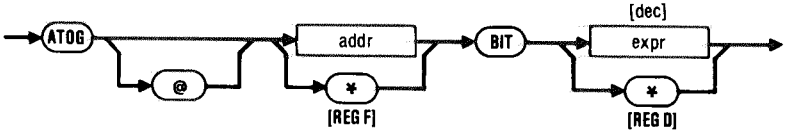
9010A PROGRAM STATEMENTS

CONTENTS

The syntax diagrams for the 9010A program statements are arranged alphabetically on the following pages. The functional groupings of the statements are as follows:

FUNCTION	STATEMENT
TESTS	AUTO TEST BUS TEST IO TEST RAM TEST ROM TEST
TROUBLESHOOTING	ATOG DTOG RAMP READ WALK WRITE
MODE	REPT/LOOP RUN UUT STOP
TEST SEQUENCING	AUX DPY EXECUTE GOTO IF LABEL
UUT MEMORY MAPPING	LEARN
PROBE	PROBE SYNC
REGISTER OPERATION	REG UNARY (CPL, DEC, INC, SHL, SHR)

Syntax



Function

Toggles an operator-specified address bit from one logic state to another. Two read operations are performed, one at the original address and another after the bit is toggled.

- If the bit number is explicitly specified in the expression, it must have a decimal value in the range 0 - (n–1) where n equals bits in the address bus.

Example

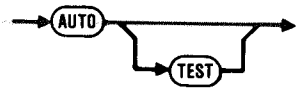
ATOG @ 13FC BIT 7

See Also

DTOG, RAMP, READ, WALK, WRITE

AUTO TEST

Syntax



Function

Performs in sequence Bus Test, ROM Test, RAM Short Test, and IO Test for versions prior to 2C. For versions 2C and later, the sequence is Bus Test, RAM Short Test, ROM Test, and IO Test.

- Errors are reported and locations are identified as described for the individual tests.

Example

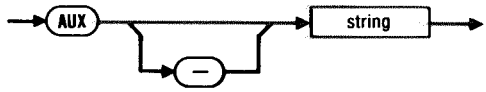
AUTO TEST

AUTO

See Also

BUS TEST, IO TEST, RAM TEST, ROM TEST

Syntax



Function

Allows for sending and receiving data between the 9010A and other devices using the RS-232 Interface Option.

- The string parameter represents the text to be sent.
- The text is separated from the AUX keyword by a single space, hyphen, or tab.
- Any spaces beyond the single separating character are treated as part of the display message, resulting in leading blanks.
- The AUX string can contain a maximum of 32 characters.
- Spaces at the end of an AUX string are ignored. If trailing blanks are desired, the appropriate number of underscores should be appended to the AUX string.
- Characters allowed in the AUX string are limited to those available on the 9010A. The valid characters are:

A-Z	+
0-9	-
;	'
@	%
=	*
<	\
>	/
,	"
.	$
?	space
#	_ (underscores will be converted to spaces)

- The functions of the special AUX I/F characters are shown on the next two pages. Symbolic register names can be used with these special AUX characters. Symbolic register names are counted as one character in the AUX string.

- A symbolic register name cannot be immediately followed by a hexadecimal character (0-9, A-F). A separating space is required.

Example

AUX ROM SIGNATURE IS $ROMSIG1 — *ROMSIG is a symbolic register name. The string to be sent is "ROM SIGNATURE IS" followed by the hexadecimal contents of ROMSIG.*

AUX - tests complete

See Also

DPY

Functions of AUX I/F Characters

CHARACTER	ACTION CAUSED
#	Sends a control G (bell) to the RS-232 interface.
$	When followed by a hexadecimal digit or symbolic register name, $ causes the contents of the designated register to be transmitted in hexadecimal to the RS-232 interface.
@	The same as for the $ symbol, except that the contents are transmitted in decimal.
/	When / is followed by a hexadecimal digit or symbolic register name, it suspends program execution, waits for the next byte of data from the RS-232 interface, and places the value of the byte in the designated register. (The upper three bytes of the register equal zero.) If the RS-232 interface is configured to transfer eight data bits, then eight data bits appear. Otherwise, the eighth data bit (bit 7) is zero.

When \ is followed by a hexadecimal digit or symbolic register name, it places the status of the RS-232 interface in the lower five bits of the designated register. (The upper 27 bits are zero.) The five status bits are as follows:

Bit 0:	1 = Parity Error 0 = No Parity Error
Bit 1:	1 = Framing Error 0 = No Framing Error
Bit 2:	1 = Overrun Error 0 = No Overrun Error
Bit 3:	Status of Receive Buffer 1 = Character Received 0 = No Character Received
Bit 4:	Status of Transmit Buffer 1 = Transmit Buffer is Empty; Ready for Next Character 0 = Character Still Being Sent

%

When % is followed by a hexadecimal digit or symbolic register name, it transmits the low-order byte contained in the designated register. This provides a way for the programmer to send the full range of ASCII characters (including characters not usually allowed in an AUX string) to the AUX I/F. Eight data bits are sent if the RS-232 interface is configured to transfer eight bits.

+

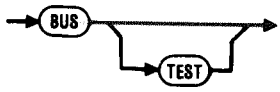
When + is the last character in an AUX I/F step, it prevents the NEWLINE termination sequence from being sent at the end of the line.

NOTE: In order to cause one of the special symbols $, @, /, \, or % to be sent to the RS-232 interface in the case where the symbol is followed by a hexadecimal digit or symbolic register name, the symbol must appear twice in the specification.

EXAMPLE:

STATEMENT	TEXT SENT
AUX $1	(contents of REG1)
AUX $$1	$1
AUX $X	$X

Syntax



Function

Tests for proper function of the UUT control lines, data lines, and address lines.

- When Bus Test is performed, testing of data lines occurs at the address specified in the Bus Test setup parameter.

Example

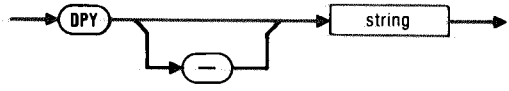
BUS TEST

BUS

See Also

AUTO TEST, IO TEST, RAM TEST, ROM TEST, and BUS TEST (in Setup Parameters part of this section)

Syntax



Function

Displays the string on the 9010A.

- Text to be displayed is separated from the DPY keyword by a single space, hyphen, or tab.
- Any spaces beyond the single separating character are treated as part of the display message resulting in leading blanks.
- The DPY string can contain a maximum of 32 characters.
- Spaces at the end of a DPY string are ignored. If trailing blanks are desired, the appropriate number of underscores should be appended to the DPY string.
- Characters allowed in the DPY string are limited to those available on the 9010A. The valid characters are:

A-Z	+
0-9	-
;	,
@	%
=	*
<	\
>	/
'	"
.	$
?	space
#	_ (underscores will be converted to spaces)

- The functions of the special DPY characters are shown on the next page. Symbolic register names can be used with these special DPY symbols. The symbolic register names are counted as one character in the DPY string.
- A symbolic register name cannot be immediately followed by a hexadecimal character (0-9, A-F). A separating space is required.

DPY

Example

DPY - test 3 complete - pass
*DPY - trailing blank_*

See Also

AUX

Functions of DPY Characters

CHARACTER	ACTION CAUSED
#	Causes the 9010A to beep when DPY is executed. This symbol does not appear on the display when DPY is executed.
$	When $ is followed by a hexadecimal or symbolic register name, it causes the contents of the designated register to be displayed in hexadecimal on the display.
@	The same as for the $ symbol except that the contents are displayed in decimal.
/	When / is followed by a hexadecimal digit or symbolic register name, it suspends program execution and waits for input. When the operator enters a hexadecimal value terminated by ENTER, the 9010A places the value in the designated register and resumes program execution. Pressing ENTER without specifying a hexadecimal value causes the value to default to the previous contents of the register.
\	The same as for the / symbol, except that the 9010A accepts only a decimal entry.

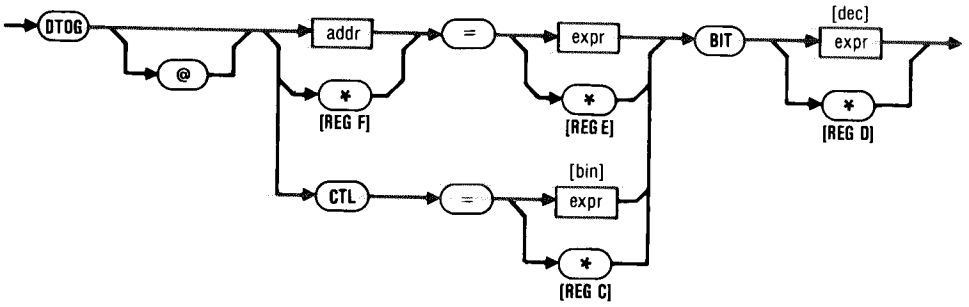
?	When ? is followed by a hexadecimal digit or symbolic register name, it suspends program execution and displays the question mark (?). If the operator presses the CLEAR/NO key, the 9010A places a 0 in the designated register. If the operator presses the ENTER/YES key, the 9010A places a 1 in the designated register. After the 1 or 0 is placed in the register, the 9010A removes the question mark and then resumes program execution.
%	When % is followed by a hexadecimal digit or symbolic register name, it enables or disables asynchronous input from the operator during execution. Asynchronous input is stored in the register designated by the hexadecimal digit or symbolic register name
+	When + is the first character in the specification, it causes following characters in the specification to be appended to the text that is on the display at the time DPY is executed.

NOTE: In order to cause one of the special symbols $, @, /, \, ?, or % to be displayed in the case where the symbol is followed by a hexadecimal digit or symbolic register name, the symbol must appear twice in the specification.

EXAMPLE:

STATEMENT	TEXT DISPLAYED
DPY $1	(contents of REG1)
DPY $$1	$1
DPY $X	$X

Syntax



Function

Toggles a programmer-specified data bit from one binary logic state to another by performing two write operations at a programmer-specified address.

The DTOG @ CTL function toggles a programmer-specified control line from one binary logic state to another.

- If the DTOG @ CTL form is used and the expression immediately following the equal sign (=) is specified explicitly, the expression must be a binary value from 0 to 11111111.
- If the address (not the DTOG @ CTL form) is specified, then the following bit number expression (after BIT) must have a decimal value in the range 0-(n-1) where n equals the number of bits in the microprocessor data bus.
- In the DTOG @ CTL form, if an expression is used to specify the bit number, it must have a decimal value in the range 0-7.
- Refer to the pod instruction manuals or the label on the interface pod to identify which control lines are user-writable for a specific pod.

Example

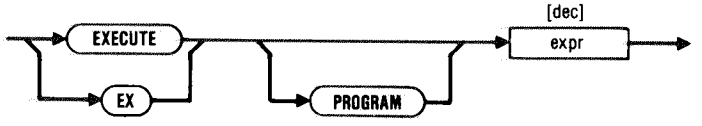
DTOG @ REGF = FF BIT REG3

DTOG @ CTL = 01011111 BIT 5

See Also

ATOG, RAMP, READ, WALK, WRITE

Syntax



Function

Executes one program from within another program in a subroutine-like fashion.

- Program numbers are limited to the range 0-99.
- A program may call a program which in turn calls another program. Programs may be called up to ten levels of nesting.
- If multiple levels of programs are called, a program may not call any program from a previous level.
- A program may not call itself.
- Symbolic program names can be used in this statement.
- The compiler issues a warning message if you attempt to execute a program that is not contained in the files being compiled.

Example

EXECUTE PROGRAM 5

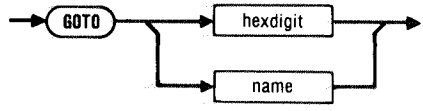
EX 5

EXECUTE DELAY

See Also

PROGRAM

Syntax



Function

Allows the programmer to construct GOTO (unconditional branch) steps which redirect program execution to a label in the program.

- Symbolic label names can be used in this statement.
- Within a single program, symbolic names cannot be mixed with hexadecimal label numbers (0-9, A-F).
- More than one GOTO step may redirect program execution to the same label.
- The label to which program execution is redirected may appear anywhere in the program.

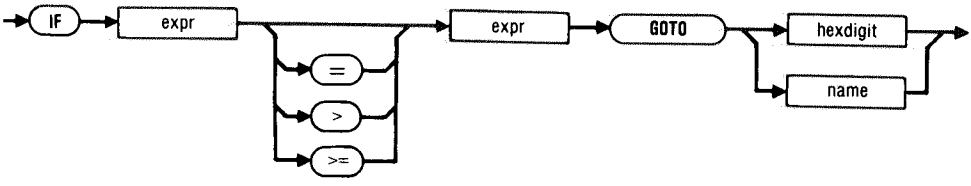
Example

GOTO 3

See Also

LABEL, IF

Syntax



Function

Creates conditional branch steps.

- Symbolic label names can be used in this statement.
- Within a single program, symbolic labels cannot be mixed with hexadecimal label numbers (0-9, A-F).
- More than one IF step may redirect program execution to the same label.
- The label to which program execution is redirected may appear anywhere in the program.

Example

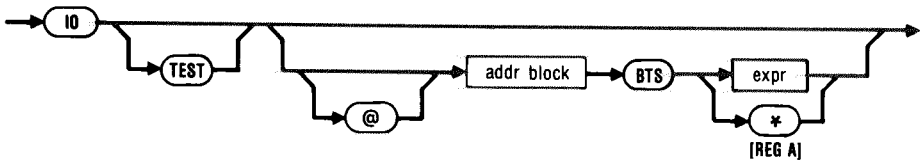
IF REG3 AND 7F > REG4 GOTO 1

See Also

GOTO, LABEL

IO TEST

Syntax



Function

Tests the read-write capability of all bits in I/O registers described as having read-write capability.

- If an expression is used to specify the bit mask (following BTS), it must have a hexadecimal value in one of the following ranges:

1-FF	8-bit microprocessor
1-FFFF	16-bit microprocessor
1-FFFFFF	24-bit microprocessor
1-FFFFFFFF	32-bit mocroprocessor

- Bits that are equal to 1 in the bit mask correspond to data lines that are to be tested for read-write capability. Bits that are equal to 0 in the bit mask correspond to data lines that are not to be tested for read-write capability.

- If no address block is specified, then the 9010A performs the specified IO TEST over all blocks of memory described as I/O under Address Space Information.

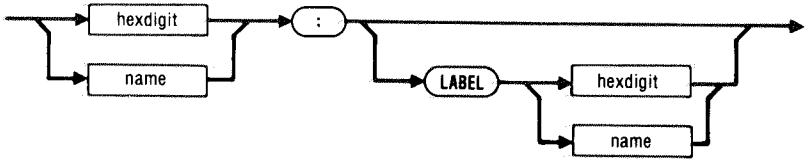
Example

IO TEST @ 4010 - 401F BTS 3D

See Also

AUTO TEST, BUS TEST, RAM TEST, ROM TEST, LEARN, ADDRESS DESCRIPTOR (in Source File Syntax part of this section)

Syntax



Function

Allows the programmer to create labels, i.e., program steps inserted into programs to provide points of entry for branching steps. Identifies a specific location in a program.

- Each label is identified by a single hexadecimal digit (0-9 and A-F) or with a symbolic name.
- Within a single program, symbolic label names cannot be mixed with hexadecimal label numbers (0-9, A-F).
- 9010A Language keywords must not be chosen as symbolic label names (such as LOOP).
- There are 16 possible labels for each program.
- All label names must be distinct.
- Labels may appear in any order.
- A label may exist without a branch (GOTO) step to the label.
- A 9010 program statement can follow the colon.

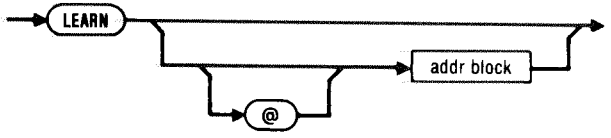
Example

```
L1:
DONE: STOP
FOUND: LABEL FOUND
```

See Also

GOTO, IF

Syntax



Function

Tests each address location in sequence and identifies it as RAM, ROM, I/O, or unassigned. Also creates an address descriptor for each block of memory which was identified.

- If no addr block is specified, the Learn operation is performed on the entire microprocessor address space. Refer to the pod instruction manuals for specific address information.

Example

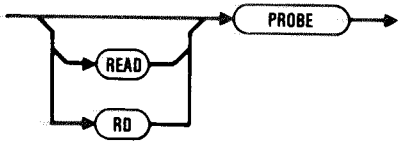
LEARN

LEARN @ 1000 - 4FFF

See Also

IO TEST, RAM TEST, ROM TEST, ADDRESS DESCRIPTOR (in Source File Syntax part of this section)

Syntax



Function

The Read Probe function places accumulated probe data into Register 0. Probe data consists of the logic levels detected, the number of events counted, and the signature computed at the probe tip.

In Register 0, event counts are assigned to bits 0-6, signatures are assigned to bits 8-23, and logic levels are assigned to bits 24-26.

Example

READ PROBE

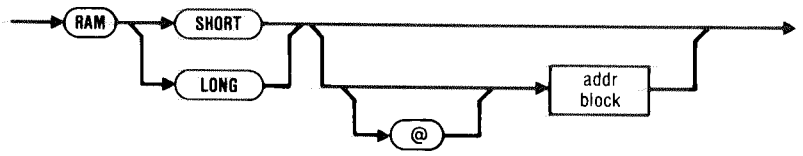
RD PROBE

PROBE

See Also

SYNC

Syntax



Function

RAM SHORT quickly identifies common RAM failures such as address decoding errors or bits that are not read-writable. RAM LONG performs the same tests as RAM SHORT and in addition, performs a pattern-sensitivity test for locating "soft" RAM errors.

- If no address block is specified, then the 9010A performs RAM test over all blocks of memory specified as RAM under the Address Space Information.

Example

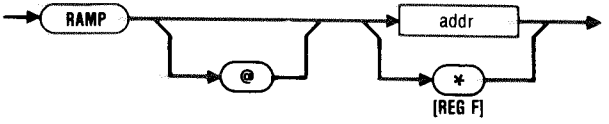
RAM SHORT @ 1000 - 3FFF

RAM LONG

See Also

AUTO TEST, BUS TEST, IO TEST, ROM TEST, LEARN, ADDRESS DESCRIPTOR (in Source File Syntax part of this section)

Syntax



Function

Performs a series of write operations at a programmer-specified location in the UUT microprocessor system, beginning with all data bits equal to zero, and increasing by one until all data bits equal one.

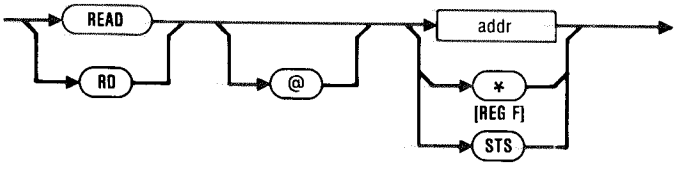
Example

RAMP @ 34F0

See Also

ATOG, DTOG, READ, WALK, WRITE

Syntax



Function

Reads a programmer-specified location in the UUT microprocessor system and places the data in register E.

READ STS reads the values of the UUT microprocessor status lines and places the corresponding value in register C.

Example

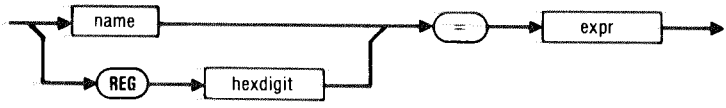
READ @ REG1

RD STS

See Also

PROBE, READ, WRITE, ATOG, DTOG, RAMP, WALK

Syntax



Function

Enters the specified data in the specified register.

- Symbolic register names can be used in this statement.
- Symbolic register names must be declared before use in the local or global declarations section.

Example

REG1 = 1FF

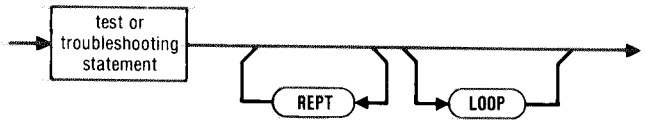
TMP = REGA SHR 4

See Also

SYMBOLIC REGISTER NAME DECLARATION (in General Information part of this section)

REPT
LOOP

Syntax



Function

REPT causes the action previously performed to be repeated once. LOOP causes the action previously performed to be repeated continuously.

- REPT and LOOP may not be specified as steps by themselves but may be specified as modifiers after a troubleshooting test or function has been specified.

- REPT and/or LOOP can follow these test or troubleshooting statements:

AUTO TEST	READ
BUS TEST	WRITE
RAM TEST	RAMP
ROM TEST	WALK
IO TEST	ATOG
	DTOG

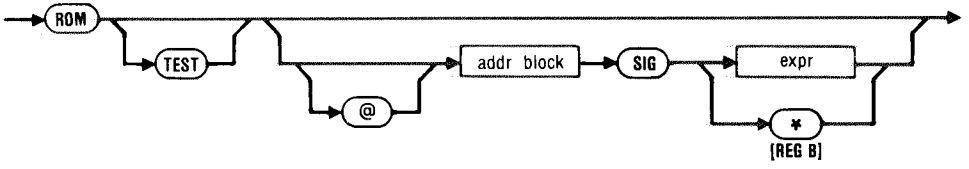
Example

RAMP @ REGF REPT REPT

WALK @ 401C = 1 LOOP

ROM TEST

Syntax



Function

Computes a ROM signature for each block of ROM and compares it to the reference ROM signature.

- If no address block is specified, then the 9010A performs a ROM Test over all blocks of memory specified under Address Space Description and compares the signatures to those specified in the Address Space Information.

- The signature expression must have a hexadecimal value in the range 0-FFFF.

Example

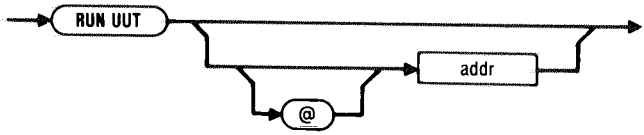
ROM TEST

ROM TEST @ 8000 - 9FFF SIG AFC7

See Also

AUTO TEST, BUS TEST, IO TEST, RAM TEST

Syntax



Function

Allows the interface pod microprocessor to execute the program code stored in the UUT.

- If an address is specified, the UUT begins executing the code at the address indicated.
- If no address is specified but a RUN UUT setup parameter is present, the address from the setup statement is supplied.
- If no RUN UUT statement appeared in the setup section, but a pod data file was included at the beginning of the source file, then the value for UUTADR will be supplied.
- If a pod data file was not included at the beginning of the source file, the default address is 0000.

Example

RUN UUT

RUN UUT @ 1000

See Also

Default Setup Parameters (Appendix D), RUN UUT (in Setup Parameters part of this section)

Syntax

Function

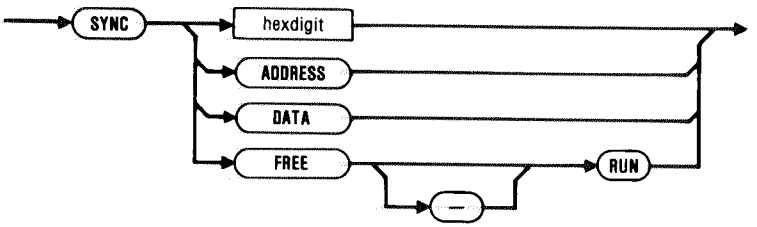
Suspends program execution at desired points.

- To cause the 9010A to resume program execution, the operator must press the CONT key.

Example

STOP

Syntax



Function

Enables the operator to synchronize the probe operation to events in the microprocessor bus or allow the probe to oscillate at 1 kHz (free run).

Example

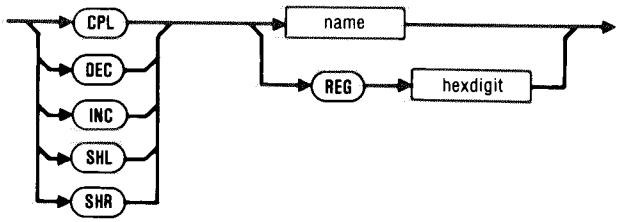
SYNC A

SYNC FREE-RUN

See Also

PROBE

Syntax



Function

Performs the specified unary operation on the contents of the indicated register.

- Symbolic register names can be used in this statement.
- Register identifiers must be previously declared in the local or global declaration section.
- Unary operator shorthand may not be used in this statement (i.e., INC 3 REG5 is a syntax error).

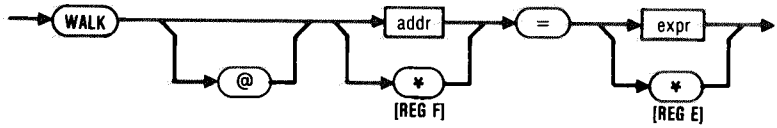
Example

```
INC REG7
INC ERRCNT      !ERRCNT is a symbolic register name
```

See Also

REG

Syntax



Function

Rotates a programmer-specified bit pattern across data lines by performing a series of write operations at a programmer-specified address. The process continues until the data bits are rotated through every possible position.

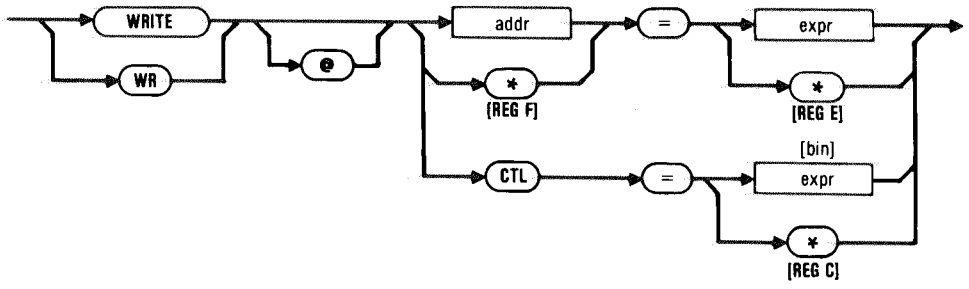
Example

WALK @ 34B0 = 7F

See Also

ATOG, DTOG, RAMP, READ, WRITE

Syntax



Function

Writes programmer-specified data to a programmer-specified location in the UUT microprocessor system.

WRITE @ CTL causes the 9010A to write control lines to the programmer-specified logic levels.

- If an expression is used with the CTL form, it must have a binary value from 0 to 11111111. The binary string corresponds to the eight possible UUT control lines. The 9010A forces control lines represented by a 1 high, and forces control lines represented by a 0 low.

- Refer to the pod instruction manuals or the label on the interface pod itself to identify which control lines are user-writable for a specific pod.

Example

WRITE @ 713B = 2F

WR CTL = 11000100

See Also

ATOG, DTOG, RAMP, READ, WALK

APPENDICES

CONTENTS

Appendix A
Keywords

* Identifies Setup Keywords

active *
address *
and
assign
atog
auto
aux

bad *
beep *
binary
bit
bits *
bts *
bus *
bytes

control *
cpl
ctl

data *
dec
declarations
dpy
dtog

enable *
err *
error *
errors *
ex
exercise *
execute

force *
free

goto

if
illegal *
inc
include
information *
interrupt *
IO

label
learn
line *
linesize *
long
loop

newline *
no *

on *
or

pod *
power *
probe
program

ram *
ramp
rd
read
reg
rept
rom *
run *

setup *
shl
short
shr
sig *
space *
stall *

stop
sts
supply *
sync

test *
timeout *
to
transition *
trap *

unstall *
uut *

walk
wr
write

yes *

Appendix B
Predefined Register Names

REGISTER	SYMBOLIC NAME	FUNCTION
A	BITMASK	Bit Mask
B	ROMSIG	ROM Signature
C	STSCTL	STS/CTL Information
D	BITNUM	Bit Number
E	DAT	Data
F	ADR	Address
0	PBDAT	Read Probe Data

Appendix C
Optional Keywords and Keyword Abbreviations

OPTIONAL KEYWORDS AND SYMBOLS

ELEMENT	RESTRICTIONS	TWO EQUIVALENT AND ACCEPTED STATEMENTS
TEST	None, always optional	AUTO TEST AUTO
@	None, always optional	WRITE @ 100FF = 25 WRITE 100FF = 25
—	Optional only in DPY, AUX, and SETUP parameters	DPY TEST MESSAGE DPY-TEST MESSAGE POD - 8080 POD 8080
LABEL x	None, always optional (used in LABEL statements)	3: LABEL 3 3:
PROGRAM	Optional only in EXECUTE PROGRAM xx commands	EXECUTE PROGRAM 35 EXECUTE 35
INFORMATION	None, always optional (used in Setup and Address Descriptor sections)	SETUP INFORMATION SETUP
READ	Optional only in READ PROBE command	READ PROBE PROBE
xx BYTES	None, always optional (used in program statements)	PROGRAM 10 524 BYTES PROGRAM 10

KEYWORD ABBREVIATIONS

KEYWORD	ABBREVIATION
SYNC ADDRESS	SYNC A
SYNC DATA	SYNC D
SYNC FREE-RUN	SYNC F
READ	RD
WRITE	WR
EXECUTE	EX

Appendix D
Default Setup Parameters

The information in the following table applies only to these pods:

1802, 6502, 6800, 68000, 6802, 6809/6809E, 8041/8048, 8080, 8085, 8086/8086MX, 8088/8088MX, 9900, Z80

Setup Parameters Common to All Pods Listed Above

PARAMETER	DEFAULT VALUE
TRAP BAD POWER SUPPLY	YES
TRAP ILLEGAL ADDRESS	YES
TRAP ACTIVE INTERRUPT	NO
TRAP ACTIVE FORCE LINE	YES
TRAP CONTROL ERROR	YES
TRAP ADDRESS ERROR	YES
TRAP DATA ERROR	YES
EXERCISE ERRORS	YES
BEEP ON ERR TRANSITION	YES
TIMEOUT	200
STALL	13
UNSTALL	11
NEWLINE	00000D0A
LINESIZE	79
*BUS TEST @	0000
*RUN UUT @	0000

*If a pod name is not specified in the setup parameter section of the source file, then the default address for BUS TEST and RUN UUT are as indicated. If a pod data file is included and the pod name is specified or if a pod is connected to the 9010A when the hex file is downloaded, then the specified pod's default BUS TEST and RUN UUT addresses will override these.

POD-SPECIFIC SETUP PARAMETERS

POD	BUS TEST @	RUN UUT @	ENABLEABLE LINE	DEFAULT VALUE
1802	FFFF	0000	WAIT	YES
6502	0000	FFFFFFFC	RDY	YES
6800	0000	FFFFFFFE	TSC	YES
			DBE	YES
			HALT	YES
68000	1000FFE	F6000000	HALT	YES
			BR/ACK	YES
			INTR	YES
6802	0000	FFFFFFFE	MR	YES
			HALT	YES
6809	0000	FFFFFFFE	HALT	YES
			DMA	YES
			MR	YES
6809E	0000	FFFFFFFE	TSC	YES
			HALT	YES
8041	2000	3000	UNUSED	YES
8048	1100	0000	UNUSED	YES
8080	FFFF	0000	READY	YES
			HOLD	YES
8085	FFFF	0000	READY	YES
			HOLD	YES
8086	0000	FFFF0	READY	YES
			HOLD	YES
			INTR	YES
8086MX	0000	FFFF0	READY	YES
			RQGT0	YES
			RQGT1	YES
			INTR	YES
8088	0000	FFFF0	READY	YES
			HOLD	YES
			INTR	YES

8088MX	0000	FFFF0	READY	YES
			INTR	YES
			RQGT0	YES
			RQGT1	YES
9900	F000	0000	READY	YES
			HOLD	YES
Z80	FFFF	0000	BUSRQ	YES
			WAIT	YES

Appendix E
Parameter Limits

SETUP PARAMETER LIMITS

PARAMETER	LIMIT
BUS TEST	0-FFFFFFFF
RUN UUT	0-FFFFFFFF
STALL	0-FF
UNSTALL	0-FF
LINESIZE	10-255
TIMEOUT	0-60000
NEWLINE	8 Hexadecimal Digits

ADDRESS DESCRIPTOR PARAMETER LIMITS

PARAMETER	LIMIT
signature (ROM)	0-FFFF
bit mask (IO)	1-FFFFFFFF

Appendix F
Error Messages

INTRODUCTION

This appendix describes error messages that may be produced by the 9010A Language Compiler programs. The appendix is divided into three parts: Compiler Program Error Messages (9LC), File Transfer Error Messages (XFER), and Disk Verification Program Error Messages (VERIFY). Along with each error message is a description of possible causes for the error. The description is not meant to be a comprehensive list; other causes may also be possible.

Other messages may be produced by the host computer system. For explanations of system-dependent errors, refer to System Dependencies in Section 3 and to the user manual for the host system.

COMPILER PROGRAM ERROR MESSAGES

Address range error

In an address range, the second address was incorrectly specified smaller than the first address.

EXAMPLE: RAM @ 10000 - 10FF

Attempt to redefine symbolic name

A symbolic name was used in the wrong context (i.e., the name was already used as a program name, but now you are attempting to use it as a global register name, you are using a local register name as the target of a GOTO, or you are using a label name as a program name in an EXECUTE statement).

Binary number expected

Can occur if you try to write a non-binary value to CTL or try to DTOG a non-binary value for DTOG @ CTL.

Cannot define REG0-7 as global registers

You tried to assign a symbolic name to a local register (REG0-7) in a globally declared ASSIGN statement. You can only assign symbolic names to these registers locally.

Cannot open file ‹filename›

An illegal file name was entered.
You attempted to open a file for writing on a write-protected disk.
You attempted to open a file that does not exist.
You attempted to create a file on a full disk.
An Include file cannot be opened because it would result in more files being opened concurrently than your system allows.

Checksum error, should be xx

A checksum error was encountered.

Duplicate label

A label was used more than once.

Duplicate program

An attempt was made to compile a source file with two programs with the same number or same name.

Error in hex line

There was a missing character in a binary program.

Illegal address

An address with more than eight hexadecimal digits (past the 32-bit limit) was specified.

Illegal bitmask

A bitmask equal to 0, or with more than eight hexadecimal digits was specified.

Illegal bitnumber

A bitnumber was specified as hexadecimal rather than decimal, or the bitnumber was out-of-range for the statement (i.e. in ATOG or DTOG statement, bitnumber $>$ 31 will cause this error, in DTOG @ CTL, bitnumber $>$ 7 will cause this error). Consult the appropriate page in Section 6, Language Reference, for the statement in error to determine the bitnumber limits.

Illegal label number

A hexadecimal label number (a single digit) is out of the range 0-9 or A-F. (For example, FF was used as a label number, or GOTO AB was attempted.)

Illegal option

You have entered an illegal listing option from the interactive mode, or have an illegal listing option in the command line.

Illegal program name

A keyword was used as a program name, a program name of all hexadecimal characters was used, or one of the predefined register names was used as a program name.

EXAMPLES: Program test
Program abcd
Program bitmask

Illegal program number

A program number out of the range 0-99, or a bad program number, such as PROGRAM 44R, was used.

Illegal program order

Numbered programs are not in numerical order. Too many symbolically named programs are between numbered programs. Programs appear in the source file after program 99.

Illegal register number

A hexadecimal register number (a single digit) is out of the range 0-9 or A-F (i.e., REG FF z 100FF).

Illegal signature

More than four hexadecimal digits were used in a signature.

Illegal value

A value is out-of-limits.

EXAMPLES: LINESIZE 300
TIMEOUT 70000

You should check the appropriate page in Section 6, Language Reference, to determine the legal range of values.

INCLUDEs nested too deeply

INCLUDE statements are nested past the maximum depth of five. (Because this is a system dependency, your system may not allow nesting to five.)

Input line too long

Lines longer than the maximum of 255 characters were used.

Invalid forcing line

You probably did not include the appropriate pod data file in the source file.
You may have misspelled the name of the forcing line in an ENABLE statement.
The pod data file may have been modified to contain a FORCELN name more than six characters long.

Missing checksum, should be xxxx

There were no checksums in a binary program, or the checksums were missing the delimiter characters (* or $).

Missing label

A label was used as the target of a GOTO, but was not created (through a LABEL statement). Also, check for misspelling of label names.

Mixed symbolic label names with hex label numbers

Within a single program, all of the labels must be symbolically named or all of the labels must be hexadecimal digits. The two cannot be mixed.

Program not found

A literally-numbered program used as the target of an execute statement (i.e., EXECUTE PROGRAM 96) was not present in the source file(s) that was compiled.

Syntax error

The indicated line contains a statement that is incorrectly formed. It may have a misspelled word, it may be incomplete, it may be missing a keyword, or a keyword or hexadecimal constant may have been used as a symbolic name.

Note that the spelling that the 9010A uses on its display is not strictly compatible with the compiled language.

Example: 9010A Display: SET-TRAP BAD PWR SUPPLY? YES
9LC Syntax: TRAP BAD POWER SUPPLY - YES

Refer to the appropriate syntax diagram to verify correct spelling and syntax.

Too many labels

More than 16 labels were used in one program.

Too many symbolic names

You used more than 100 local symbolic names (register/label names) or more than 200 global symbolic names (program names and register names).

Undefined symbolic name

A symbolic register name was used before that register was declared in an ASSIGN statement or the register name was misspelled or a symbolically-named program used as the target of an Execute statement (i.e., EXECUTE PROGRAM MISSING) was not present in the source file(s) that was compiled.

USAGE: 9lc [-isda] [-h hexfile] [-l [lisfile]] srcfile

You have tried to use the compiler program incorrectly (i.e., an illegal option was specified, you did not put a filename after the -h flag, etc.). The usage line above shows the correct format for using the compiler.

Warning: illegal character

A character outside of the DPY/AUX character set has been used. For example, you have attempted to use parenthesis() or brackets [] in a DPY/AUX string. Valid characters are described on the AUX and DPY pages in Section 6, Language Reference.

Warning: incorrect byte count

The byte count listed on the program statement is incorrect. The program has probably been edited.

Warning: invalid separator character

A character other than a tab, space, or dash was used to separate a DPY/AUX string from the keyword.

Warning: string too long, discarding: xxxx

There are more than 32 characters in the AUX/DPY string. The compiler program will ignore all characters past the first 32.

" expected in INCLUDE statement

Missing quote surrounding the filename to be included.

FILE TRANSFER ERROR MESSAGES

Address descriptors must precede program information

You attempted to save address descriptors after saving programs.

Address descriptors previously saved in this file

You attempted to save address descriptors more than once.

Cannot open ⟨filename⟩

An illegal file name was entered.
You attempted to open a file for writing on a write-protected disk.
You attempted to open a file that does not exist.
You attempted to create a file on a full disk.

Cannot open temporary file

There is not enough room to open a temporary file.

Data transmission error detected

A checksum error was detected, indicating that data transmission errors occurred. This is possibly due to a bad connection between the 9010A and host computer, or the time delay specified by the Setup parameter NEWLINE is not a large enough value. Check the connections and try again, or try a larger time delay value.

Illegal option

An illegal option was used. Enter a valid option.

Illegal program ordering

You attempted to save a program with a number LOWER than a program already saved.

Incorrect data format for transfer

You pressed the wrong keys on the 9010A.
Port setup parameters were set incorrectly.
The 9010A started from a stall.

Check the port parameters and try again.

No address descriptors to save

You attempted to save address descriptors when none were transferred from the 9010A.

No program information to save

You attempted to save programs when none were transferred from the 9010A.

Not a valid port

The port name entered is not valid for the host system. Use a valid port name.

Other information already saved prevents entire file save

You attempted to save an entire file after already saving other information.

Program ⟨program number⟩ already saved

You attempted to save the same program more than once.

Program ⟨program number⟩ not found

You attempted to save a program that was not transferred from the 9010A.

Programs already saved will cause illegal ordering

You attempted to save all programs after some have already been saved.

Setup information must precede program information

You attempted to save setup information after saving programs.

Setup information previously saved in this file

You attempted to save setup information more than once.

DISK VERIFICATION PROGRAM ERROR MESSAGES

The following messages are the result of file configuration errors. If the errors persist after an attempt to recopy the indicated files, contact a Fluke Technical Service Center for advice.

Data file VERIFY.DAT not found

The file VERIFY.DAT does not reside on the system default device. Copy VERIFY.DAT from the original diskette to the system default device.

File ⟨filename⟩ not found

The file filename does not reside on the system default device. If the file is needed, copy it from the original diskette to the system default device.

File ⟨filename⟩ error -- signature is ⟨sig⟩, should be ⟨sig⟩

The indicated file has been corrupted or has been modified. Check that the appropriate Copy command was used (in systems where different commands are used for binary and ASCII data), check for bad blocks on the disk, or verify that the version number for the file is the same as specified in the VERIFY.DAT file.

Illegal or missing signature for file ⟨filename⟩

The VERIFY.DAT file may have been altered. Try using a new copy from the original diskette.

⟨x⟩ files tested -- ⟨i⟩ bad signatures, ⟨i⟩ missing files

Provides a summary of the errors that occurred while running the VERIFY program.

INDEX

9010A Language Compiler Software Error Report Form

We would like to know how the 9010A Language Compiler meets your expectations, and whether you encountered any shortcomings, including missing features you consider important, cases where the program does something unexpected, and bugs of all kinds. This information will help us to improve the product.

We suggest that you retain this sheet as an original and use a photocopy for each report.

Date: _____ Name of User: _____

Co. Name: _____ Dept: _____

Street: _____ City: _____

Mail Stop: _____ Phone No. _____

Model Number (i.e., 9010A-920, etc. from diskette label): _____

Software Version Number (from diskette label): _____

Program Name and Version Number (i.e., XFER ver 1.0, 8080.POD ver. 1.2, etc.): _____

Host Operating System (include Version Number): _____

Host Computer System (i.e., IBM PC): _____

Description of problem: _____

How can problem be reproduced? (Attach listing or separate sheet of paper, if appropriate) _____

Were you able to work around the problem? If so, how? _____

Return completed form to:

John Fluke Mfg. Co., Inc.
Digital Service Products
M/S 267D
9LC Product Manager
P.O. Box C9090
Everett, WA 98206

CHANGE/ERRATA INFORMATION

ISSUE NO: 1 12/84

This change/errata contains information necessary to ensure the accuracy of the following manual. Enter the corrections in the manual if either one of the following conditions exist:

1. The revision letter stamped on the indicated PCB is equal to or higher than that given with each change.
2. No revision letter is indicated at the beginning of the change/errata.

MANUAL

Title: 9010A Language Compiler
Print Date: December 1983
Rev.- Date: ---

C/E PAGE EFFECTIVITY

Page No.	Print Date
1	12/84
2	12/84
3	12/84
4	12/84
5	12/84
6	12/84

ERRATA #1

On page 3-1, under Fluke 1720A Instrument Controller, System Dependencies,
CHANGE: Test Editor
TO: Text Editor

On page 3-6, change the second sentence in Step 3 to read:

SET, a 1720A system program, is included on the 1720A System Disk for this purpose.

On page 3-11, change the second sentence in Step 3 to read:

The Set Utility program (SET), a 1722A system program, is included on the 1722A System Disk for this purpose.

On page 3-12, delete the second paragraph under Text Editor.

On page 3-15, add the NOTE at the end of Step 2:

NOTE

2400 baud is the fastest data transfer rate allowable. If transfer problems occur at 2400 baud, try again at 1200 baud (switch setting 4).

Following step 3, replace both paragraphs with:

You may use the IBM MODE command to configure the serial port. The command line for the IBM PC that is used to implement the suggested setting is:

MODE COM1: 24,E,8,1

for 2400 baud, and

MODE COM1: 12,E,8,1

for 1200 baud.

On page 3-17, between the two paragraphs under Introduction, add the following NOTE:

NOTE

The 9LC program will look for a file on the first operational disk drive that it encounters, and will "hang up" if that disk drive is empty. If, for example, the 9LC disk is in drive 1, and drive 0 is empty, the program will hang up looking for the file on drive 0. The disk activity indicator on drive 0 will be on, and the display will show "9LC".

On page 3-20:
Add the following to the end of the second paragraph:

(The Kaypro version of the program will only prompt for the baud rate.)

Replace the last sentence of the fourth paragraph with:

This file contains the status and data addresses. The Kaypro version also includes SIO initialization bytes.

On page 4-11, add the following to the end of the first paragraph:

Data files may not be available for newer pods. If a data file is not included for the pod you are using, consult the Pod Instruction Manual for information about creating the proper data file. You may add this new data file to the disk and INCLUDE it, or you may insert the information directly into the beginning of the program.

On page 5-7, insert the following note between the last two paragraphs:

NOTE

The total number of bytes required must not be more than the maximum memory size of the 9010A--10,192 bytes. If the "TOTAL = xx bytes" message printed by the Compiler program exceeds 10,192, then you must reduce the size of the source program(s).

On page 5-8, replace the format line, with:

[device]9LC [-listoptions] [-H hexfile] [-L [listfile]] srcfile <RETURN>

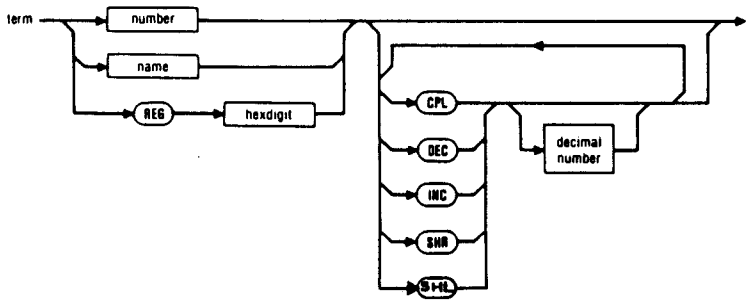
On page 5-11, add the following paragraphs at the bottom of the page:

The CPM version of the software allows you to redirect the reporting of syntax errors to a file, instead of to the display. The Command Line for directing the syntax error report to a file is:

[device]9LC [-listoptions] [-H hexfile] [-L [listfile]] srcfile > msgfile <RETURN>

The "msgfile" specified in the command line will receive all reports of syntax errors that may occur during compiling, and also other status messages that would normally be printed on the display.

On page 6-9, replace the syntax diagram at the bottom of the page, with:



On page 6-69, place the \ character in the left column at the top of the page.

On page D-1, add the following to the list of pods:

Z8000, 8051, 8051X, 8031, 80186, 80188

On page D-3, expand the table to include:

Z8000	0800FFFE	0000	BUSREQ	YES
			WAIT	YES
8051	30000	0000	UNUSED	YES
8051X	20000	0000	UNUSED	YES
8031	10000	0000	UNUSED	YES
80186	0000	FFFF0	HOLD	YES
			EXTRDY	YES
80188	0000	FFFF0	HOLD	YES
			EXTRDY	YES

Addendum

The following supplementary information is provided to clarify or expand material in this manual.

COMMAND-LINE SPACING

Some host computers have specific requirements for mandatory spacing in the command lines. You must pay particular attention to providing the correct specifying syntax. For example, the SET RS-232 Utility for the Fluke 1720A Instrument Controller requires that a space be inserted after "KB1:" and between each parameter.

ASCII TEXT EDITORS

The Text Editor used on the host computer system must produce a source file that contains only standard printable ASCII text characters and no special control or formatting characters.

SET RS-232 UTILITY PROGRAM

Some early versions of the 1720A Set RS-232-C Utility program do not implement the STALL option (versions 1.x). You may upgrade your software by purchasing the 1720A-200U software upgrade package which contains, among other things, the new version of the Set RS-232-C Utility program. Contact a Fluke Technical Service Center for information.

REQUIRED SPACES

In the description of terms to be used in expressions (shown with a syntax diagram on page 6-9), if more than one operator is specified, each operator must be separated from the rest by a space. A space must also be inserted between REG and the following hex register number.

POD DATA FILES

New interface pods are continually being developed by Fluke. If a data file is not on the disk for your pod, you may use the simple procedure shown below to create one (also, refer to the Instruction Manual for you pod):

1. Using the editor, create a new file named <podname>.POD. <podname> is the name of your new pod, such as 80186.

2. Copy the following lines into the file.

```
!
! <podname> Pod data file
!
FORCELN <name> = <n>
busadr = <address>
uutadr = <address>
```

<name> is the name of an enableable forcing line. <n> is the bit in the enable mask that corresponds to that forcing line. Use a

separate line for each enableable forcing line. You must define all of the pod's available forcing lines. The 80186 Pod, for example, has two enableable forcing lines, EXTRDY and HOLD, so you would insert

```
FORCELN extrdy = 0
FORCELN hold = 1
```

<address> is the hex address to be used for either BUS TEST (busadr) or RUN UUT (uutadr). For example

```
busadr = 00000
uutadr = FFFF0
```

To specify the BUS TEST to begin at 00000 and the RUN UUT to begin at FFFF0.

3. Save this new file as file <filename>.POD on the disk.

If you would like to add your new Pod data file to the list of files that are checked by the VERIFY program, do the following steps:

1. Edit file VERIFY.DAT (supplied on the 9LC disk) and add the following line to the end of the file:

   ```
   <filename>.POD     DDDD
   ```

   <filename>.POD is the name of the new Pod data file and DDDD is a dummy checksum for the file. (You'll replace the dummy checksum with a real one later.)

2. Save the modified VERIFY.DAT file on the disk.

3. Run the VERIFY program. The last two messages that it reports should be:

   ```
   File <filename>.POD error - signature is CCCC, should be DDDD

   zz files tested - 1 bad signatures, 0 missing files
   ```

   <filename>.POD is the name of the new Pod data file, CCCC is the correct checksum for the Pod data file, and zz is the number of files tested.

4. Write down the correct checksum for the Pod data file (CCCC).

5. Re-edit the file VERIFY.DAT and replace the dummy checksum that you entered before (DDDD) with the correct checksum (CCCC).

6. Run the VERIFY program again to confirm that all changes have been made satisfactorily. The last two messages that it reports should now be:

   ```
   File <filename>.POD verified
   ```

zz files tested -- no errors

RS-232 INTERFACE CABLE

The cable used to connect the host computer to the Troubleshooter must implement this wiring scheme:

(1) Optional. Use if your host computer requires RLSD (Received Line Signal Detector) to be asserted high.

(2) Optional. Use if your host computer requires DSR (Data Set Ready) to be asserted high.

(3) Optional. Use if your host computer requires RI (Ring Indicator) to be asserted high.