

Series 32000

TDS: Tiny Development Systems
User's Manual



National
Semiconductor
Corporation

Customer Order Number NSP-TDS-M
NSC Publication Number 420306440-001B
December 1984

Series 32000™

TDS™: Tiny Development System
User's Manual

© 1984 National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGES
A	09/83	First Release. TDS16: Tiny Development System User's Manual Publication No. 420306440-001
B	02/84	Updated to include operation information for TDS on a DB16000A or a DB32000 Development Board.
	12/84	Manual title changed to <i>Series 32000 TDS: Tiny Development System User's Manual</i> and related <i>Series 32000</i> name changes where applicable.

PREFACE

This manual describes the Tiny Development System (TDS™) software. TDS provides the user with a way to generate *Series 32000*™ executable code.

TDS supports National Semiconductor Corporation's *Series 32000* family of advanced microprocessors.

APPLICABILITY AND CONFIGURATION

There is a TDS program for the DB16000, for the DB32016, and for the DB32000. This manual describes the generic TDS and notes differences where applicable. These TDS programs work with these boards.

		DB16000	DB32016	DB32000-XXXS	DB32000-XXXD
TDS16	484306440-005 to 008	YES	YES	NO	NO
TDS16A	484007346-001 to 002	NO	YES	NO	NO
TDS32	484407272-013 to 016	NO	NO	YES	NO

Series 32000, TDS, and SERIES/80 are trademarks of the National Semiconductor Corporation.

Series 32000 Documentation

<i>Series 32000</i> Instruction Set Reference Manual	(Pub. No. 420010099-001)
<i>Series 32000</i> NSX Cross-Support Utilities Reference Manual	(Pub. No. 420306617-002)
<i>Series 32000</i> Pascal Language and Compiler Reference Manual	(Pub. No. 420306618-002)
<i>Series 32000</i> Cross-Assembler Reference Manual	(Pub. No. 420306619-002)
<i>Series 32000</i> ISE16: NS32016 and NS32008 In-System Emulators User's Manual	(Pub. No. 420306675-002)
<i>Series 32000</i> Symbolic Debugger Reference Manual	(Pub. No. 420306676-002)
<i>Series 32000</i> Run-Time Support Library Reference Manual	(Pub. No. 420308038-002)
<i>Series 32000</i> Floating-Point Support Library Reference Manual	(Pub. No. 420308220-002)
<i>Series 32000</i> Development Board Monitor Reference Manual	(Pub. No. 420308221-002)
<i>Series 32000</i> NSX Operations Manual	(Pub. No. 424009011-002)
<i>Series 32000</i> DB32000 Development Board User's Manual	(Pub. No. 420010144-001)
<i>Series 32000</i> TDS: Tiny Development System	(Pub. No. 420306440-001)
<i>Series 32000</i> DB32016 Development Board User's Manual	(Pub. No. 420310111-001)
<i>Series 32000</i> EXEC: ROMable Real-Time Multitasking EXECUTIVE Reference Manual	(Pub. No. 420010206-001)
<i>Series 32000</i> GENIX Cross-Support Software Programmer's Manual	
Volume 1	(Pub. No. 424010106-001)
Volume 2	(Pub. No. 424010106-002)
<i>Series 32000</i> GENIX Programmer's Manual	
Volume 1	(Pub. No. 424308225-001)
Volume 2	(Pub. No. 424308225-002)
<i>Series 32000</i> GENIX Debugging Reference Manuals	
GENIX ISE16: NS32008 and NS32016 In-System Emulators	(Pub. No. 420308165-001)
GENIX Symbolic Debugger	(Pub. No. 424010149-001)
Development Board Monitor Reference Manual	(Pub. No. 420308221-002)
<i>Series 32000</i> SYS32 System Manual	(Pub. No. 420308225-001)
<i>Series 32000</i> SYS32 Site Installation Guide	(Pub. No. 409308225-001)
<i>Series 32000</i> SYS32 Add-On Disk/Disk Module Installation Guide	(Pub. No. 409308226-001)

The information contained in this manual is for reference only and is subject to change without notice.

No part of this document may be reproduced in any form or by any means without the prior written consent of National Semiconductor Corporation.

CONTENTS

Chapter 1	INTRODUCTION	1-1
1.1	PRODUCT DESCRIPTION	1-1
1.2	FUNCTIONAL OVERVIEW	1-1
1.2.1	Interactive TDS Software Environment	1-1
1.2.2	Peripheral Environment and Run-Time Support	1-2
1.3	RELATED DOCUMENTATION	1-2
1.4	MANUAL ORGANIZATION	1-3
Chapter 2	GENERAL SYSTEM CONVENTIONS	2-1
2.1	INTRODUCTION	2-1
2.2	DOCUMENTATION CONVENTIONS	2-1
Chapter 3	SYSTEM INITIALIZATION	3-1
3.1	INTRODUCTION	3-1
3.2	INITIALIZATION PROCEDURE	3-1
3.3	TERMINAL COMMUNICATIONS AND SYSTEM USE COMMANDS	3-1
3.3.1	RX Command	3-1
3.3.2	OM Command	3-2
Chapter 4	TEXT EDITOR	4-1
4.1	INTRODUCTION	4-1
4.2	TEXT EDITOR COMMANDS	4-1
4.2.1	Insert Command	4-1
4.2.2	Type Command	4-2
4.2.3	Replace Command	4-3
4.2.4	Kill Command	4-3
4.2.5	Reset Command	4-3
4.3	EDITOR ERROR MESSAGES	4-3
Chapter 5	ASSEMBLER	5-1
5.1	INTRODUCTION	5-1
5.2	VALID ASSEMBLER STRINGS	5-1
5.3	ASSEMBLER COMMANDS	5-2
5.3.1	Redirect Listing	5-3
5.3.2	Assemble	5-3
5.4	ASSEMBLER ERROR MESSAGES	5-4
5.4.1	Pass1 Error Messages	5-4
5.4.2	Pass2 Error Messages	5-4
Chapter 6	SYMBOLIC DEBUGGER	6-1

6.1	INTRODUCTION	6-1
6.2	PROGRAM INITIALIZATION	6-1
6.2.1	Manual Program Initialization	6-2
6.2.2	Automatic Program Initialization	6-3
6.3	DEBUGGER PRINT COMMANDS	6-3
6.4	DEBUGGER CHANGE COMMANDS	6-4
6.5	DEBUGGER EXECUTION COMMANDS	6-5
6.5.1	STEP N INSTRUCTIONS Command	6-5
6.5.2	STEP UNTIL Command	6-5
6.5.3	STEP WHILE Command	6-5
6.5.4	JUMP SUBROUTINE Command	6-6
6.5.5	CALL EXTERNAL SUBROUTINE Command	6-6
6.5.6	GO Command	6-6
6.6	SPECIAL PROCESSING COMMANDS	6-6
6.6.1	MOVE BLOCK OF DATA Command	6-7
6.6.2	FILL MEMORY WITH DATA Command	6-7
6.6.3	SEARCH FOR DATA Command	6-7
6.6.4	HEX MEMORY DUMP Command	6-7
6.7	DEBUGGER USER NOTES	6-7
6.8	DEBUGGER ERROR MESSAGES	6-8
Chapter 7	RUN-TIME SUPPORT	7-1
7.1	INTRODUCTION	7-1
7.2	SERIAL COMMUNICATIONS	7-1
7.2.1	Read and Buffer Serial Line Data	7-1
7.2.2	Write Buffered Data to Serial Line	7-2
7.3	NUMERICAL CONVERSIONS	7-2
7.3.1	ASCII to Binary Conversion	7-2
7.3.2	Binary to ASCII Conversion	7-3
7.4	PARALLEL PRINTER COMMUNICATIONS	7-3
Chapter 8	PROGRAM EXECUTION AND SAMPLE DEVELOPMENT SESSION	8-1
8.1	INTRODUCTION	8-1
8.2	TEXT EDITING	8-1
8.3	ASSEMBLY	8-3
8.4	DEBUG DATA	8-4
8.5	PROGRAM INITIALIZATION AND EXECUTION	8-5
8.6	RUN-TIME SUPPORT EXAMPLES	8-6
8.7	SLAVE SUPPORT	8-8
8.7.1	FPU Support	8-8
8.7.2	MMU Support	8-8
Chapter 9	PERIPHERAL INTERFACING	9-1

9.1	INTRODUCTION	9-1
9.2	CENTRONICS-TYPE PARALLEL PRINTER	9-1
9.3	SERIAL COMMUNICATIONS	9-1
9.4	CASSETTE RECORDER	9-1
	9.4.1 Data Storage and Retrieval	9-2
	9.4.2 Cassette Interface Cable	9-2
	9.4.3 Recorder Use	9-2
	9.4.4 Recorder Error Messages	9-3
9.5	NEW COMMANDS FOR TDS ON THE DB32000 AND DB32016	9-3
	9.5.1 Read Data In From Port	9-4
	9.5.2 Write Data Out To Port	9-4
	9.5.3 The AT R Command	9-5
	9.5.4 The AT M Command	9-5
	9.5.5 Sample Program	9-6
Appendix A	TDS ON THE DB16000	A-1
	A.1 AUXILIARY PORT BLX-351 SETUP PROCEDURES (DB16000)	A-1
	A.2 BAUD RATE SETUPS (DB16000)	A-2
	A.3 NUMERICAL DATA (DB16000)	A-2
	A.4 PARALLEL PRINTER INTERFACE DATA (DB16000)	A-3
	A.5 CASSETTE INTERFACE DATA (DB16000)	A-4
	A.6 ASSEMBLER USER NOTES	A-6
	A.7 TDS MEMORY MAP (DB16000)	A-7
	A.8 TDS ERROR SUMMARY (DB16000)	A-8
	A.9 TDS COMMAND SUMMARY (DB16000)	A-10
Appendix B	TDS ON THE DB32016	B-1
	B.1 AUXILIARY PORT SETUP PROCEDURES (DB32016)	B-1
	B.2 BAUD RATE SETUPS (DB32016)	B-1
	B.3 NUMERICAL DATA (DB32016)	B-2
	B.4 PARALLEL PRINTER INTERFACE DATA (DB32016)	B-3
	B.5 CASSETTE INTERFACE DATA (DB32016)	B-4
	B.6 ASSEMBLER USER NOTES (DB32016)	B-6
	B.7 TDS MEMORY MAP (DB32016)	B-7
	B.8 TDS ERROR SUMMARY (DB32016)	B-8
	B.9 TDS COMMAND SUMMARY (DB32016)	B-9

Appendix C	TDS ON THE DB32000	C-1
C.1	AUXILIARY PORT SETUP PROCEDURES (DB32000)	C-1
C.2	BAUD RATE SETUPS (DB32000)	C-1
C.3	NUMERICAL DATA (DB32000)	C-2
C.4	PARALLEL PRINTER INTERFACE DATA (DB32000)	C-2
C.5	CASSETTE INTERFACE DATA (DB32000)	C-3
C.6	ASSEMBLER USER NOTES (DB32000)	C-6
C.7	TDS MEMORY MAP (DB32000)	C-6
C.8	TDS ERROR SUMMARY (DB32000)	C-7
C.9	TDS COMMAND SUMMARY (DB32000)	C-9

Chapter 1

INTRODUCTION

1.1 PRODUCT DESCRIPTION

The TDS (Tiny Development System) is a stand-alone board-level software support system for the *Series 32000* family of advanced microprocessors. The TDS software provides the means for the user to generate *Series 32000* executable code.

TDS utilities enable the user to originate, edit, assemble, debug, and execute *Series 32000* small source programs. These also support the use of an optional cassette tape recorder for the storage and retrieval of source programs, and the use of a parallel printer for the generation of program listings. TDS is intended as a demonstration tool for *Series 32000* development boards. TDS utilities cannot substitute for a full development system and are not designed to be integrated into a customer product.

In addition to a text editor, an assembler, and a symbolic debugger, TDS includes run-time support routines that manage terminal I/O access to peripheral devices, and complete numerical conversions requestable by user programs.

On the DB16000 board TDS PROMs plug into sockets U9, U10, U11 and U12. On the DB32016 board TDS plugs into U13 and U15. On the DB32000 board TDS plugs into U163, U145, U125, and U111.

1.2 FUNCTIONAL OVERVIEW

1.2.1 Interactive TDS Software Environment

The TDS software system resides in PROMS on currently available *Series 32000* development boards. The appendices contain information specific to the particular development board used. TDS facilities are available to the user once the development board is running in a stand-alone mode. Refer to the individual development board manual for information necessary to establish the required stand-alone environment.

The available TDS utilities are:

- A Text Editor
- An Assembler
- A Symbolic Debugger

Text Editor

Enables the user to create and maintain source programs. Text editor commands support the creation, deletion, insertion, replacement, and display of user-originated programs. Source lines created or edited are automatically sequenced for easy use and reference.

Assembler

Assembles user programs and produces machine executable code. Assembler error messages alert the user to errors detected during the assembly of a source program.

Symbolic Debugger

Provides program debug aids that allow the user to step through an assembled program, follow the program flow at source level by setting up breakpoints symbolically if the source line is labeled, symbolically print or change the contents of memory and system registers, search system memory for specific data, move blocks of data from one location in system memory to another, and fill contiguous locations of system memory with data.

1.2.2 Peripheral Environment and Run-Time Support

TDS also provides the following for the user:

- Peripheral Interfacing
- Run-Time Support

TDS supports two RS232-C ports and one parallel port. One of the RS232-C ports is designated as port 0 and the other port is designated as port 1. Port 0 is the main system port where the user terminal is connected, and port 1 is the system auxiliary port. Port 1 can accommodate any serial peripheral device. The parallel port is used to implement the audio cassette interface for source program/data storage and retrieval. It can also support any Centronics-type parallel printer to generate source listings or record data outputs.

Run-time support services provide the user with the ability to access ports 0 and 1, and the parallel printer port. Run-time support services also permit the user to perform ASCII to binary and binary to ASCII numerical conversions on four user-selected bases.

1.3 RELATED DOCUMENTATION

If TDS is to be used to develop source programs for *Series 32000* microprocessors, the user must understand the architecture of the *Series 32000* family of microprocessors and their programming requirements. See the *Series 32000 Instruction Set Reference Manual*, Customer Order Number NSP-INST-REF-M, for additional information.

1.4 MANUAL ORGANIZATION

This manual is divided into nine chapters and appendices A through C. Each appendix contains nine sections. Appendix A references DB16000; Appendix B references DB32016; and Appendix C references DB32000. The manual is organized in the following order:

- Chapter 2 – General System Conventions
- Chapter 3 – System Initialization
- Chapter 4 – Text Editor
- Chapter 5 – Assembler
- Chapter 6 – Symbolic Debugger
- Chapter 7 – Run-Time Support
- Chapter 8 – Program Execution and Sample Development Session
- Chapter 9 – Peripheral Interfacing
- Appendix A – TDS on the DB16000
- Appendix B – TDS on the DB32016
- Appendix C – TDS on the DB32000

Each appendix is divided into nine sections as listed below:

- Sections A.1, B.1, C.1 – Auxiliary Port Setup Procedures
- Sections A.2, B.2, C.2 – Baud Rate Setups
- Sections A.3, B.3, C.3 – Numerical Data
- Sections A.4, B.4, C.4 – Parallel Printer Interface Data
- Sections A.5, B.5, C.5 – Cassette Interface Data
- Sections A.6, B.6, C.6 – Assembler User Notes
- Sections A.7, B.7, C.7 – TDS Memory Map
- Sections A.8, B.8, C.8 – TDS Error Summary
- Sections A.9, B.9, C.9 – TDS Command Summary

Chapter 2 introduces the user TDS procedural fundamentals and defines its documentation conventions.

Chapter 3 provides information on initializing TDS, selecting communication mode for the user terminal, and selecting the system default base for the data output.

Chapter 4 lists and defines the commands, error messages, and procedures made available by the text editor.

Chapter 5 lists and defines the commands and the assembly syntax accepted by the assembler, and the error messages output by the assembler.

Chapter 6 lists and defines the symbolic debugger printer commands, change commands, special processing commands, and execution commands. The chapter also lists and defines error messages output by the debugger, and the method of program initialization.

Chapter 7 explains RTS use and lists the conventions of the routines used to support peripheral access and numerical conversions.

Chapter 8 presents a sample development session where source code is entered for sample programs and assembled. Debugging tools are demonstrated and the program is executed. The chapter also provides information concerning slave support.

Chapter 9 provides information on the requirements and use of the peripheral devices supported by TDS.

A complete set of appendices for the various development boards provides information on hardware set-ups, ranges and syntax of numerical values, parallel printer interface cable pin-outs, and cassette tape interface cable construction details and interface timing data. The appendices also provide memory maps of the TDS system, a complete summary of all TDS error messages, and a summary of all commands (listed in functional sequence) supported by TDS.

Chapter 2

GENERAL SYSTEM CONVENTIONS

2.1 INTRODUCTION

The user interface to TDS is the command line interpretive (CLI) mode. The CLI mode is the user's link to the facilities provided by TDS.

The CLI mode is the main mode of control for TDS and is characterized by an asterisk (*) at the left edge of the user's terminal screen, marking the beginning of a line. The asterisk indicates that a user may enter a command. The CLI mode listens to user commands and responds with desired results or an error message. Errors in commands or syntax will cause TDS to respond with a question mark (?). Whenever the "?" is displayed by TDS, the command line originally entered is ignored. After the question mark is displayed, TDS issues a line feed if the communication option is set (see Section 3.3.2, OM Command) TDS also displays the "*" once again. Consult Chapter 3 for the proper setting of the communications mode for the terminal.

Commands consist of one or two letters (upper or lower case) and are entered immediately after the "*". Command parameters are separated by a single space. Commands are executed when a <CR> carriage return is entered. Any exceptions to this convention are covered during individual command descriptions.

Tabs, line feeds, or special characters are ignored.

The backspace or delete key may be used to delete characters. A deleted character is replaced by a space and the cursor is positioned over the deleted character. Backspacing or deleting to the beginning of the line will null that line and TDS will display the "*" prompt.

TDS uses the control "Q" and "S" keys to control scrolling functions. The <ctrl/S> is used to stop terminal output, and the <ctrl/Q> continues the terminal output. The scrolling functions are supported only at the main port (port 0).

The maximum character length of a valid command line is 62. If this maximum is exceeded, TDS outputs the "?" and the command line input is ignored.

2.2 DOCUMENTATION CONVENTIONS

The following documentation conventions are used in describing commands and parameters. Except for mnemonics any combination of upper- and lower-case letters may actually be used when entering commands.

Upper-case letters show the instruction and directive mnemonics. The mnemonics must be entered exactly as shown.

Italics are used for items supplied by the user. The italicized word is a generic term for the actual operand that the user enters.

Spaces or blanks, when present, are significant; they must be entered as shown. Multiple blanks or horizontal tabs may be used in place of a single blank.

- { } Large braces enclose two or more items of which one, and only one, must be used. The items are separated from each other by a logical OR sign “|”.
- [] Large brackets enclose optional item(s).
- | Logical OR sign separates items of which one, and only one, may be used.
- ... Three consecutive periods indicate optional repetition of the preceding item(s). If a group of items can be repeated, the group is enclosed in large parentheses “()”.
- ... Three consecutive commas indicate optional repetition of the preceding item. Items must be separated by commas. If a group of items can be repeated, the group is enclosed in large parentheses “()”.
- () Large parentheses enclose items which need to be grouped together for optional repetition. If three consecutive commas or periods follow an item, only that item may be repeated. The parentheses indicate that the group may be repeated.
- Indicates a space. □ is only used to indicate a specific number of required spaces.

User inputs of nonprinting keys are indicated by enclosing a name for the key in angle brackets <> (<CR> indicates the RETURN key).

All other characters or symbols appearing in the syntax must be entered as shown. Brackets, parentheses, or braces which must be entered, are smaller than the symbols used to describe the syntax. (Compare user-entered [], with [] which show optional items.)

In interactive examples where both user input and system responses are shown, the machine output is in regular type. User-entered input is in boldface type.

Parameters:

Number (*n*) *digits*
Where digits equal one or more decimal digits (0 through 9).

General-Number {D'|H'|d'|h'} [-] *digits*
(*gn*) Where digits are compatible with selected base. Hex value digits are 0 through 9 and A through F (upper or lower case). (-) specifies negative value. Default base numbers require only *digits* for input. *gn* is zero-extended or truncated to byte, word, or double-word according to any size restriction common to a particular operation. See Appendices A.3, B.3, and C.3 for additional numerical information.

Real-Number (<i>rn</i>)	$n [. [n]] [e E [+ -] n]$ When used in symbolic debugger operations, the [E] must be upper case. See Appendices A.3, B.3, and C.3 for further information.
General Address (<i>addr</i>) CPU register (<i>cpureg</i>)	gn Where $gn \geq 0$ {PC SB FP US IS SP IN PS MOD}

Chapter 3

SYSTEM INITIALIZATION

3.1 INTRODUCTION

The TDS user must enter the required initialization command and optionally set terminal communication mode and system base for numerical interpretations. Sections 3.2 through 3.3.2 detail this information.

3.2 INITIALIZATION PROCEDURE

Once the development board is powered up and reset in stand-alone mode, TDS displays the following message:

```
*TDS Rev N.NN DAY-MON-YR (BOARD VERSION)
```

The “*” should appear on the left-most side of the terminal screen. The asterisk indicates that TDS is in the CLI mode and is prepared to accept a command line input.

Enter **IT** (for initialize) at the terminal and press <CR>.

The **IT** command initializes functions not performed by a hardware reset. If the **IT** command is not entered before the user attempts to use TDS facilities, TDS will not function properly.

3.3 TERMINAL COMMUNICATIONS AND SYSTEM USE COMMANDS

Two commands are supported by TDS that enable the user to set up specific conditions within the TDS system. One command sets the default base for symbolic debugger functions, and the other command sets user communication modes at the main user port (port 0). Those two commands respectively are: **RX** and **OM**. The following paragraphs present the syntax of each command and discuss their use. If the user does not use the **RX** and **OM** commands to make the applicable selections, TDS will use default values.

3.3.1 RX Command

The select radix (**RX**) command sets the default base (hex or decimal) for the symbolic debugger command parameters and causes all command responses to be output in the selected base. The syntax of the command is:

```
RX {H|D}
```

If a command argument is different from the selected base, it must be preceded by **H**, **h**, **D**, or **d**.

For example, if a user wishes to use the debugger facilities to display the contents of system memory at decimal location 100, and the selected base is hex, the command line input must specify **D'100** as the address. The output is in hex notation. If the user wants to examine hex

location 100, it is not necessary to use the H'100 notation in the command line since the selected base is already hex. The IT command initially sets the radix to hex.

The commands and facilities of the symbolic debugger are described in Chapter 6.

3.3.2 OM Command

The operation mode (OM) command sets the communications mode of the user terminal connected to the main port (port 0) of the development board.

The syntax of the command is:

OM *gn*

Valid values for *gn* in this case are: 0 through 4. OM is initially set to 0 by the hardware reset.

TDS uses full duplex communications with the user terminal. All inputs to TDS are echoed back to the user terminal. The OM command controls how carriage returns are treated during echo and how carriage returns are handled during output responses to commands. The values for *gn* have the following meaning:

<i>Gn</i>	ACTION
0	Echo <CR> as <CR>+ <LF>
1	Echo <CR> as <CR>
2	Output <CR> as <CR>+ <LF>
3	Echo <CR> as <CR> and output <CR> only
4	Echo <CR> as space

Chapter 4

TEXT EDITOR

4.1 INTRODUCTION

The system text editor is used to perform the following tasks:

- Create source programs for *Series 32000* microprocessors.
- Change source text.
- Display multiple lines of source text.
- Insert new lines of source text.
- Delete lines of source text.

Section 4.2 defines the text editor commands that initiate and complete user tasks. Section 4.3 lists and defines the error messages output by the text editor.

4.2 TEXT EDITOR COMMANDS

TDS provides a line numbering sequence that can automatically number or renumber lines. Each command supporting the text editor tracks the line numbering sequence. Switching between commands requires that the user specify desired line numbers.

The commands supporting the text editor functions are: INSERT, TYPE, REPLACE, KILL, and RESET. Sections 4.2.1 through 4.2.5 define the syntax of each command and their respective uses.

4.2.1 Insert Command

The Insert command is used to create source text if the editor text buffer is empty, or to insert one or more new text lines in text already contained in the editor buffer, or to append one or more new text lines to text contained in the editor buffer.

The syntax of the Insert command is:

IN [*n*] *string*

Where *n* is the line number and *string* is a valid string that conforms to one of the conventions outlined in Section 5.2.

If the editor text buffer is empty, the automatic sequencing feature of the editor allows the user to begin creating source text by entering:

IN *string* <CR>

The first line of text entered becomes line 0000 in the buffer. Entering the same command line again, with perhaps a different *string*, results in the second line becoming line 0001 of text in the buffer. The sequence continues until the user has entered the desired number of text lines.

To insert new text lines into existing text, the user chooses the existing text line above which the new text line is to be inserted.

4.2.2 Type Command

The Type command is used to list or display a specific line or multiple lines of text contained in the text buffer.

The syntax of the command is:

```
TP [n1 [/i>n2]]
```

Where *n1* is the first line to be typed and *n2* is the number of lines to be typed.

To list text line 0003 from the text buffer, enter:

```
TP 3 <CR>
```

When the Type command is used without specifying *n1*, the value of *n1* follows the value of *n* set by the Insert or Replace command.

For example, if the Insert command is used to insert a new line of text, and the Type command is used immediately afterwards without specifying an *n1* or *n2* value, the line previously inserted is displayed. If an *n2* value is specified and an *n1* value is not, the desired number of lines are displayed beginning with the line previously inserted.

The Type command displays a maximum of 127 lines.

For example, a user wishing to insert a line at line 2, would enter:

```
IN 2 TEXT
```

To type three lines of text beginning with the line just inserted, enter:

```
TP /3
```

However, entering

```
IN 2 TEXT
TP 7/2
IN NEWTEXT
```

may produce incorrect results. If the user inserts a line at line 2 and then types out 2 lines beginning at line 7, the user cannot use simply the IN command to begin inserting text after the last line of text inserted. The TP command changed the default value of *n* to 7. Using the IN command without respecifying *n* would insert a line or lines beginning at line 7 of text.

4.2.3 Replace Command

The Replace command is used to replace a line of text in the buffer with a new line of text.

The syntax of the command is:

RP *n string*

When the RP command is used, *n* must be specified.

4.2.4 Kill Command

The Kill command is used to delete a line of text from the text buffer.

The syntax of the Kill command is:

KI *n*

When the KI command is used, *n* must be specified.

4.2.5 Reset Command

The Reset command clears the text editor buffer.

The syntax of the command is:

RS

Whenever the Reset command is invoked, the text buffer is cleared and is immediately available for the input of new text.

4.3 EDITOR ERROR MESSAGES

The text editor can issue two error messages:

BAD_TXT

ERR_ED#

The bad text (BAD_TXT) error message is output by the editor if an incorrect text sequence is entered. The error message may also occur if a runaway program altered text buffer or editor parameters. In such an event, the RESET command should be entered.

The error in edit number (ERR_ED#) error message is output by the editor if the user specifies an inappropriate or bad number value in a command line. The error message may occur if the user did not respecify a default value for *n* for a new operation after using *n* to complete a previous operation. The ERR_ED# error message may also occur if the user attempts to use the text editor before the IT command is invoked.

Chapter 5

ASSEMBLER

5.1 INTRODUCTION

The principal features of the assembler are as follows:

- The syntax is a compatible subset of existing *Series 32000* assemblers.
- It accepts special syntax for register list and string instruction options.
- It accepts symbols for PC displacements.
- It produces optimized code for displacements entered symbolically.
- It supports symbolic definition of global variables.
- Input may be upper or lower case.
- Program listing output to user terminal (default) includes line number, PC value, source text, and generated code.
- Listing available for hardcopy through Parallel port or Auxiliary RS232 port.
- Floating-point support provided through .FLOAT and .LONG psuedo-ops.
- Assembler can perform simple symbol operation necessary to create displacement table needed by CASE instruction.
- All *Series 32000* family addressing modes accepted.
- Assembler accepts values in decimal (default) or hex.
- Error messages for incorrect instructions, psuedo-ops, values, and duplicated or undefined symbols.

Section 5.2 details input strings accepted by the assembler. Section 5.3 describes assembler commands available to the user, and Section 5.4 discusses error messages output by the assembler.

5.2 VALID ASSEMBLER STRINGS

Valid assembler input strings are characterized as follows:

Symbol { *letter* [*letter*]: | *letter* [*digit*]: }

Symbols are depicted by two characters only, upper or lower case.
No special characters allowed.

Comment ; *alpha-letters-and/or-digits*

May be alpha letters and/or digits, or empty.

Instruction [*symbol*] *instruction* [*comment*]

An instruction is any valid *Series 32000* mnemonic instruction with required operands.

Pseudo-Op [*symbol*] *Pseudo-op* [*comment*]

Valid Pseudo-ops are:

.BYTE {*gn* | *symbol1-symbol2* | *alpha-letters/digits*}

.WORD {*gn* | *symbol1-symbol2*}

.DOUBLE {*gn* | *symbol1-symbol2*}

.FLOAT *rn*

.LONG *rn*

.BLKB {*gn*}

.BLKW {*gn*}

.BLKD {*gn*}

.STATIC

.ENDSEG

A string to TDS is any of the above. It can be a symbol, an instruction, a pseudo-op or a comment.

TDS symbols may be either PC-relative by labelling a code or data generation line, or may be static-base-relative by using a data allocation label. These symbols may be used only as arguments for PC or SB relative address modes. A simple subtraction operation is allowed for PC symbols in order to build a case displacement table. For data generation pseudo-ops, *gn* must be entered. For data allocation pseudo-ops, default *gn* = 1. The .BYTE pseudo-op may be used to create a string enclosed by double quote marks.

Static base area must be defined as the first element of the program if the SB area is used. It must start with .STATIC and terminate with .ENDSEG.

Gn in source is assumed to be decimal. For hex values, specify H' or h'. Consult Section A.6, B.6, and C.6 for user notes.

5.3 ASSEMBLER COMMANDS

Two commands directly associated with assembler functions are Redirect Listing (RL) and Assemble (AS). Sections 5.3.1 and 5.3.2 list the syntax of both commands and define their uses.

5.3.1 Redirect Listing

Program listings are transmitted to the user terminal. The redirect listing (RL) command permits the user to direct listing generation to the parallel printer port or port 1 (the auxiliary serial port).

The syntax of the RL command is as follows:

```
RL [LPT:|ASN:] [C]
```

where:	LPT:	Directs the assembled listing of a source program to a Centronics-type parallel printer connected to the parallel port.
	ASN:	Directs the listing to port 1 (auxiliary port).
	C	Ends listing line S with a <CR>. Default is <CR> <LF>.

No option cancels previously selected device.

5.3.2 Assemble

The syntax of the AS command is as follows:

```
AS [addr]
```

where:	<i>addr</i>	Begins code generation at specified hex address. Decimal values must be preceded by d' or D'. If <i>addr</i> is not specified, the assembler uses the next available page boundary <i>addr</i> . If the <i>addr</i> specified by the user conflicts with system or text data, or the debug table, the assembler will output the BAD_MEM error message.
--------	-------------	--

Description: The actual assembly of source text begins when the user enters:

```
AS [addr] <CR>
```

The source program is assembled in two passes: Pass1 and Pass2. During Pass1 each line of source is displayed on the user terminal along with any assembler error messages. If errors are detected during Pass1, the assembly is terminated and lists the number of errors encountered. If Pass1 is error free, the assembler starts Pass2. During Pass2, the line number, the start address of that line, source text, and generated code are displayed (on the user terminal or a device specified by the RL command) along with any further error messages.

It should be noted that once a source program has been successfully assembled, any changes in source text (using the text editor), invalidates PC and generated code values and debug data originally created by the assembler. This may cause certain debug command errors. To ensure correct operation, always reassemble after changing the source.

5.4 ASSEMBLER ERROR MESSAGES

5.4.1 Pass1 Error Messages

BAD_MEM	Code start <i>addr</i> conflicts with text or debug data. Enter a higher value for <i>addr</i> .
BAD_SEQ	Attempting to assemble null or bad text.
BAD_INS	No such instruction.
BAD_PSU	No such pseudo-op.
ERR_VAL	Bad number syntax or bad operand.
BAD_LIN	Gross line error or stand-alone label.
ERR_SEG	Using data allocation pseudo-ops within PC segment or vice versa.

5.4.2 Pass2 Error Messages

BAD_NUM	Bad floating-point syntax or value range.
BAD_SYM	Duplicate symbol.
UND_SYM	Undefined symbol.
BAD_TAB	Assembler was not initialized by IT command.

Chapter 6

SYMBOLIC DEBUGGER

6.1 INTRODUCTION

The symbolic debugger provides the following features:

- The ability to print or change memory locations, internal general purpose registers (GPR), internal special-purpose registers (SPR), and memory management unit (MMU) and floating-point unit (FPU) registers.
- The ability to print or change command parameters in either hex or decimal base as selected by the user.
- Program stepping. Single or multiple steps.
- Stepping as a function of variables or register values.
- The ability to set breakpoints.
- Commands to dump, search and fill memory.
- ASCII equivalent values printed along with hex memory dump.
- The ability to create symbolic breakpoints using source defined labels.
- Command to print the address of a symbol.
- The ability to make symbolic changes.
- The ability to print global memory or PC segment.
- The ability to print register sets.
- Error messages for unknown SVCs, external aborts, illegal user instructions, undefined op-codes, and memory verifications.

Sections 6.2 through 6.8 present the commands used to implement these features and detail the procedures required to initialize a successfully assembled source program. Note that a carriage return (<CR>) terminates all debugger commands.

6.2 PROGRAM INITIALIZATION

After a source program is successfully assembled, the program must be initialized before the user can actually run it or investigate its logic flow using the symbolic debugger. Program initialization is done manually or automatically.

The manual initialization of an assembled program is used if an address was specified at assembly time and involves two basic steps. The first step encompasses the creation of a module table, and the second step encompasses the setting up of specific system registers. If the program is initialized manually, the module table should be created once for the life of the program while specific system registers must be set up prior to each execution of the

program. Section 6.2.1 outlines both procedures.

Automatic initialization and manual initialization of a program perform the same function.

Once the user program is properly initialized, it may be logically analyzed using the symbolic debugger or run.

Sections 6.2.1 and 6.2.2 define both methods of program initialization. See Sections A.7, B.7 and C.7 for an illustration of the TDS memory space.

6.2.1 Manual Program Initialization

A module table is created using the change memory commands. The module table address must not corrupt source text. No system warning is given if the source text is corrupted by the module table inputs. Experiment with the bad memory (BAD_MEM) error message given at assembly time to determine valid working RAM space. The change memory commands are described in Section 6.4.

A sample command sequence needed to construct a module table is as follows:

```
CMD D000 = D100
CMD D004 = 0
CMD D008 = E000
CMD D00C = 0
```

The first entry is the static base start address. The second entry, the link table start address, is set to zero since the link table is not used in this case. The third entry is the program code start address, and the final entry is reserved and must be set to zero. It is not necessary to repeat this sequence of command unless the module table is corrupted or destroyed.

The following sample sequence of commands initializes system registers and must be entered in the sequence shown before each execution of the program.

```
CPS = 300
CSP = FFF0
CFP = 0
B D000 0
```

The first command loads the value of the PSR required for a user program. The PSR is set to user mode. The second command loads the user stack start while the third initializes the frame pointer. The final command loads the user program *mod* register value and sets the user program static base register and PC. The set values become valid and in use upon entry into the user program by any of the program execution commands. The interrupt stack (IS) value must be valid if the user program will require any run-time support services.

6.2.2 Automatic Program Initialization

The Begin command is used to initialize a user program if an explicit *addr* was not entered when the program was assembled. The syntax of the command is:

B [Z|*mod offset*]

The Begin command loads the MOD register with *mod* and PC with code-start address plus *offset*. ITG also updates the SB register with a value from the module table (see Section 6.2.1 for SB value). If an explicit assembly code start was not used, then *mod* and *offset* may be omitted and a module table is created beginning at PC minus H'100. The SB area will be on the first page boundary after the code. Link is set to zero. SP is set to high memory and IS is set to the IS area. FP is set to zero and PSR is set to 300. If the Z option is selected, memory from SB to SP is zeroed.

To initiate the automatic initialization of a program, enter:

B Z

6.3 DEBUGGER PRINT COMMANDS

The debugger print commands are as follows:

PM {B W D F L} { <i>addr symbol</i> }	Print memory contents of <i>addr</i> or <i>symbol</i> according to selected base. Byte, Word, Double-word, Floating-point, or Long.
PAD <i>symbol</i>	Print address of symbol.
PR {0 1 2 3 4 5 6 7}	Print the contents of one of eight general purpose registers.
Pcpureg	Print the contents of the CPU register.
PMS	Print the contents of the MMU MSR register.
PPT {0 1}	Print the contents of the MMU PTB0 or PTB1 register.
PEI	Print the contents of the MMU EADDR/INVAL register.
PPF {0 1}	Print the contents of the MMU PF0 or PF1 register.
PSC	Print the contents of the MMU SC register.
PBP {0 ... [15 f]}	Print the contents of one of 16 breakpoint registers.
PBC	Print the contents of the MMU BCNT register.
PF {0 1 2 3 4 5 6 7}	Print the contents of one of eight FPU registers.

PFS	Print the contents of the FPU status register.
PCF	Print the contents of the configuration register.
PMM	Print the contents of the main MSR register.
AR	Print the contents of all general purpose registers.
AC	Print the contents of all CPU registers.
AM	Print the contents of all MMU registers.
AF	Print the contents of all FPU registers.

The output format of the printed data is displayed according to the base selected by the user. See Section 3.3.1 for more information concerning base selection.

6.4 DEBUGGER CHANGE COMMANDS

The debugger change commands change the contents of the corresponding user registers unless otherwise specified. The debugger change commands are as follows:

CM {B W D} { <i>addr</i> <i>symbol</i> = <i>gn</i> }	Change memory contents, at <i>addr</i> or <i>symbol</i> to an integer value.
CM {F L} { <i>addr</i> <i>symbol</i> = <i>rn</i> }	Change contents, at specified location, to a real number.
Cpureg = <i>gn</i>	Change the contents of the CPU register to <i>gn</i> .
CMS = <i>gn</i>	Change the contents of the MSR register to <i>gn</i> .
CPT {0 1} = <i>gn</i>	Change the contents of the PTB0 or PTB1 register to <i>gn</i> .
CEI = <i>gn</i>	Change the contents of the EADDR/INVAL register to <i>gn</i> .
CSC = <i>gn</i>	Change the contents of the MMU SC register to <i>gn</i> .
CBP {0 ... 15 f} = { <i>addr</i> <i>symbol</i> }	Change the current <i>addr</i> or <i>symbol</i> of one of 16 breakpoint registers to <i>addr</i> or <i>symbol</i> . Use 0 to 15 if radix is decimal. Use 0-F if radix is hex.
CBC = <i>gn</i>	Change the contents of the MMU NCNT register to <i>gn</i> .
CF {0 1 2 3 4 5 7} = <i>gn</i>	Change the contents of one of eight floating-point registers to <i>gn</i> .
CFS = <i>gn</i>	Change the contents of the floating-point status register to <i>gn</i> .

CCF = <i>gn</i>	Change the contents of the configuration register to <i>gn</i> .
CMM = <i>gn</i>	Change the contents of the main MSR register to <i>gn</i> .

6.5 DEBUGGER EXECUTION COMMANDS

The debugger execution commands allow the user to trace the logical flow of a program by using commands to conditionally run a properly assembled program. The commands available to the user are: STEP N INSTRUCTIONS, STEP UNTIL, STEP WHILE, RUN, JUMP SUBROUTINE, CALL EXTERNAL SUBROUTINE, and GO.

Sections 6.5.1 through 6.5.6 define the syntax and the usage of each command.

6.5.1 STEP N INSTRUCTIONS Command

The STEP N INSTRUCTIONS command executes a user program beginning at the current PC for *n* instructions followed by a break. If *n* is omitted, then default *n* is equal to one. The syntax of the STEP N INSTRUCTIONS command is as follows:

```
ST [n]
```

6.5.2 STEP UNTIL Command

The syntax of the STEP UNTIL command is as follows:

```
SU M {B|W|D} {addr|gpreg|cpureg} gn1 gn2 gnmask
```

The STEP UNTIL command steps through or executes a program until the contents of memory at either *addr*, or *gpreg* (R0 through R7), or *cpureg* masked with *gnmask*, is greater than, or equal to, *gn1* and less than, or equal to, *gn2*. When *addr* is used, BWD must be specified. *Gn1*, *gn2*, and *gnmask* are truncated to the specified BWD. For example, SU MW67900 1F00 F000 FFFF means step until the next memory word.

6.5.3 STEP WHILE Command

The syntax of the STEP WHILE command is as follows:

```
SW {addr|gpreg|cpureg} gn1 gn2 gnmask
```

The STEP WHILE command executes a program while the memory contents of either *addr*, or *gpreg*, or *cpureg* masked with *gnmask*, are greater than, or equal to, *gn1* and less than, or equal to, *gn2*.

6.5.4 JUMP SUBROUTINE Command

The syntax of the JUMP SUBROUTINE command is:

JS addr

The JUMP SUBROUTINE command simulates the jump subroutine instruction. It calls a user subroutine at address *addr*. When the user subroutine executes the RET instruction, control is passed back to TDS. TDS then prompts with the B RET message.

6.5.5 CALL EXTERNAL SUBROUTINE Command

The syntax of the CALL EXTERNAL SUBROUTINE command is:

CX mod offset

The CX command is the same as the JS command but it simulates the CXP instruction. The subroutine address is calculated using the program address from the module table and subroutine *offset* from link table *offset*. B RET is issued and displayed on the user terminal when a return is made from the external subroutine.

6.5.6 GO Command

The GO command directs TDS to load the CPU internal registers and registers R0 through R7, and then passes control to the user program. The user program runs from the current PC until a breakpoint or final RXP 0 instruction is encountered. The termination of a user program can cause one of two end messages. B RET is issued as the end result of a return from a subroutine (due to the execution of a JS command) since subroutines end with the RET *n* instruction. B END is the end of the more normal B and G (Begin and Go) sequence since the user program ends with the RXP 0 instruction.

The syntax of the GO command is:

G

6.6 SPECIAL PROCESSING COMMANDS

The symbolic debugger supports the use of special commands that allow the user to move blocks of data from one memory location to another, fill contiguous memory locations with specific data, search contiguous memory locations for specific data, and dump portions of memory to the user terminal.

Respectively, those commands are: MOVE BLOCK OF DATA, FILL MEMORY WITH DATA, SEARCH FOR DATA, and HEX MEMORY DUMP. Sections 6.6.1 through 6.6.4 present the syntax of each command and define their uses.

6.6.1 MOVE BLOCK OF DATA Command

The syntax of the MOVE BLOCK OF DATA command is:

```
M addr1 addr2 gn
```

The MOVE BLOCK OF DATA command specifies the moving of *gn* sequential bytes of data from memory location *addr1* to memory location *addr2*.

6.6.2 FILL MEMORY WITH DATA Command

The syntax of the FILL MEMORY WITH DATA command is:

```
F addr1 addr2 gn [B|W|D]
```

The FILL MEMORY WITH DATA command specifies the filling of contiguous memory locations from *addr1* to *addr2* with data *gn*. The default for BWD is byte.

6.6.3 SEARCH FOR DATA Command

The syntax of the SEARCH FOR DATA command is:

```
SR addr1 addr2 gn [B|W|D]
```

The SEARCH FOR DATA command directs TDS to search memory from *addr1* to *addr2* for the first occurrence of the value *gn*, and print its address. If the data is not found, the E SRC (error search failed) debugger error message is output. The default for BWD is B.

6.6.4 HEX MEMORY DUMP Command

The syntax of the HEX MEMORY DUMP command is:

```
D addr1 gn
```

The HEX MEMORY DUMP command instructs TDS to dump *gn* memory locations, beginning at *addr1*, to the user terminal. The output is in hex with ASCII equivalents included on the left side of the line.

6.7 DEBUGGER USER NOTES

Debug command action is intended to operate only on the user memory area. That area starts at PC after the successful assembly of a program and after the begin command (B or BZ) is invoked. Failure to follow this procedure may destroy system parameters.

The debugger supports 16 breakpoints. Breakpoints 0 and 1 are MMU breakpoints and the printing/changing of BP 0 or 1 either reads from or writes to an MMU breakpoint register. The remainder are software breakpoints and may be used on a development board without an MMU. Breakpoints are cancelled after a hardware reset. Breakpoints must be set on instruction boundaries for correct operation. To cancel a specific breakpoint manually, set it to zero. Addresses for breakpoints are taken from a successfully assembled user program by

using the TP (TYPE) command to list lines of the source text. The listing shows the address of the code, the source string, and the actual code generated.

The commands to print/change system registers show the value of the register at the time the user program was run (except the INTBASE base register). The SB register value is printed/changed in the area pointed to by the MOD register. The SP (software register) tracks either the US (user stack) or IS (interrupt stack) depending on the PSR value. SP contains the stack address value at the last entry to TDS.

TDS maintains two images for the MSR register. The MM (main MSR) is used during the time commands are issued to TDS. The MS reflects the value of MSR during the execution of a user program. See Chapter 8 for slave support details.

Input/output base interpretation of debug commands is controlled by the RX command (see Section 3.3.1), with the exception of the Dump command (which always outputs data in hex format), and the print/change F/L (floating and long floating-point formats) which require real numbers.

User program flow trace is provided by the action TDS takes at breakpoints or step stops. At these points, either a B TRC (for step command) or B *n* (breakpoint *n*) and the line number is displayed followed by the code address of that line, and the source string itself.

If a hex value is equivalent to a symbol it must start with a zero for that value to be interpreted as hex; otherwise it will be treated as a symbol.

If it is necessary to use explicit addresses of variables defined symbolically, use the print address command to obtain that value. The automatic or manual initialization of a user program must occur before symbols may be correctly used.

6.8 DEBUGGER ERROR MESSAGES

The error messages output by the symbolic debugger are summarized as follows:

E NMI	Error—non-maskable interrupt
E NVI	Error—non-vectored interrupt (not implemented)
E FPU	Error—FPU trap
E DVZ	Error—divide by zero
E UND	Error—undefined opcode (trying to use non-existing MMU or FPU)
E FLG	Error—flag trap
E BPT	Error—non-debugger BPT instruction
E ILL	Error—illegal for user instruction
E EXT	Error—external abort
E BPR	Error—MMU breakpoint
E NST	Error—MMU nonsequential trace trap

E ABT Error—MMU address translation
E SRC Error—memory search failed
E SVC Error—unknown SVC
E CXP Error—more than one call command
E VRF Memory verify error (breakpoint could not be inserted)

Chapter 7

RUN-TIME SUPPORT

7.1 INTRODUCTION

The principal features of the TDS run-time support are:

1. All functions are accessed through supervisor calls (SVCs).
2. Routines are available which:
 - Support output to a Centronics-type printer.
 - Convert a binary value to an ASCII string.
 - Convert an ASCII string to a binary value.
3. Conversions are available in decimal, hex, and long or short floating-point formats.

The following sections define the uses and specification of each of the TDS run-time support functions. Chapter 8 of this manual conducts a sample user development session and presents a program that illustrates the use of the run-time support routines.

7.2 SERIAL COMMUNICATIONS

The run-time support functions of TDS provide two SVCs that may be used by a user program to establish terminal I/O at port 0 or access port 1 (auxiliary port). The two SVCs provide a user program with the ability to read and write buffer data from/to port 0 or port 1. To implement these functions, the user program must properly set up registers R0 through R4 and execute the Supervisor Call Instruction (SVC). The following sections define the correct setting for registers R0 through R4 for each function. See Appendices A.2, B.2, and C.2 for further information concerning system port 0 and port 1.

7.2.1 Read and Buffer Serial Line Data

- R0: = 3
- R1: = Address of buffer to be written into
- R2: = Number of characters to be read. If number is less than zero, then the port will be read until a <CR> is encountered, but no more than ABS(R2). (<CR> is counted and included in the buffer.)
- R3: = Port to be read
0 for main port
1 for auxiliary port

Upon return from the SVC instruction, R2 will contain the number of characters actually read.

7.2.2 Write Buffered Data to Serial Line

- R0: = 4
- R1: = Address of buffer to be read
- R2: = Number of characters to be written
- R3: = Port characters are to be written to
0 for main port
1 for auxiliary port

7.3 NUMERICAL CONVERSIONS

TDS run-time support provides two numerical conversion routines. An ASCII string to binary value conversion routine, and a binary value to an ASCII string conversion routine. The conversion base is user-selected to decimal, hex, or short or long floating-point. Sections 7.3.1 and 7.3.2 list the correct setups for registers R0 through R4 that the user program must perform. After setting up the registers, the user program issues an SVC. See Appendices A.3, B.3, and C.3 for additional numerical information.

7.3.1 ASCII to Binary Conversion

- R0: = 5
- R1: = Starting address of ASCII string buffer
- R2: = Length of ASCII string (up to and including <CR>)
- R3: = Destination address of binary value
- R4: = Conversion base

If R4 is not equal to 0 when the conversion is completed, a conversion error occurred.

TDS provides four conversion bases:

Hexadecimal conversion	Base = 3
Decimal conversion	Base = 2
Long floating-point conversion	Base = 1
Short floating-point conversion	Base = 0

7.3.2 Binary to ASCII Conversion

- R0: = 6
- R1: = Address of binary value
- R2: = Conversion base
- R3: = Address of output character buffer (35-byte minimum capacity)

7.4 PARALLEL PRINTER COMMUNICATIONS

The TDS run-time support facilities provide a user program with the ability to access and use a Centronics-type parallel printer. The user program must set up registers R0 through R2 in the following manner then issue an SVC:

- R0: = 7
- R1: = Address of printable text
- R2: = Number of characters to be printed

If bit 31 of R2 is not set, the printer routine will output a line feed every time a <CR> is encountered.

If the printer is not on-line when the printer routine SVC call is made, the printer routine will wait approximately one minute, after which it will output the "check lpt:" error message. See Sections A.4, B.4, and C.4 for specific printer interface data.

Chapter 8

PROGRAM EXECUTION AND SAMPLE DEVELOPMENT SESSION

8.1 INTRODUCTION

This chapter describes the use of the text editor to create the source text of a sample program, as well as the use of the assembler to assemble that sample program and run the program. The chapter also presents a sample program illustrating the functions of the routines provided by the TDS run-time support facilities. In addition, this chapter details the TDS slave support facilities and presents a sample program for the MMU support.

8.2 TEXT EDITING

When the development board has been powered up and reset, and the IT command has been entered, the user may select the terminal communication mode using the OM command and the default base using the RX command.

Use the text editor commands to enter the following source program into the text editor buffer. If the text editor buffer is empty, use the INSERT command to begin the input of source text. Use the other editor commands to manipulate the text as needed. The use of the TYPE command shows all source lines numbered and in sequence. If any of the source lines are changed, the TYPE command will display the properly sequenced and updated line numbers.

```

;TDS DEMO PROGRAM
;DEFINE MESSAGES
;
BR AA:W
P1:  .BYTE "WRITE_$$"
      .BYTE "GET_LINE"
      .BYTE "OUT_LINE"
HU:  .BYTE "SAY-HUH"
      .BYTE H'OD

;
;CASE LIMITS
IX:  .BYTE 2
      .BYTE 0

;
;PROGRAM START
AA:  MOVQD 3, R0
      ADDR 0(SB), R1
      MOVXBD -9, R2
      MOVQD 0, R3
;SUPERVISOR CALL READ
SVC

;
CMPB "!", 0(SB)
;IF ! THEN DONE
BEQ EX:W
MOVQD 0, R7
BR LS:B
LI:  ADDQD 1, R7
;PARSE INPUT
LS:  CMPMB 0(R1), P1 [R7:Q], 8
      BEQ CA:W
      CHECKB R6, IX, R7
      BFC LI
;OUTPUT ERROR MESSAGE
MOVQD 4, R0
ADDR HU, R1
MOVZBD 8, R2
SVC
BR AA
CA:  CASEW TB:B [R7:W]
TB:  .WORD WR - CA
      .WORD GT - CA
      .WORD OT - CA
DS:  .BYTE "$$$$$$"
      .BYTE H'OD
WR:  MOVQD 4, R0

```

```

                ADDR DS, R1
                MOVZBD 8, R2
                SVC
                BR AA
GT:             MOVQD 3, R0
                ADDR 12(SB), R1
                MOVXBD -9, R2
                SVC
                MOVB R2, 24(SB)
                BR AA
OT:             MOVQD 4, R0
                ADDR 12(SB), R1
                MOVZBD 24(SB), R2
                SVC
                BR AA
EX:             RXP 0
                ;END OF PROGRAM

```

The sample program demonstrates TDS ability to handle special instructions and provide user access to terminal I/O. Note that if a static base variable segment is used, it must be the first element of the program.

8.3 ASSEMBLY

Once the source text of the sample program has been entered into the text editor buffer, the source program is ready to be assembled. If a listing device is available, use the RL command to select either the auxiliary port (port 1) or the parallel printer port.

This example uses an explicit address for the beginning of the code. In this case, the Begin command cannot be used without an explicit mod table address.

To assemble the program, enter the following:

```
AS H'E000
```

If the address is too low, an error message will be output and the process halted. In such a case, enter a higher address to continue.

Once the assembly process begins, both passes (Pass1 and Pass2) are displayed on the user terminal.

Pass1 error messages are denoted by the ERR_xxx appearing on the left-most side of the screen to mark incorrect lines of code. By using <ctrl/S> to stop the scrolling of the display, the user can record the line numbers of incorrect code, and then use the <ctrl/Q> to continue the scrolling of the listing. If errors are detected during Pass1, the assembly process is terminated at the end of Pass1 and the number of errors detected is displayed. Use the text editor commands to correct lines of source text that are in error. Use the same start address (H'E000) and assemble the program again. When Pass1 is error-free, the assembler proceeds with Pass2.

Pass2 shows the address of the code, the source string, the actual code generated, and any error messages. Use <ctrl/S> to record the line numbers of bad lines of code and use <ctrl/Q> to

continue the listing. If errors are detected in Pass2, the assembler lists the number of errors at the end of Pass2. Correct any errors and use H'E000 to assemble the program again.

When Pass1 and Pass2 are completed with no errors, the program has been successfully assembled. The following is a sample of the listing generated.

```
                                ;TDS DEMO PROGRAM
                                ;DEFINE MESSAGES
                                ;
0000E000  BR AA: W
                                EA8025
0000E003  P1:  .BYTE "WRITE_$$"
                                57524954455F2424
0000E00B          .BYTE "GET_LINE"
                                4745545F4C494E45
0000E013          .BYTE "OUT_LINE"
                                4F55545F4C494E45
0000E01B  HU:  .BYTE "SAY-HUH"
                                5341592D485548
0000E022          .BYTE H'OD
                                OD
                                ;
                                ;CASE LIMITS
0000E023  1X.BYTE 2
                                02
0000E024          BYTE 0
                                00
                                ;
                                ;PROGRAM START
0000E025  AA:  MOVQD 3, R0
                                DF01
0000E027          ADDR 0(SB), R1
                                67D000
0000E02A          MOVXBD -9, R2
                                CE9CA0F7
```

8.4 DEBUG DATA

After the successful assembly of the program, the use of the TYPE command will display the line number of each line of code, the address of each line of code and the source text. The address of each line of code is useful for setting up breakpoints.

If the editor commands are used to change the source text, the previously generated addresses and code are invalidated and the source line address will no longer appear in the display if the TYPE command is used.

Symbols as parameters will not be valid until the user program parameters are initialized. If a command parameter does not accept symbols, use the PAD command to obtain the explicit address. The following is a sample listing (using the TYPE command) output after a successful assembly.

```

                ; TDS DEMO PROGRAM
                ; DEFINE MESSAGES
                ;
0001 0000E000          BR AA:W
0002 0000E003          P1: .BYTE "WRITE_$$"
0003 0000E00B          BYTE "GET_LINE"
0004 0000E013          BYTE "OUT_LINE"
0005 0000E01B          HU: .BYTE "SAY-HUH"
0006 0000E022          .BYTE H'OD
0007
0008                  ;
0009                  : CASE LIMITS
0010 0000E023          IX: .BYTE 2
0011                  .BYTE 0
0012
0013                  ;
0014                  : PROGRAM START
0015 0000E02A          AA: MOVQD 3, R0
                        ADDR 0(SB), R1
                        MOVXBD -9, R2

```

8.5 PROGRAM INITIALIZATION AND EXECUTION

As discussed in Chapter 6, a successfully assembled program must be initialized, either manually or automatically, before it can be run. The manual method entails creating a module table and setting up specific system registers while the automatic method performs the functions of the manual method automatically.

Once the program is successfully assembled, construct the module table as described in Chapter 6.2.1.

The program is now properly initialized and may be run. To run the program, enter:

G

If **WRITE_\$\$** is entered as follows:

WRITE_\$\$

A string of **\$\$\$\$\$\$** is displayed on the user terminal.

If **GET_LINE** is entered and then some string of characters is input, entering **OUT_LINE** causes the character string to be displayed on the user terminal.

Entering invalid inputs, prompts the program to display "SAY-HUH".

To terminate the program and pass control back to TDS, enter ! (for done).

8.6 RUN-TIME SUPPORT EXAMPLES

The following source program demonstrates the access of TDS run-time support routines by a user program. The program specifically illustrates the use of the numerical conversion routines, the printer routine, and indirectly, terminal I/O. Enter and assemble the sample source program.

```
; THIS IS A DEMO OF TDS RTS ROUTINES
; FOR TERMINAL I/O, NUMBER CONVERSION, AND
; PRINTER ACCESS
;
.STATIC ; DEFINE GLOBAL MEMORY
IN:   .BLKB 80           ; INPUT BUFFER
BA:   .BLKB             ; CONVERSION BASE
PR:   .BLKB             ; FLAG TO PRINT RESULT
OT:   .BLKB 35          ; RESULT BUFFER
N1    .BLKB 2           ; A BINARY NUMBER (INTERGER TO LONG FLOATING)
.ENDSEG
;
BR ST:W
LF:   .WORD H'0A0D      ; LF THEN CR
MS:   .BYTE "ENTER A NUMBER> " ; PROMPT MESSAGE
; PROGRAM START
ST:   MOVQD 4, R0       ; SET UP PARAMETERS
      ADDR MS, R1      ; FOR SVC WRITE
      MOVZBD 16, R2
      MOVQD 0, R3       ; MAIN TERMINAL
      SVC              ; OUTPUT MESSAGE
;
; GET RESPONSE VIA SVC READ
      MOVQD 3, R0       ; READ OPERATION
      ADDR IN, R1
      MOVXBD -80, R2    ; 80 CHARACTERS OR UNTIL CR
SVC
;
; CONVERT ASCII INPUT TO A NUMBER
B1:   MOVQD 5, R0       ; LABEL FOR BP
      ADDR IN, R1
      ; COUNT IS ALREADY IN R2
      ADDR N1, R3       ; DESTINATION OF RESULT
      MOVB BA, R4       ; PASS CONVERSION BASE
      SVC               ; GET <NUM
      CMPQD 0, R4       ; CHECK FOR ERROR
      BNE EX:W         ; TERMINATE PROGRAM IF NOT NUMBER
; CONVERT VALUE TO ASCII STRING
B2:   MOVQD 6, R0
      ADDR N1, R1
      MOVB BA, R2
```

```

        ADDR OT, R3
        SVC                                ; PUT <NUM
; WRITE TO TERMINAL CONVERTED RESULT
B3:    MOVQD 4, R0
        MOVD R3, R1                        ; R1 POINT TO THE RESULT BUFFER
        MOVZBD 35, R2                      ; OUTPUT THE MAX CHAR
        MOVQD 0, R3
        SVC
        ADDR LF, R1                         ; OUTPUT CR LF
        MOVZBD 2, R2
        SVC
; CHECK TO SEE IF PRINTER OUTPUT IS SET
        CMPQB 0, PR
        BEQ ST
        MOVQD 7, R0
        ADDR OT, R1                        ; PASS POINTER TO CHARACTER BUFFER
        MOVZBD 35, R2
        SVC                                ; TRANSMIT CHARACTERS TO PRINTER
        ADDR LF, R1
        MOVZBD 2, R2                        ; DO A LF AND CR
        SVC
        BR ST                               ; REPEAT
EX:    RXP 0                               ; RETURN CONTROL TO TDS

```

Once the program is successfully assembled and initialized, use the CMB command to set BA (the conversion base value see Section 7.3.1) and set PR to a nonzero value so that converted strings may be printed. Enter **G <CR>** and run the program.

The program prompts with the message, "ENTER A NUMBER".

The program converts ASCII strings to binary values (in the selected base) and places the value in variable N1. The program also converts a binary value (in the selected base) to an ASCII string and places the string in variable OT. OT is also written to the terminal. If PR is set, OT is written to the parallel printer using the printer access routine.

The user can set a breakpoint at B2 (see listing) and then use the PMEM command to observe the binary value of N1. Additionally, the user can set a breakpoint at B3 (see listing) and observe the ASCII string at OT.

The RL command may be used to list the outputs on a Centronics-type parallel printer connected to the parallel printer port, or a serial device connected at port 1.

Sample outputs to the printer are as follows:

Short floating-point	10000000	E -7
Long floating-point	1000000000000000	E -15
Decimal		-12345
Hexadecimal		FFEDCBB

8.7 SLAVE SUPPORT

If the memory management unit (MMU) or the floating-point unit (FPU) are installed on the development board, TDS must be properly configured before the MMU and FPU can be accessed. This is accomplished by using the CCF command to properly set the appropriate bits of the configuration register. Sections 8.7.1 and 8.7.2 contain further information concerning the FPU and the MMU.

8.7.1 FPU Support

Before the FPU is used, it must be reset by writing a zero to the FSR using the CFS command. FPU debugging is supported by the floating-point print and change function of the CMF, CML, PMF, PML commands, and run-time support conversions in floating-point format.

8.7.2 MMU Support

Setting the MMU bit in the configuration register initializes the main MSR and the user MSR to zero. The main MSR (accessed by "MM") must be left at zero unless the user wishes to experiment with supervisor mode/self translation. All debug commands require a one-to-one mapping for access to user variables/PC address if the user MSR (accessed by "MS") is set to translate.

The following sample 10-line program is used to demonstrate MMU experiments. This program is an infinite loop that is halted by pressing the non-maskable interrupt button on the development board. Programs require mod table definition within the PTE and should be terminated by setting a breakpoint at the end.

Create a mod table at H'9200 and set the SB start to H'9030 and PC start to H'9000. Enter and assemble this sample code at H'9000 (a page boundary):


```

ST:  MOVQD  1, R0
      ADDD   R0, R0
      BR     AA: B
BB:  ADDD   R0, R0
      ADDD   R0, R0
AA:  ADDD   R0, R0
      ADDD   R0, R0
      CMPD  100, R0
      BLT   ST
      BR    BB

```

To observe MMU functions in the sample program, the following steps should be taken:

1. Use the F (fill) command to invalidate the page table area by setting it to zero.

```
F A000 A600 0 D
```

2. Set the page table base register 0.

```
CPT0 = A000
```

3. Create valid page table entries.

```
CMD A000 = A407 (full access to level two at A400)
```

```
CMD A520 = 9007 (full access to code at 9000. Note: This is indexed into
level 2 that corresponds to addr H'9000)
```

```
CMD A524 = 9207 (validate an area for the mod table)
```

The user MMU value is set by the CMS command while the rest of the user program parameters are set by entering the B *mod offset*. The program may be started by entering a G command or the program may be stepped through by entering any of the step commands provided by the symbolic debugger. Note that the use of the step commands override MMU BP (break point) and NS (nonsequential) TRACE.

Specific MSR values and their results are as follows:

CMS = 10000	TU bits set. Program runs translated.
CMS = 1810000	UT, FT, TU bits set. User program-flow trace activated. Step through the program and use the AM command to see PF registers track the program.
CMB = 710000	AI, UB, BEN, TU bits set. MMU breakpoints (0,1) enabled and serviced in the abort routine. Try CBPO = 20009004 (set enable bit) to create breakpoint. Breakpoint message will be E BPR. The AC command will verify PC value. Program may continue with a G or Step command.

CMS = 2410000

NT, AI, TU bits set. Nonsequential trap activated.
Message is E NST.

CMS = 10000

To observe page fault, change PTE at A520 = 0 and
write to invalidate register CEI = 9000. When the
program is run, an E ABT message is issued.

Chapter 9

PERIPHERAL INTERFACING

9.1 INTRODUCTION

TDS can support a number of peripheral devices:

- A parallel Centronics-type printer plugged in at the parallel port.
- A serial device (printer or terminal) plugged in at port 1.
- A cassette tape recorder plugged in at the parallel port.

Sections 9.2 through 9.4 discuss each peripheral device serviced; see Appendices A, B, and C for specific data.

9.2 CENTRONICS-TYPE PARALLEL PRINTER

A Centronics-type parallel printer plugged in at the parallel printer port provides the user with hard copies of program listings.

See Sections A.4, B.4, and C.4 for interface cable pins-outs and other cable specification and construction information.

9.3 SERIAL COMMUNICATIONS

TDS supports serial communications with a printer or a terminal at port 1.

Sections A.1, A.2, B.1, B.2, C.1, and C.2 contain information on the procedures required to set the baud rate of ports 0 and 1.

9.4 CASSETTE RECORDER

TDS supports the storage of source programs on cassette tape and the retrieval of source text from the cassette tape. The following sections provide information on the TDS commands used to save and retrieve source programs, the use of the recorder, the cassette interface cable, and error messages.

9.4.1 Data Storage and Retrieval

Two TDS commands support the storage and retrieval of source programs. They are: the Tape Write and Tape Read commands.

The syntax of the Tape Write command is as follows:

```
TW [@n] [addr1 addr2] <CR>
```

If *addr1* and *addr2* are specified, data is read from the *addr1* location (in TDS memory) to the *addr2* location, and written to the cassette tape. If *addr1* and *addr2* are not specified, the entire contents of the text buffer is written to the cassette recorder. The text buffer contains text entered at the user terminal.

@n is interpreted as a decimal value that is used to generate the required timings for a write-to-cassette tape operation. The default value is for a 7-MHz frequency since the CTTL on the development board may range from 6-10 MHz. If a development board is equipped with a crystal other than 12 MHz, the following formula can be used to compute *@n*.

$$n = 14 \times \text{CTTL}$$

where CTTL = 1/2 (crystal frequency)

See Sections A.5, B.5, and C.5 for further information on the cassette interface timing data, and on the method of using an oscilloscope to set a value for *@n*.

The syntax of the tape read command is as follows:

```
TR [addr]
```

If *addr* is specified, data read from the cassette tape is stored beginning at the location in TDS memory specified by *addr*. If *addr* is not specified, it is assumed that the data to be read is a source program and is loaded into the text editor buffer.

9.4.2 Cassette Interface Cable

The cassette interface cable is used to connect the cassette recorder to the parallel printer port. One end of the cable should contain two audio connector plugs for connection to the cassette recorder and the other end must contain a 100-pin connector for connection to the parallel printer port. See Sections A.5, B.5, and C.5 for complete details on the construction of the cassette interface cable.

9.4.3 Recorder Use

If the cassette tape to be used is not new, or has been read from twenty times, it should be erased twice before it is used. The volume control on the recorder should be set at maximum at all times.

To write data to the cassette recorder, the cassette interface cable must be connected to the parallel printer port and the auxiliary out or earphone jack of the cassette recorder. Start the recorder and leave a 20-second silent header on the tape and then (at the user terminal) enter

```
TW [@n] addr1 addr2
```

to begin the write-to-tape operation. Avoid stopping the recorder while the data operation is in progress. At the completion of the operation, TDS is returned to the CLI mode.

To read data back from the recorder (assuming the recorder is cabled up properly), start the recorder and enter

```
TR [addr]
```

on the terminal but do not press <CR>. When the tape is past the plastic leader, press <CR> to start the read-from-tape operation. At the completion of the operation, TDS is returned to the CLI mode.

Error messages output by TDS alert the user to any errors detected during either data operation.

9.4.4 Recorder Error Messages

The error messages output by TDS that support write-to-tape and read-from-tape operations are summarized as follows:

TIM_OUT	No data pulses present. This message occurs approximately 90 seconds after a data operation is initiated if no data pulses are detected.
BAD_FRM	Bad tape format. Attempted to read data that does not conform to the established data format.
CRC_ERR	A CRC error was detected.

9.5 NEW COMMANDS FOR TDS ON THE DB32000 AND DB32016

TDS for the DB32000 and DB32016 has four new commands that allow it to access either RS232 port to communicate with a host system for use of that host's mass storage facilities, *i.e.*, floppy disks in the case of a PC.

With these commands the user can either send or receive from the host the contents of or for the TDS edit buffer or any memory range.

Data is transferred in a special format. TDS uses the ASCII-hex format to represent a binary value. For instance the character "A" has a hexadecimal byte value of H'41. TDS will convert this "A" to two ASCII characters, "41", representing H'41 before it is sent to a host. A null byte (H'00) is converted to "00", and so on. The reverse occurs upon receipt of data from a host, *i.e.*, two bytes of character are converted to one byte of data stored either in the text buffer or in memory.

The record format is:

```
>h1h2h3...h16end-of-record<CR>
```

Every record starts with the character ">", followed by 16 ASCII-hex number values. (16 of these equal 8 physical bytes.) The record is terminated by writing/reading a carriage return (<CR>). If the amount of data sent/received is not an even multiple of 16 bytes, the final

record is padded with null ASCII-hex characters.

End of record format:

An end of record is indicated by a solitary "<" character followed by a carriage return.

Handshake character:

Wherever handshake is mentioned, the sync character used is the asterisk "*".

9.5.1 Read Data In From Port

```
ZI {F|S} {M|A} [startaddr endaddr ]
ZI          read data in
  F          no handshake
  S          handshake
  M          main port
  A          auxiliary port
```

If *addrs* not specified then assume text data and init edit buffer else fill mem from *startaddr* to *endaddr*.

Specifying *addrs* gives you two choices. If *endaddr* is less than EOF on the host the command receive process will terminate upon reaching *endaddr*. The host transmission program will be deadlocked if handshake was specified. Typically specify *endaddr* as some big value like h'f f f f f and let EOF terminate command.

Basic protocol on TDS end is to read data until a <CR>, process it and output a sync character (*) if handshake option was selected. Continue until the first character in line is the "<" character signifying EOF.

9.5.2 Write Data Out To Port

```
ZO {F|S} {M|A} [startaddr endaddr ]
ZO          send data out
  F          no handshake
  S          handshake
  M          main port
  A          auxiliary port
```

If addresses are not specified, the contents of edit buffer is sent from *startaddr* to *endaddr*.

Basic protocol is to read the sync character "*" + <CR> then sent one record; continue until done. Send EOF character "<". If handshake not selected, omit the read.

9.5.3 The AT R Command

@R [*baud*] The @R command will program the auxiliary port (port 1) to the specified baud rate. Baud rate is entered as a byte-long hexadecimal value corresponding to dip switch pattern (see Sections B.2 and C.2 in Appendices B and C for baud rate setup). If baud rate is omitted then the dip switch is read and the baud rate determined from the found pattern. (At reset the auxiliary port is programmed to the same speed as main according to the dip switch.) This command was added since the DB32000 and the DB32016 can have independently set port speeds. Also, the ZI, ZO commands would probably be used with a host connected to the auxiliary port, but running at a slower speed than the terminal.

NOTE: On the DB32016 the auxiliary port USART must be driven by the 8253 timer. This is the only way to have board run the two ports at different baud rates.

9.5.4 The AT M Command

@M [*memaddr*] This command will set high memory value for use by TDS. After the IT command, high mem is set to H'7f f f f (512K) for DB32000. It is set to 27FFF for DB32016 (128K + 64K). During assembly TDS will search memory space up to this default value to find the real end of memory which would depend on the type of mem chips loaded in the board. (Top of mem is used by the TDS assembler for symbol table creation and is also the value used by the auto B command to set users SP.) If *memaddr* is specified, that value will be used as top of memory. This command should be used with caution since there is no checking for validity. If *memaddr* is omitted then the TDS is reset to search for high mem in the same fashion as after the IT command.

NOTE: The IT command nulls any previously entered *memaddr* value. This command was added to preclude the possibility of TDS affecting off-board memory. User is responsible for his/her particular configuration and should be aware of performance tradeoffs due to off-board memory access and non-aligned stack pointer.

Information Common To ZI and ZO

If using the main port, it may be necessary to use the operating mode command (OM) to set terminal characteristics. Use OM3 to eliminate LF's.

Start TDS command first, then start host program, unless doing a no-handshake transmit to host. In this case, start the host end first. A fast (no-handshake) transmit on the main port is not possible.

For every record sent/received a period is written to the user terminal unless user is using the main port in which case this is suppressed. When using main, error messages are not suppressed. The host program should be aware of this.

Error Messages

BAD_SYN	Received wrong sync character
BAD_MEM	Specified forbidden memory, TDS variables, edit buffer, etc.
BAD_FRM	Received data in unknown format

9.5.5 Sample Program

The following is a sample Pascal program required for the host to communicate with TDS.

NOTE: For both send and receive procedures, a single period is written to the host console for every record sent/received. This is a visual indicator that the line is responding to data transfers and is not deadlocked.

PROGRAM TDSCOMP(INPUT, OUTPUT, DATA);

(*-----*)

This Pascal program illustrates the required communication protocol to send/receive over TDS controlled RS232 ports.

This Pascal is not standard. There are references to host-system-dependent I/O functions such as "open" and "close."

Changes may be necessary depending on system implementation of Pascal READ/READLN, WRITE/WRITELN and structure of TEXT file as opposed to FILE OF CHAR. In this instance, LINEOUT was declared as FILE OF CHAR in order to eliminate the OS-generated linefeed sent with every carriage return if LINEOUT was TEXT.


```
CONST CR = 13 ;
```

```
VAR  IOBUF          :  PACKED ARRAY[1..130] OF CHAR ;  
     FILENAME       :  PACKED ARRAY[1..30] OF CHAR ;  
     LINEIN         :  TEXT;           RS232 input  
     LINEOUT        :  FILE OF CHAR;   RS232 output  
     DATA          :  TEXT;           Host-resident file  
  
     COUNT, AMOUNT,  
     DOT           :  INTEGER ;  
     FAST, TOTDS  
     DONE          :  BOOLEAN ;  
     CH            :  CHAR ;
```

```
PROCEDURE START;
```

```
    (* This procedure will set Boolean variables for handshake, and data transfer  
       direction, and will ask for the name of a host resident file. The host file will  
       be created if data is from TDS. *)
```

```
BEGIN
```

```
    WRITELN; WRITE('Handshake? (<Y>es or <N>o) >');  
    READ(CH); READLN;  
    IF CH = 'Y' THEN FAST := FALSE ELSE FAST := TRUE;
```

```
    WRITELN; WRITE('Name of data file?');  
    READLN(FILENAME);
```

```
    WRITELN; WRITE('Direction of transfer? <T>o TDS or <F>rom TDS >');  
    READ(CH); READLN;  
    IF CH = 'T' THEN TOTDS := TRUE ELSE TOTDS := FALSE;  
    WRITELN;  
    IF TOTDS
```

```
        (* System-dependent file handling routines *)
```

```
        THEN BEGIN OPEN(DATA, FILENAME, HISTORY:=OLD); RESET(DATA) END  
        ELSE BEGIN OPEN(DATA, FILENAME, HISTORY:=NEW); REWRITE(DATA) END
```

```
END;
```

```
PROCEDURE SEND ;
```

```
BEGIN
```

```
    (* System dependent line and file opens *)
```

```

IF NOT FAST
  THEN BEGIN
    OPEN(LINEIN, 'LINEIN');
    RESET(LINEIN);
    END;
  OPEN(LINEOUT, 'LINEOUT');
  REWRITE(LINEOUT);

  DONE := FALSE; DOT :=0;

  (* Main loop *)
REPEAT
  (* Read a line of data *)
  READLN(DATA, IOBUF);
  IF IOBUF[1] = '>'
    (* If record starts with '>' then write 33 characters *)
    THEN AMOUNT := 33
    (* Else write only 1, the '<' character, and set done flag true *)
    ELSE BEGIN AMOUNT := 1; DONE := TRUE END;
  COUNT := 1;

REPEAT
  (* Write one char at a time *)
  CH := IOBUF[COUNT];
  WRITE(LINEOUT, CH);
  COUNT := COUNT + 1;

UNTIL COUNT > AMOUNT;

  (* Count number of records written in DOTS *)
  DOT := DOT + 1;
  WRITE('.');
  (* Check if a new dot line should be started *)
  IF DOT > 60 THEN BEGIN DOT := 0; WRITELN END;
  (* Flush out the host buffer by writing a <CR>
  THIS MAY BE HOST DEPENDENT! *)

  WRITE(LINEOUT, CHR(CR));

  (* If no handshake then do not send sync char *) IF NOT FAST THEN BEGIN
    READLN(LINEIN, IOBUF);
    CH := IOBUF[1];
    IF CH <> '*' THEN WRITELN('SYNC ERROR');
    END;

UNTIL DONE
  (* End main loop *)

```

```

IF NOT FAST THEN CLOSE(LINEIN);
CLOSE(LINEOUT);
CLOSE(DATA);
END;

PROCEDURE RECV;
BEGIN
    (* System dependent line and file opens *)
    OPEN(LINEIN, 'LINEIN');
    RESET(LINEIN);
    IF NOT FAST
    THEN BEGIN
        OPEN(LINEOUT, LINEOUT);
        REWRITE('LINEOUT');
        END;

    DONE :=FALSE; DOT :=0;

    (* Main loop *)
    REPEAT
        (* If handshake then write sync char *)
        IF NOT FAST THEN WRITE(LINEOUT, '*', CHR(CR));

        (* Read in line from TDS *)
        READLN(LINEIN, IOBUF);

        (* Determine if a record or end of data *)
        IF IOBUF[1] = '>'
        THEN AMOUNT := 33 ELSE BEGIN AMOUNT := 1; DONE := TRUE END;
        COUNT :=1;
        (* Write to host file, one char at a time *)
        REPEAT
            CH := IOBUF[COUNT];
            WRITE(DATA,CH);
            COUNT := COUNT + 1;
        UNTIL COUNT > AMOUNT;
        (* Set host file eol *)
        WRITELN(DATA);

        DOT := DOT + 1;
        WRITE('.'); IF DOT > 60 THEN BEGIN WRITELN; DOT :=0 END;
        UNTIL DONE;

    (* End main loop *)

```

```
IF NOT FAST THEN CLOSE(LINEOUT);  
CLOSE(LINEIN);  
CLOSE(DATA);  
END
```

```
BEGIN      (*Main procedure*)  
  START ;  
  IF TOTDS  
    THEN SEND  
    ELSE RECV;  
END;
```

Appendix A

TDS ON THE DB16000

A.1 AUXILIARY PORT BLX-351 SETUP PROCEDURES (DB16000)

Gang Mode: (BLX-351 rate is main port rate)

On the BLX-351

1. Remove berg connector between E29 and E30.
2. Remove berg connector between E27 and E28.
3. Wire wrap E25 to E27 to E29.

On the DB16000

1. Close W12 pins 3 to 4.
2. Mount BLX-351 at J3.

Independent Mode: (Program the 8253 on the BLX-351)

On the BLX-351 - (Factory configuration)

1. Jumper E29 and E30
2. Jumper E27 and E28

On the DB16000

1. Assuming that the TDS system is powered up, reset, and initialized, enter the following sequence of commands to program the 8253:

```
CMB C00056=B6 ;8253 mode control
CMB C00054=40 ;LSB of count divisor
CMB C00054=00 ;MSB of count divisor
```

This sequence of commands sets the BLX-351 to run at 1200 baud.

The general formula for divisor is:

$$d = 76800/\text{BAUD}$$

Convert d to hex format. Enter LSB first. A 0 MSB must be entered if MSB = 0.

A.2 BAUD RATE SETUPS (DB16000)

The following dip switch settings on the DB16000 control the baud rates of port 0 and port 1.

S4	S3	S2	S1	Baud Rate
on	on	on	on	19200
on	on	on	off	9600
on	on	off	on	7200
on	on	off	off	4800
on	off	on	on	3600
on	off	on	off	2400
on	off	off	on	2000
on	off	off	off	1800
off	on	on	on	1200
off	on	on	off	600
off	on	off	on	300
off	on	off	off	150
off	off	on	on	134
off	off	on	off	110
off	off	off	on	75
off	off	off	off	50

Switches 5, 6, 7 and 8 are unused.

NOTE: Programming is possible only on a DB16000 equipped with an ICU.

A.3 NUMERICAL DATA (DB16000)

Integer Ranges and Data:

Decimal	-2147483648...21474836647
Hex	0...FFFFFFFF

- No equivalence checking.
- No overflow checking in ASCII to binary conversion.
- Syntax checked in ASCII to binary conversion.
- Hex output interpreted as unsigned.
- Decimal treated as signed for binary to ASCII conversion.
- Decimal treated as signed or unsigned depending on the value in an ASCII to binary conversion.
- Numbers too large for operand size are truncated in the high bits.

Floating-Point Ranges:

Long max = 16 digits \pm 307 exponent range

Short max = 8 digits \pm 37 exponent range

Floating-Point Accuracy: (Conversion with no arithmetic operation)

Long format: 16 digits \pm 2 LSD

14 digits \pm .5 LSD

Short format: 8 digits \pm 2 LSD

6 digits \pm .5 LSD

Floating-Point Errors:

Syntax and range checking for ASCII to binary conversions.

No NaN recognition for binary to ASCII conversion.

A.4 PARALLEL PRINTER INTERFACE DATA (DB16000)

The parallel printer port supports a Centronics or an equivalently strobed printer. This information is provided for the user who wishes to construct a printer interface cable. In this case, a high quality twisted-pair cable is mandatory. However, it is advisable to use the SPX Centronics printer interface cable (part number 601304044-001).

Shown below are the signal pin-outs for the printer interface cable.

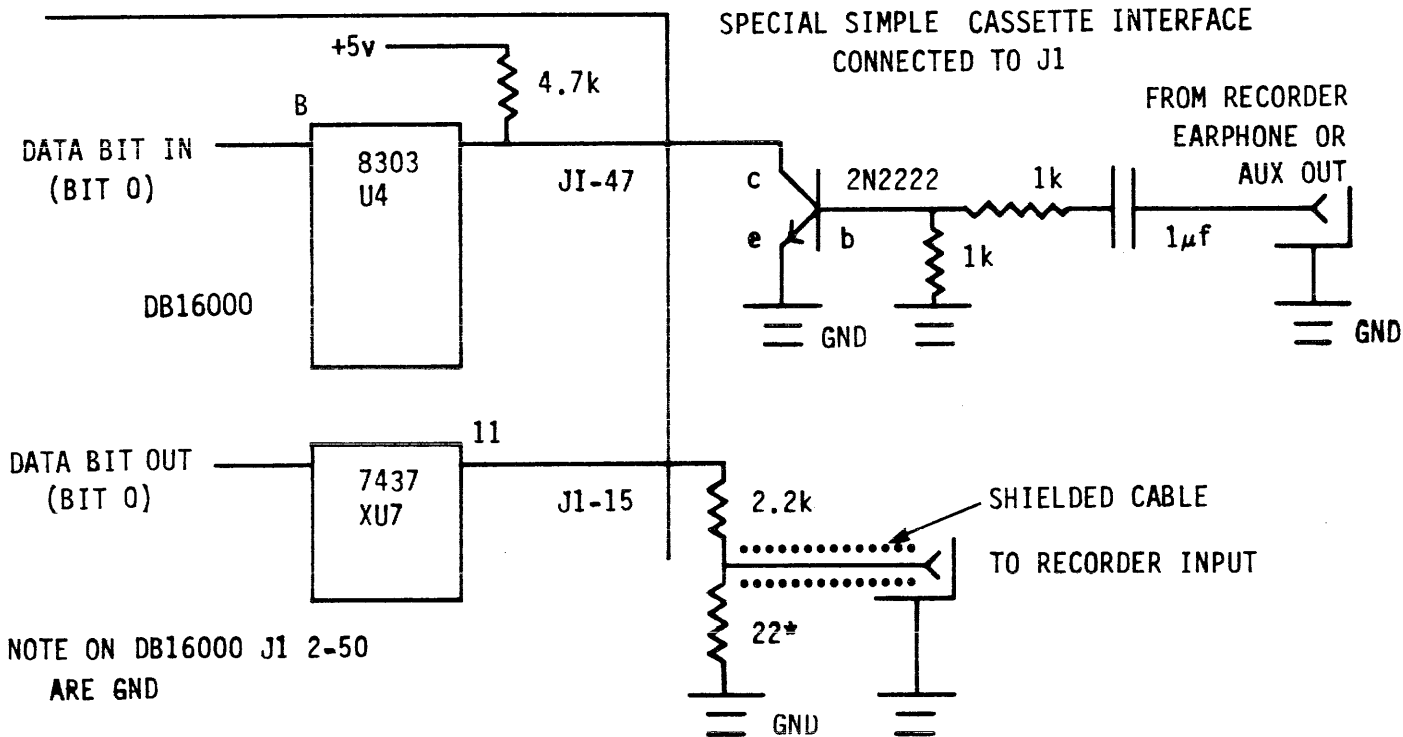
DB16000 PARALLEL PORT	100-PIN EDGE CONNECTOR	36-PIN D CONNECTOR	PRINTER SIGNAL NAMES
47	A24	11 <29>	BUSY (From Printer)
37	A19	13	SELECTED (From Printer)
23	A12	2 <20>	DATA 1 (To Printer)
21	A11	3 <21>	DATA 2 (To Printer)
19	A10	4 <22>	DATA 3 (To Printer)
17	A09	5 <23>	DATA 4 (To Printer)
25	A13	6 <24>	DATA 5 (To Printer)
27	A14	7 <25>	DATA 6 (To Printer)
29	A15	8 <26>	DATA 7 (To Printer)
31	A16	9 <27>	DATA 8 (To Printer)
15	A08	1 <19>	STROBE (To Printer)
	Bn ground	<n> ground	

The software driver logic is compatible with U4 as a 8303 and XU6, XU7, and XU8 as 7437. Ensure that U4 is selected in receive mode (berg connector across 1-2 at W11).

A.5 CASSETTE INTERFACE DATA (DB16000)

CASSETTE INTERFACE CABLE

The cassette interface cable can be constructed from readily available components. It is connected to the auxiliary out or earphone jack of a cassette recorder and the system parallel port. Additionally, interface circuitry is required at the connector to the parallel port. The diagram illustrates the hardware requirements at both ends of the interface cable. The components required for the parallel port interfacing are presented in the parts listing.



FC-01-0

PARTS LIST

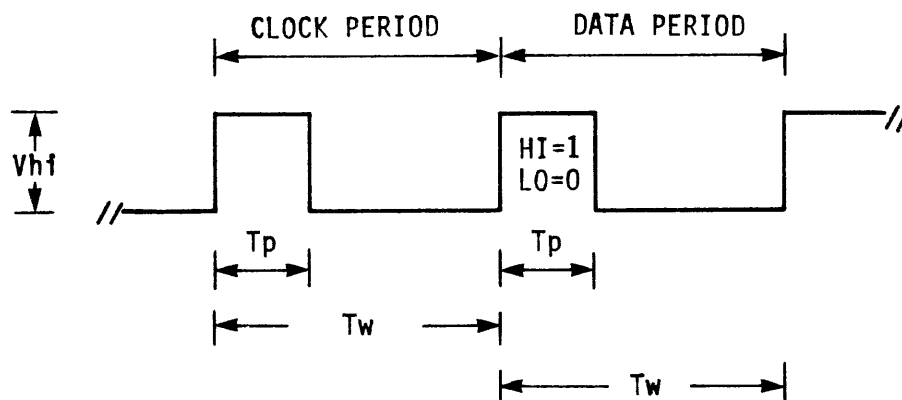
QTY	DESC
1	2N2222 Transistor
2	1K Resistor
1	2.2K Resistor
1	22 ohm Resistor*
1	.1 microfarad Capacitor
2	Audio Connector Plugs
1	100-pin connector

*This value may be dependent on the particular recorder used.

INTERFACE TIMING DATA

Timing data is provided to aid the user in calculating the value of @n as it is used in the TW (tape write) command, for board frequencies other than 14 MHz.

The timings generated by the program produce a serial data flow rate of approximately 330 baud. The effective recording frequency is approximately 660 Hz. The diagram illustrates the timings of pulse waveforms.



$$\text{BAUD RATE} = \frac{1}{2 \times T_w}$$

FC-02-0

OUTPUT PULSE TIMING: (At UX7 pin 11)

$T_p = .500 \text{ ms} \pm 10\%$
 $T_w = 1.5 \text{ ms} \pm 10\%$
 $V_{hi} = \text{TTL level } V_{oh}$

Input to the recorder at the voltage divider should be 20-100 mV depending on the recorder used. The user may connect an oscilloscope probe to pin 11 of UX7 and then use the TW command to enter trial values for n until the output pulse timing shown in the diagram is observed.

INPUT PULSE TIMING: (At U4 pin 8)

$T_p = .280 \text{ ms minimum}$
 $T_w = 1.5 \text{ ms} \pm 10\%$
 $V_{hi} = \text{TTL level } V_{oh}$

This waveform can also be observed at the tape header or by recording binary data with all bytes set to -1, and then by reading the data back.

A.6 ASSEMBLER USER NOTES

The assembler was written to be as consistent as possible with available assembler documentation, but due to its compactness, certain exceptions must be observed. Those exceptions are listed below:

1. ASCII strings are delimited by double quotes only.
2. Symbols are valid for PC address and SB segment mode only.
3. Symbols must contain only two characters.
4. The only operation allowed is subtraction between symbols.

5. Strings are valid only for .BYTE Psuedo-op and immediate operands.
6. R0 through R7 are the only recognized register mnemonics. All others must be hex encoded.
7. CXP argument is simply n.
8. CBITli and SBITli instructions are not implemented.
9. Registers R0 through R7 must be used for F0 through F7.
10. Immediate floating-point values not allowed.
11. The STATIC area must be declared as the first element of the program.
12. Pseudo-ops parse on first four characters only.
13. All lower case characters in the string of .BYTE will be changed to upper case during assembly.
14. LXPd instruction not implemented. Use ADDR EXT(n), DESTINATION.

The assembler does not fully support operand legality-checking, value range checking, and some soft parsing. Immediate operand values are truncated and no error message is issued.

Certain parsing errors such as balanced parentheses are not checked. Characters other than those allowed by syntax rules are not checked for and may not output an error message.

A.7 TDS MEMORY MAP (DB16000)

The diagram is a graphic representation of the memory space available to TDSs.

ADDR 0

PROM RANGE

H'3FFF

TDS Code	
<-H'3F08	TDS Module table
<-H'3FC8	TDS Interrupt and Trap table

H'8000

RAM RANGE

TDS variables	
<-H'8200	
TDS stack	
<-H'8500	
<-H'8600	User IS
<-H'8700	TDS variables
//	//
<-x = H'600 + # of characters in source + 9*(# of lines in source)	
In auto mode user module table is-----	> <- @ first available 1/2 page boundary *
User Code is	-----> <- @ first available page boundary
User Static base	-----> <- @ first available page after code
User SP	-----> <- @ Highest mem addr - 7 (rxp pushed onto stack)

*Unless memory available is greater than or equal to 64K, the module table is constructed within the reserved system area.

A.8 TDS ERROR SUMMARY (DB16000)

TDS system error messages are summarized as follows:

- BAD_TXT Bad text input.
- ERR_ED# Error in edit number.
- BAD_MEM Code start *addr* conflicts with text or debug data. Enter a higher value for *addr*.
- BAD_SEQ Attempting to assemble null or bad text.

BAD_INS	No such instruction.
BAD_PSU	No such Psuedo-op.
ERR_VAL	Bad number syntax or bad operand.
BAD_LIN	Gross line error or standalone label.
ERR_SEG	Using data allocation Psuedo-ops within PC segment or vice versa.
BAD_NUM	Bad floating-point syntax or value range.
BAD_SYM	Duplicate symbol.
UND_SYM	Undefined symbol.
BAD_TAB	Assembler was not initiated by IT command.
E NMI	Error - Non-maskable interrupt.
E NVI	Error - Non-vectored interrupt (not implemented).
E FPU	Error - FPU trap.
E DVZ	Error - divide by zero.
E UND	Error - undefined op-code (trying to use non-existing MMU or FPU).
E FLG	Error - flag trap.
E BPT	Error - non-debugger BPT instruction.
E ILL	Error - illegal for user instruction.
E EXT	Error - external abort.
E BPR	Error - MMU breakpoint.
E NST	Error - MMU non-sequential trace trap.
E ABT	Error - MMU address translation.
E SRC	Error - memory search failed.
E SVC	Error - unknown SVC.
E CXP	Error - more than one call command.
E VRF	Memory verify error (breakpoint could not be inserted).
TIM_OUT	No data pulses present. This message occurs approximately 90 seconds after a data operation is initiated if no data pulses are detected.
BAD_FRM	Bad tape format. Attempted to read data that does not conform to the established data format.
CRC_ERR	A CRC error was detected.

A.9 TDS COMMAND SUMMARY (DB16000)

The commands used in the TDS system are summarized as follows:

RX {H D}	Select the system base to be hexadecimal or decimal.
OM <i>gn</i>	Set the terminal communications mode. See Section 3.3.2.
IN [<i>n</i>] { <i>string</i> }	Insert <i>string</i> above or after line <i>n</i> of the text buffer.
TP [<i>n1</i> [/ <i>n2</i>]]	Type line <i>n1</i> of text buffer or type <i>n2</i> lines of buffer beginning with <i>n1</i> .
RP <i>n</i> { <i>string</i> }	Replace line <i>n</i> of buffer with <i>string</i> .
K <i>n</i>	Kill or delete line <i>n</i> from text buffer.
RS	Reset clear text buffer.
RL [LPT: ASN:] [C]	Redirect listing from terminal to printer at parallel port or serial device at port 1, and end listing with a <CR> or cancel previously selected device.
AS [<i>addr</i>]	Begin the assembly of user program at address <i>addr</i> .
B [Z <i>mod offset</i>]	Initialize a user program if an explicit address was used as assembly time. If an explicit address was not used, <i>mod</i> and <i>offset</i> may be omitted. The Z option zeros memory from the SB to the SP area.
PM {B W D F L} { <i>addr symbol</i> }	Print memory contents according to selected base. Byte, Word, Double-word, Floating-point or Long.
PAD { <i>symbol</i> }	Print address of <i>symbol</i> .
PR {0 1 2 3 4 5 6 7}	Print the contents of one of eight general purpose registers.
Pcpureg	Print the contents of the CPU register.
PMS	Print the contents of the MMU MSR register.
PPT {0 1}	Print the contents of the MMU PTB register.
PEI	Print the contents of the MMU EADDR/INVAL register.
PPF {0 1}	Print the contents of the MMU PF register.

PSC	Print the contents of the MMU SC register.
PBP {0 ... [15 f]}	Print the contents of one of 16 breakpoint registers.
PBC	Print the contents of one of the MMU BCNT registers.
PF {0 1 2 3 4 5 6 7}	Print the contents of one of eight FPU registers.
PFS	Print the contents of the FPU status register.
PCF	Print configuration.
PMM	Print the contents of the main MSF register.
AR	Print the contents of all general purpose registers.
AC	Print the contents of all CPU registers.
AM	Print the contents of all MMU registers.
AF	Print the contents of all FPU registers.
CM {B W D} { <i>addr</i> <i>symbol</i> = <i>gn</i> }	Change memory contents, at specified location, to an integer value.
CM {F L} { <i>addr</i> <i>symbol</i> = <i>rn</i> }	Change memory contents at <i>addr</i> or <i>symbol</i> to a real number.
Cpureg = { <i>gn</i> }	Change contents of the CPU register to <i>gn</i> .
CMS = { <i>gn</i> }	Change the contents of the MSR register to <i>gn</i> .
CPT {0 1} = <i>gn</i>	Change the contents of the PTB register to <i>gn</i> .
CEI = <i>gn</i>	Change the contents of the EADR/INVAL register to <i>gn</i> .
CSC = <i>gn</i>	Change the contents of the MMU SC register.
CBP {0 ... [15 f]} = { <i>addr</i> <i>symbol</i> }	Change the current <i>addr</i> or <i>symbol</i> of one of sixteen breakpoint registers to <i>addr</i> or <i>symbol</i> .
CBC = <i>gn</i>	Change the memory contents of the MMU NCNT register to <i>gn</i> .
CF {0 1 2 3 4 5 7} = <i>gn</i>	Change the memory contents of one of eight FPU registers to <i>gn</i> .

CFS = <i>gn</i>	Change the memory contents of the floating-point status register to <i>gn</i> .
CCF = <i>gn</i>	Change the memory contents of the configuration register to <i>gn</i> .
CMM = <i>gn</i>	Change the memory contents of the main MSR register to <i>gn</i> .
ST [<i>n</i>]	Step through the user program for <i>n</i> instructions and then break. Default for <i>n</i> is one.
SU { <i>addr</i> <i>gpreg</i> <i>cpureg</i> } <i>gn1 gn2 gnmask</i> }	Step through a user program until the contents of memory at <i>addr</i> , or <i>gpreg</i> (R0-R7), or <i>cpureg</i> masked with <i>gnmask</i> , is greater than or equal to <i>gn1</i> and less than or equal to <i>gn2</i> .
SW { <i>addr</i> <i>gpreg</i> <i>cpureggn1 gn2 gnmask</i> }	Step through a user program while the memory contents of either <i>addr</i> , or <i>gpreg</i> , or <i>cpureg</i> masked with <i>gnmask</i> , are greater than or equal to <i>gn1</i> and less than or equal to <i>gn2</i> .
JS <i>addr</i>	Call a subroutine at location <i>addr</i> .
CX <i>mod offset</i>	Call external subroutine at address calculated using <i>mod</i> and <i>offset</i> from the module table.
GO	Begin program execution.
M <i>addr1 addr2</i>	Move <i>gn</i> sequential bytes of data from memory location <i>addr1</i> to <i>addr2</i> .
F <i>addr1 addr2 gn</i> [B W D]	Fill memory locations from <i>addr1</i> to <i>addr2</i> with data <i>gn</i> .
SR <i>addr1 addr2 gn</i> [B W D]	Search memory between <i>addr1</i> and <i>addr2</i> for data <i>gn</i> and print its address according to BWD.
D <i>addr1 gn</i>	Dump <i>gn</i> locations beginning at <i>addr1</i> , to the user terminal.
TW [@ <i>n</i>] [<i>addr1 addr2</i>]	Write data between location <i>addr1</i> and <i>addr2</i> to the cassette recorder. Otherwise, write contents of editor text buffer to the cassette recorder. @ <i>n</i> specifies timings for boards not equipped with a 14 MHz crystal. See Chapter 9.2 for details.

TR [*addr*]

Read data from the cassette tape and store it in memory beginning at location *addr*. Otherwise, load the data into the editor text buffer.

Appendix B

TDS ON THE DB32016

B.1 AUXILIARY PORT SETUP PROCEDURES (DB32016)

As shipped the DB32016 is configured to run at 9600 baud at reset independent of the dip switch. Follow this jumper change sequence to allow port programability.

MAIN PORT0 (J2)	AUX PORT1 (J3)
W21 9-10 Closed	W21 7-8 Closed
W34 4-5 Open	W29 4-5 Open
W34 5-6 Closed	W29 5-6 Closed

At reset the configured port will run at baud determined by the dip switch (see Section B.2). If auxiliary port is so configured, it can be independently programmed by use of the @R command.

B.2 BAUD RATE SETUPS (DB32016)

The following dip switch settings on the DB32016 control the baud rates of port 0 and port 1.

S4	S3	S2	S1	Baud Rate
on	on	on	on	19200
on	on	on	off	9600
on	on	off	on	7200
on	on	off	off	4800
on	off	on	on	3600
on	off	on	off	2400
on	off	off	on	2000
on	off	off	off	1800
off	on	on	on	1200
off	on	on	off	600
off	on	off	on	300
off	on	off	off	150
off	off	on	on	134
off	off	on	off	110
off	off	off	on	75
off	off	off	off	50

Switches 5, 6, 7 and 8 are unused.

NOTES: 1. Required terminal characteristics:

- a. Parity - disabled
 - b. Stop bits = 1
 - c. Character length = 8 bits
2. Port 1 (auxiliary) may be independently programmed using @R command (see Section 9.5.3).
 3. To calculate @R value, ON=0, OFF=1.

B.3 NUMERICAL DATA (DB32016)

Integer Ranges and Data:

Decimal	-2147483648...21474836647
Hex	0...FFFFFFFF

- No equivalence checking.
- No overflow checking in ASCII to binary conversion.
- Syntax checked in ASCII to binary conversion.
- Hex output interpreted as unsigned.
- Decimal treated as signed for binary to ASCII conversion.
- Decimal treated as signed or unsigned depending on the value in an ASCII to binary conversion.
- Numbers too large for operand size are truncated in the high bits.

Floating-Point Ranges:

Long max = 16 digits \pm 307 exponent range
Short max = 8 digits \pm 37 exponent range

Floating-Point Accuracy: (Conversion with no arithmetic operation)

Long format:	16 digits \pm 2 LSD
	14 digits \pm .5 LSD
Short format:	8 digits \pm 2 LSD
	6 digits \pm .5 LSD

Floating-Point Errors:

- Syntax and range checking for ASCII to binary conversions.
- No NaN recognition for binary to ASCII conversion.

B.4 PARALLEL PRINTER INTERFACE DATA (DB32016)

The parallel printer port supports a Centronics or an equivalently strobed printer. This information is provided for the user who wishes to construct a printer interface cable. In this case, a high quality twisted-pair cable is mandatory. However, it is advisable to use the SPX Centronics printer interface cable (part number 601304044-001).

Shown below are the signal pin-outs for the printer interface cable.

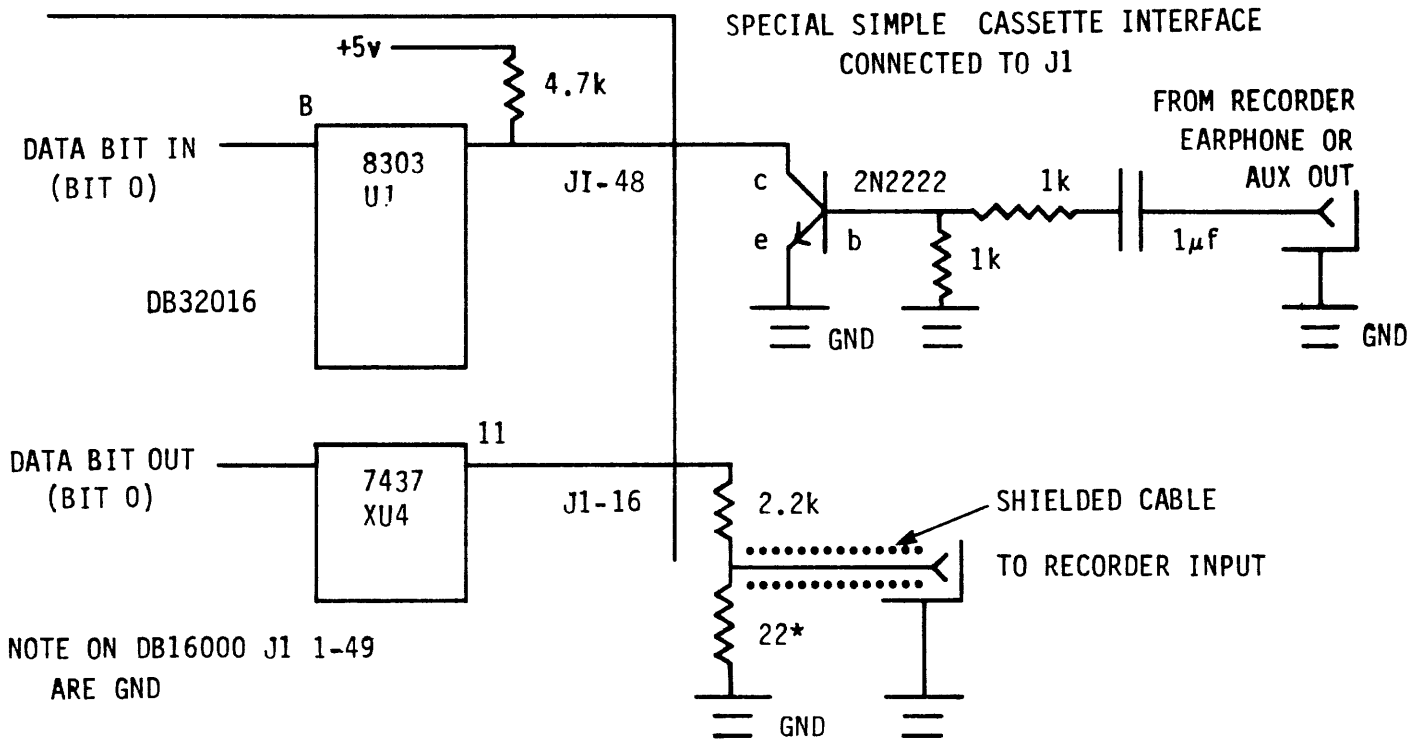
DB32016 PARALLEL PORT	100-PIN EDGE CONNECTOR	36-PIN D CONNECTOR	PRINTER SIGNAL NAMES
48	A24	11 <29>	BUSY (From Printer)
38	A19	13	SELECTED (From Printer)
24	A12	2 <20>	DATA 1 (To Printer)
22	A11	3 <21>	DATA 2 (To Printer)
20	A10	4 <22>	DATA 3 (To Printer)
18	A09	5 <23>	DATA 4 (To Printer)
26	A13	6 <24>	DATA 5 (To Printer)
28	A14	7 <25>	DATA 6 (To Printer)
30	A15	8 <26>	DATA 7 (To Printer)
32	A16	9 <27>	DATA 8 (To Printer)
16	A08	1 <19>	STROBE (To Printer)
	Bn ground	<n> ground	

The software driver logic is compatible with U4 as a 8303 and XU6, XU7, and XU8 as 7437. Ensure that U4 is selected in receive mode (berg connector across 1-2 at W11).

B.5 CASSETTE INTERFACE DATA (DB32016)

Cassette Interface Cable

The cassette interface cable can be constructed from readily available components. It is connected to the auxiliary out or earphone jack of a cassette recorder and the system parallel port. Additionally, interface circuitry is required at the connector to the parallel port. The diagram illustrates the hardware requirements at both ends of the interface cable. The components required for the parallel port interfacing are presented in the parts listing.



FC-03-0

PARTS LIST

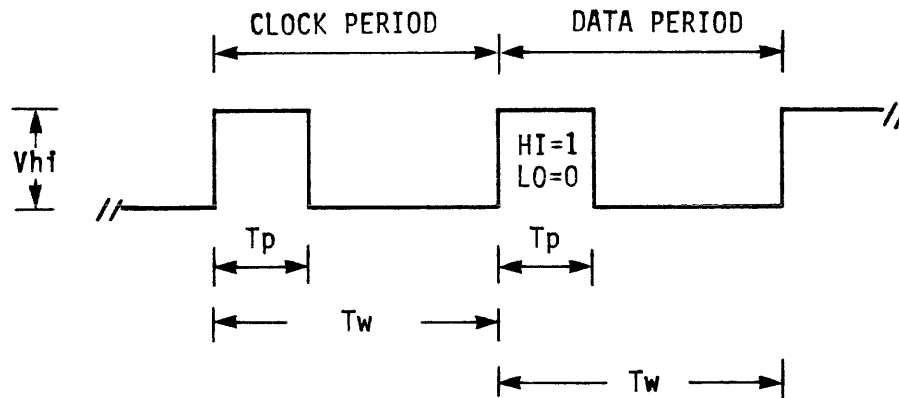
QTY	DESC
1	2N2222 Transistor
2	1K Resistor
1	2.2K Resistor
1	22 ohm Resistor*
1	.1 microfarad Capacitor
2	Audio Connector Plugs
1	100-pin connector

*This value may be dependent on the particular recorder used.

INTERFACE TIMING DATA

Timing data is provided to aid the user in calculating the value of @n as it is used in the TW (tape write) command, for board frequencies other than 14 MHz.

The timings generated by the program produce a serial data flow rate of approximately 330 baud. The effective recording frequency is approximately 660 Hz. The diagram illustrates the timings of pulse waveforms.



$$\text{BAUD RATE} = \frac{1}{2 \times T_w}$$

FC-02-0

OUTPUT PULSE TIMING: (At UX7 pin 11)

- $T_p = .500 \text{ ms} \pm 10\%$
- $T_w = 1.5 \text{ ms} \pm 10\%$
- $V_{hi} = \text{TTL level } V_{oh}$

Input to the recorder at the voltage divider should be 20-100 mV depending on the recorder used. The user may connect an oscilloscope probe to pin 11 of UX7 and then use the TW command to enter trial values for n until the output pulse timing shown in the diagram is observed.

INPUT PULSE TIMING: (At U4 pin 8)

$T_p = .280$ ms minimum

$T_w = 1.5$ ms \pm 10%

$V_{hi} =$ TTL level V_{oh}

This waveform can also be observed at the tape header or by recording binary data with all bytes set to -1, and then by reading the data back.

B.6 ASSEMBLER USER NOTES (DB32016)

The assembler was written to be as consistent as possible with available assembler documentation, but due to its compactness, certain exceptions must be observed. Those exceptions are listed below:

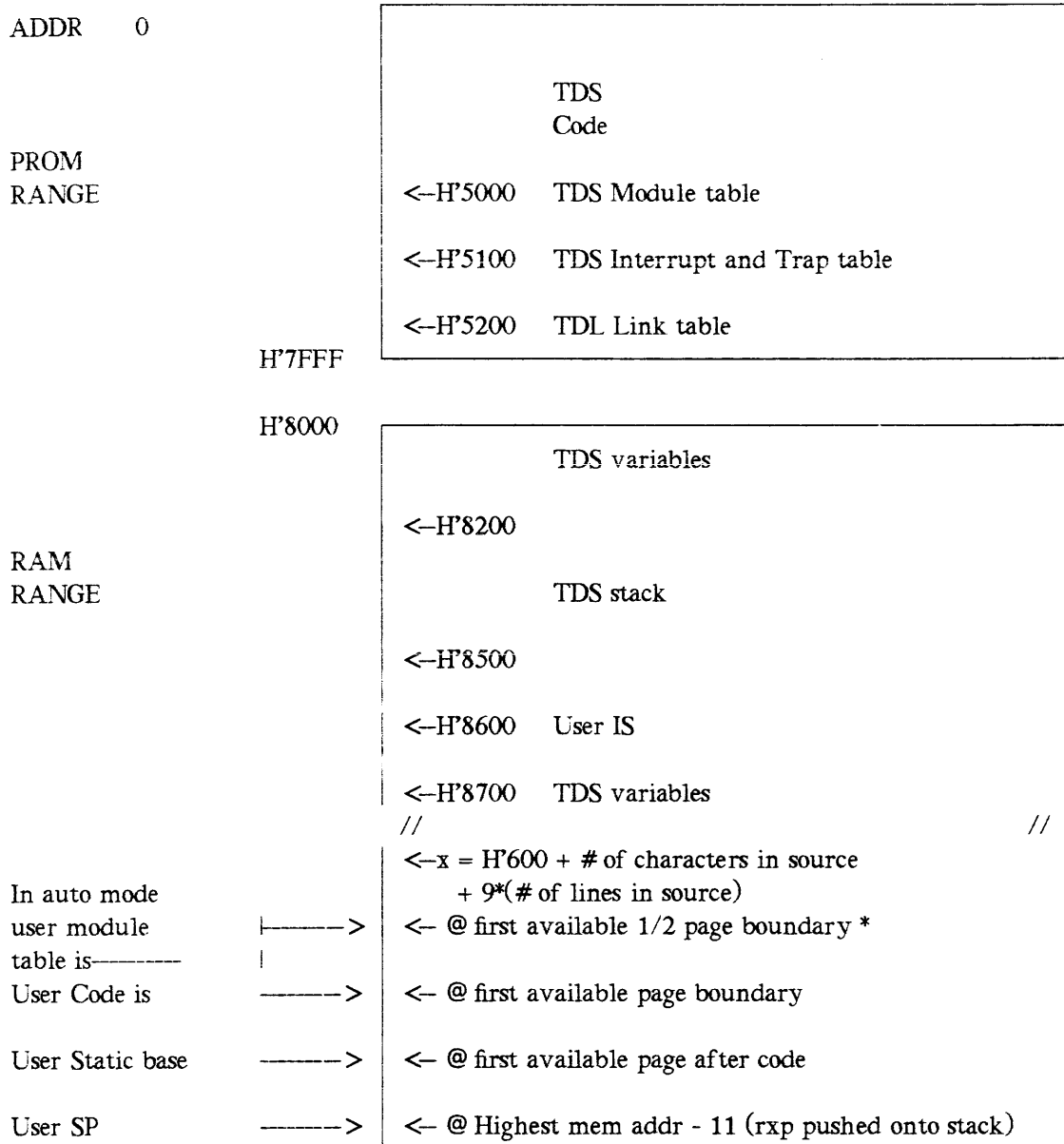
1. ASCII strings are delimited by double quotes only.
2. Symbols are valid for PC address and SB segment mode only.
3. Symbols must contain only two characters.
4. The only operation allowed is subtraction between symbols.
5. Strings are valid only for .BYTE Pseudo-op and immediate operands.
6. R0 through R7 are the only recognized register mnemonics. All others must be hex encoded.
7. CXP argument is simply n.
8. CBITi and SBITi instructions are not implemented.
9. Registers R0 through R7 must be used for F0 through F7.
10. Immediate floating-point values not allowed.
11. The STATIC area must be declared as the first element of the program.
12. Pseudo-ops parse on first four characters only.
13. All lower case characters in the string of .BYTE will be changed to upper case during assembly.
14. LXPD instruction not implemented. Use ADDR EXT(n), DESTINATION.

The assembler does not fully support operand legality checking, value range checking, and some soft parsing. Immediate operand values are truncated and no error message is issued.

Certain parsing errors such as balanced parentheses are not checked. Characters other than those allowed by syntax rules are not checked for and may not output an error message.

B.7 TDS MEMORY MAP (DB32016)

The diagram is a graphic representation of the memory space available to TDSs.



*Unless memory available is greater than or equal to 64K, the module table is constructed within the reserved system area.

B.8 TDS ERROR SUMMARY (DB32016)

TDS system error messages are summarized as follows:

BAD_TXT	Bad text input.
ERR_ED#	Error in edit number.
BAD_MEM	Code start <i>addr</i> conflicts with text or debug data. Enter a higher value for <i>addr</i> .
BAD_SEQ	Attempting to assemble null or bad text.
BAD_INS	No such instruction.
BAD_PSU	No such Psuedo-op.
ERR_VAL	Bad number syntax or bad operand.
BAD_LIN	Gross line error or standalone label.
ERR_SEG	Using data allocation Psuedo-ops within PC segment or vice versa.
BAD_NUM	Bad floating-point syntax or value range.
BAD_SYM	Duplicate symbol.
UND_SYM	Undefined symbol.
BAD_TAB	Assembler was not initiated by IT command.
E NMI	Error - Non-maskable interrupt.
E NVI	Error - Non-vectored interrupt (not implemented).
E FPU	Error - FPU trap.
E DVZ	Error - divide by zero.
E UND	Error - undefined op-code (trying to use non-existing MMU or FPU).
E FLG	Error - flag trap.
E BPT	Error - non-debugger BPT instruction.
E ILL	Error - illegal for user instruction.
E EXT	Error - external abort.
E BPR	Error - MMU breakpoint.
E NST	Error - MMU non-sequential trace trap.
E ABT	Error - MMU address translation.
E SRC	Error - memory search failed.
E SVC	Error - unknown SVC.
E CXP	Error - more than one call command.

E VRF	Memory verify error (breakpoint could not be inserted).
TIM_OUT	No data pulses present. This message occurs approximately 90 seconds after a data operation is initiated if no data pulses are detected.
BAD_FRM	Bad tape format. Attempted to read data that does not conform to the established data format.
CRC_ERR	A CRC error was detected.

B.9 TDS COMMAND SUMMARY (DB32016)

The commands used in the TDS system are summarized as follows:

RX {H D}	Select the system base to be hexadecimal or decimal.
OM <i>gn</i>	Set the terminal communications mode. See Section 3.3.2.
IN [<i>n</i>] <i>string</i>	Insert <i>string</i> above or after line <i>n</i> of the text buffer.
TP [<i>n1</i> [/ <i>n2</i>]]	Type line <i>n1</i> of text buffer or type <i>n2</i> lines of buffer beginning with <i>n1</i> .
RP <i>n string</i>	Replace line <i>n</i> of buffer with <i>string</i> .
K <i>n</i>	Kill or delete line <i>n</i> from text buffer.
RS	Reset clear text buffer.
RL [LPT: ASN:] [C]	Redirect listing from terminal to printer at parallel port or serial device at port 1, and end listing with a <CR> or cancel previously selected device.
AS [<i>addr</i>]	Begin the assembly of user program at address <i>addr</i> .
B {Z <i>mod offset</i> }	Initialize a user program if an explicit address was used as assembly time. If an explicit address was not used, <i>mod</i> and <i>offset</i> may be omitted. The Z option zeros memory from the SB to the SP area.
PM {B W D F L} { <i>addr symbol</i> }	Print memory contents according to selected base. Byte, Word, Double-word, Floating-point or Long.
PAD <i>symbol</i>	Print address of <i>symbol</i> .

PR {0 1 2 3 4 5 6 7}	Print the contents of one of eight general purpose registers.
<i>Pcpureg</i>	Print the contents of the CPU register.
PMS	Print the contents of the MMU MSR register.
PPT {0 1}	Print the contents of the MMU PTB register.
PEI	Print the contents of the MMU EADDR/INVAL register.
PPF {0 1}	Print the contents of the MMU PF register.
PSC	Print the contents of the MMU SC register.
PBP {0 ... [15 f]}	Print the contents of one of 16 breakpoint registers.
PBC	Print the contents of one of the MMU BCNT registers.
PF {0 1 2 3 4 5 6 7}	Print the contents of one of eight FPU registers.
PFS	Print the contents of the FPU status register.
PCF	Print configuration.
PMM	Print the contents of the main MSF register.
AR	Print the contents of all general purpose registers.
AC	Print the contents of all CPU registers.
AM	Print the contents of all MMU registers.
AF	Print the contents of all FPU registers.
CM {B W D} { <i>addr symbol=gn</i> }	Change memory contents, at specified location, to an integer value.
CM {F L} { <i>addr symbol=rn</i> }	Change memory contents at <i>addr</i> or <i>symbol</i> to a real number.
<i>Ccpureg = gn</i>	Change contents of the CPU register to <i>gn</i> .
<i>CMS = gn</i>	Change the contents of the MSR register to <i>gn</i> .
<i>CPT {0 1} = gn</i>	Change the contents of the PTB register to <i>gn</i> .
<i>CEI = gn</i>	Change the contents of the EADR/INVAL register to <i>gn</i> .

CSC = <i>gn</i>	Change the contents of the MMU SC register.
CBP {0 ... [15 f]} = { <i>addr symbol</i> }	Change the current <i>addr</i> or <i>symbol</i> of one of sixteen breakpoint registers to <i>addr</i> or <i>symbol</i> .
CBC = <i>gn</i>	Change the memory contents of the MMU NCNT register to <i>gn</i> .
CF {0 1 2 3 4 5 7} = <i>gn</i>	Change the memory contents of one of eight FPU registers to <i>gn</i> .
CFS = <i>gn</i>	Change the memory contents of the floating-point status register to <i>gn</i> .
CCF = <i>gn</i>	Change the memory contents of the configuration register to <i>gn</i> .
CMM = <i>gn</i>	Change the memory contents of the main MSR register to <i>gn</i> .
ST [<i>n</i>]	Step through the user program for <i>n</i> instructions and then break. Default for <i>n</i> is one.
SU { <i>addr gpreg cpureg</i> } <i>gn1 gn2 gnmask</i>	Step through a user program until the contents of memory at <i>addr</i> , or <i>gpreg</i> (R0-R7), or <i>cpureg</i> masked with <i>gnmask</i> , is greater than or equal to <i>gn1</i> and less than or equal to <i>gn2</i> .
SW { <i>addr gpreg cpureg</i> } <i>gn1 gn2 gnmask</i>	Step through a user program while the memory contents of either <i>addr</i> , or <i>gpreg</i> , or <i>cpureg</i> masked with <i>gnmask</i> , are greater than or equal to <i>gn1</i> and less than or equal to <i>gn2</i> .
JS <i>addr</i>	Call a subroutine at location <i>addr</i> .
CX <i>mod offset</i>	Call external subroutine at address calculated using <i>mod</i> and <i>offset</i> from the module table.
GO	Begin program execution.
M <i>addr1 addr2 gn</i>	Move <i>gn</i> sequential bytes of data from memory location <i>addr1</i> to <i>addr2</i> .
F <i>addr1 addr2 gn</i> [B W D]	Fill memory locations from <i>addr1</i> to <i>addr2</i> with data <i>gn</i> .

SR *addr1 addr2 gn* [B|W|D]

Search memory between *addr1* and *addr2* for data *gn* and print its address according to BWD.

D *addr1 gn*

Dump *gn* locations beginning at *addr1*, to the user terminal.

TW [@n] [*addr1 addr2*]

Write data between location *addr1* and *addr2* to the cassette recorder. Otherwise, write contents of editor text buffer to the cassette recorder. @n specifies timings for boards not equipped with a 14 MHz crystal. See Chapter 9.2 for details.

TR [*addr*]

Read data from the cassette tape and store it in memory beginning at location *addr*. Otherwise, load the data into the editor text buffer.

Appendix C

TDS ON THE DB32000

C.1 AUXILIARY PORT SETUP PROCEDURES (DB32000)

The auxiliary port (J2) of the DB32000 is an integral part of the board. It requires no setup unless user wishes to deviate from shipping configuration. Port is fully baud programmable as determined by dip switch setting (see Section C.2).

C.2 BAUD RATE SETUPS (DB32000)

The following dip switch settings on the DB32000 control the baud rates of port 0 and port 1.

S4	S3	S2	S1	Baud Rate
on	on	on	on	19200
on	on	on	off	9600
on	on	off	on	7200
on	on	off	off	4800
on	off	on	on	3600
on	off	on	off	2400
on	off	off	on	2000
on	off	off	off	1800
off	on	on	on	1200
off	on	on	off	600
off	on	off	on	300
off	on	off	off	150
off	off	on	on	134
off	off	on	off	110
off	off	off	on	75
off	off	off	off	50

Switches 5, 6, 7, 8, 9 and 10 unused.

NOTES: 1. Required terminal characteristics:

- a. Parity - disabled
 - b. Stop bits = 1
 - c. Character length = 8 bits
2. At reset, both ports programmed to switch setting.
 3. Port 1 (auxiliary) may be independently programmed using @R command (see Section 9.5.3).
 4. To calculate @R value ON=1, OFF=0.

C.3 NUMERICAL DATA (DB32000)

Integer Ranges and Data:

Decimal	-2147483648...21474836647
Hex	0...FFFFFFFF

- No equivalence checking.
- No overflow checking in ASCII to binary conversion.
- Syntax checked in ASCII to binary conversion.
- Hex output interpreted as unsigned.
- Decimal treated as signed for binary to ASCII conversion.
- Decimal treated as signed or unsigned depending on the value in an ASCII to binary conversion.
- Numbers too large for operand size are truncated in the high bits.

Floating-Point Ranges:

Long max = 16 digits \pm 307 exponent range
Short max = 8 digits \pm 37 exponent range

Floating-Point Accuracy: (Conversion with no arithmetic operation)

Long format:	16 digits \pm 2 LSD
	14 digits \pm .5 LSD
Short format:	8 digits \pm 2 LSD
	6 digits \pm .5 LSD

Floating-Point Errors:

Syntax and range checking for ASCII to binary conversions.
No NaN recognition for binary to ASCII conversion.

C.4 PARALLEL PRINTER INTERFACE DATA (DB32000)

The parallel printer port supports a Centronics or an equivalently strobed printer. This information is provided for the user who wishes to construct a printer interface cable. In this case, a high quality twisted-pair cable is mandatory. However, it is advisable to use the SPX Centronics printer interface cable (part number 601304044-001).

Shown below are the signal pin-outs for the printer interface cable.

DB32000 PARALLEL PORT	100-PIN EDGE CONNECTOR	36-PIN D CONNECTOR	PRINTER SIGNAL NAMES
47	A24	11 <29>	BUSY (From Printer)
37	A19	13	SELECTED (From Printer)
23	A12	2 <20>	DATA 1 (To Printer)
21	A11	3 <21>	DATA 2 (To Printer)
19	A10	4 <22>	DATA 3 (To Printer)
17	A09	5 <23>	DATA 4 (To Printer)
25	A13	6 <24>	DATA 5 (To Printer)
27	A14	7 <25>	DATA 6 (To Printer)
29	A15	8 <26>	DATA 7 (To Printer)
31	A16	9 <27>	DATA 8 (To Printer)
15	A08	1 <19>	STROBE (To Printer)
	Bn ground	<n> ground	

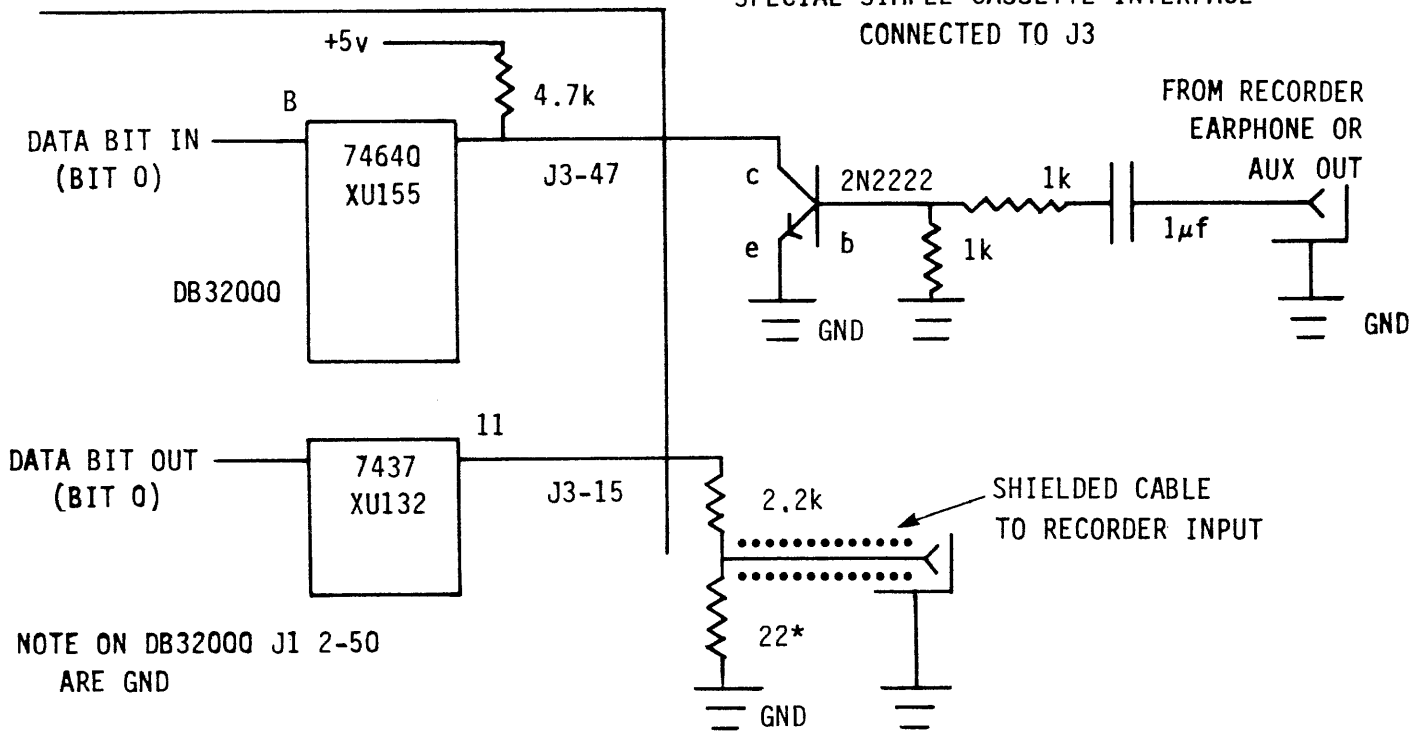
The software driver logic is compatible with XU155 as a 74640 and XU138, XU149, and XU132 as 7437. Ensure that XU155 is selected in receive mode (berg connector across 1-2 at W60).

C.5 CASSETTE INTERFACE DATA (DB32000)

Cassette Interface Cable

The cassette interface cable can be constructed from readily available components. It is connected to the auxiliary out or earphone jack of a cassette recorder and the system parallel port. Additionally, interface circuitry is required at the connector to the parallel port. The diagram illustrates the hardware requirements at both ends of the interface cable. The components required for the parallel port interfacing are presented in the parts listing.

SPECIAL SIMPLE CASSETTE INTERFACE
CONNECTED TO J3



FC-04-0

PARTS LIST

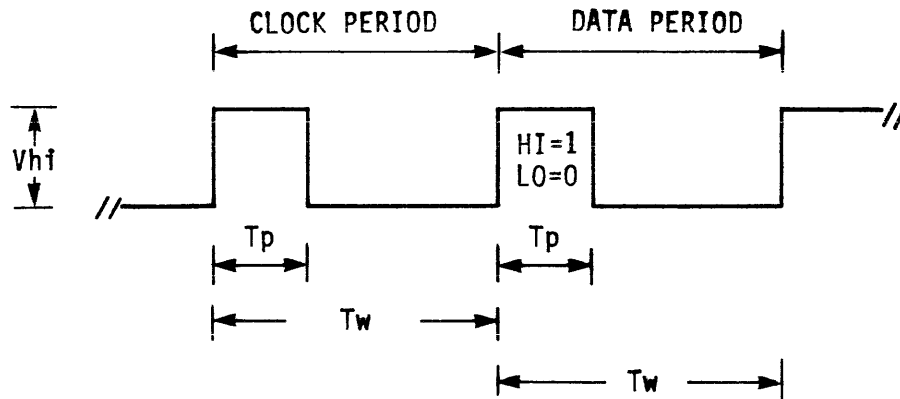
QTY	DESC
1	2N2222 Transistor
2	1K Resistor
1	2.2K Resistor
1	22 ohm Resistor*
1	.1 microfarad Capacitor
2	Audio Connector Plugs
1	100-pin connector

*This value may be dependent on the particular recorder used.

INTERFACE TIMING DATA

Timing data is provided to aid the user in calculating the value of @n as it is used in the TW (tape write) command, for board frequencies other than 14 MHz.

The timings generated by the program produce a serial data flow rate of approximately 330 baud. The effective recording frequency is approximately 660 Hz. The diagram illustrates the timings of pulse waveforms.



$$\text{BAUD RATE} = \frac{1}{2 \times T_w}$$

FC-02-0

OUTPUT PULSE TIMING: (At UX7 pin 11)

$T_p = .500 \text{ ms} \pm 10\%$
 $T_w = 1.5 \text{ ms} \pm 10\%$
 $V_{hi} = \text{TTL level } V_{oh}$

Input to the recorder at the voltage divider should be 20-100 mV depending on the recorder used. The user may connect an oscilloscope probe to pin 11 of UX7 and then use the TW command to enter trial values for n until the output pulse timing shown in the diagram is observed.

INPUT PULSE TIMING: (At U4 pin 8)

$T_p = .280 \text{ ms minimum}$
 $T_w = 1.5 \text{ ms} \pm 10\%$
 $V_{hi} = \text{TTL level } V_{oh}$

This waveform can also be observed at the tape header or by recording binary data with all bytes set to -1, and then by reading the data back.

C.6 ASSEMBLER USER NOTES (DB32000)

The assembler was written to be as consistent as possible with available assembler documentation, but due to its compactness, certain exceptions must be observed. Those exceptions are listed below:

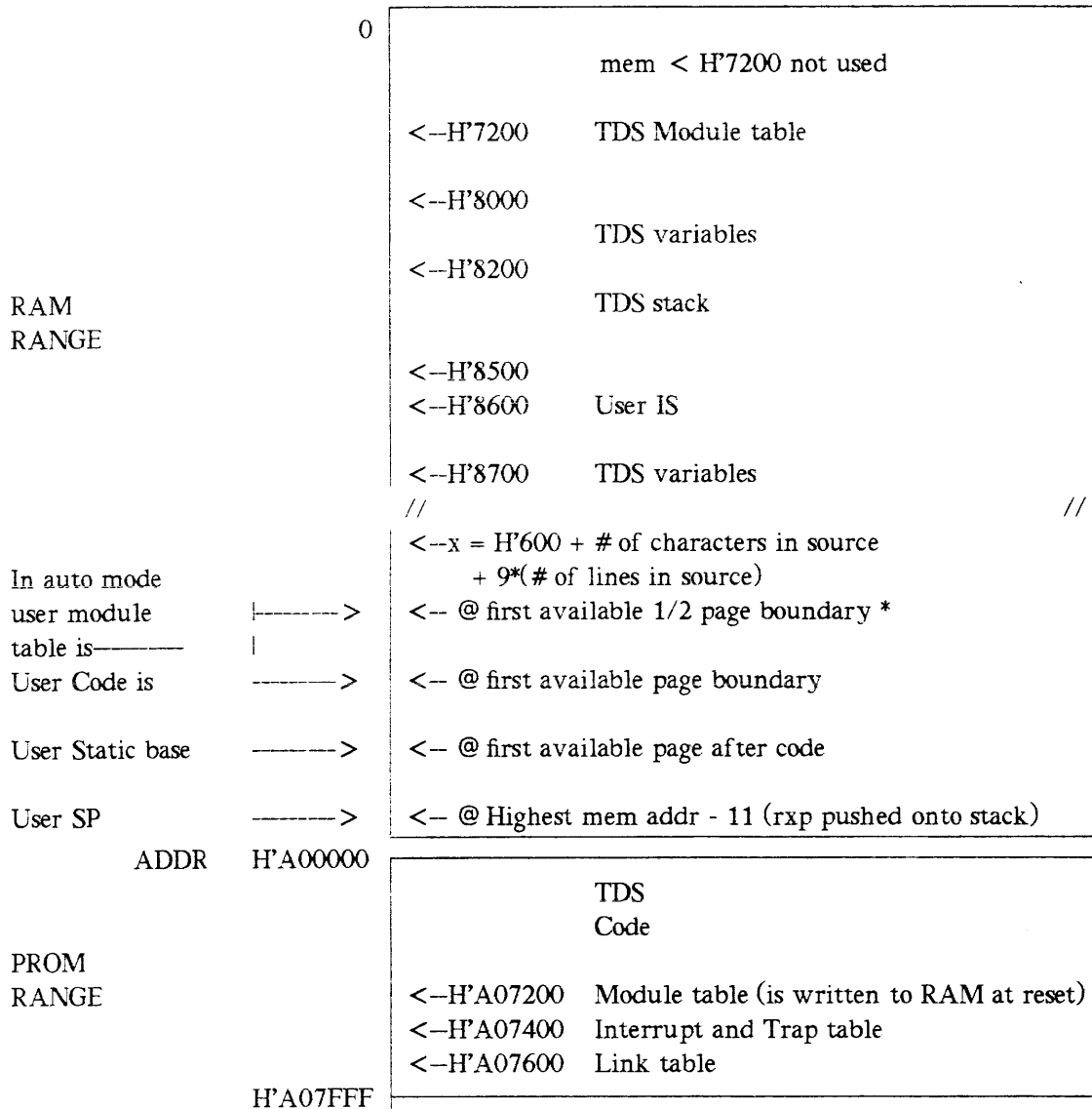
1. ASCII strings are delimited by double quotes only.
2. Symbols are valid for PC address and SB segment mode only.
3. Symbols must contain only two characters.
4. The only operation allowed is subtraction between symbols.
5. Strings are valid only for .BYTE Pseudo-op and immediate operands.
6. R0 through R7 are the only recognized register mnemonics. All others must be hex encoded.
7. CXP argument is simply n.
8. CBITI and SBITI instructions are not implemented.
9. Registers R0 through R7 must be used for F0 through F7.
10. Immediate floating-point values not allowed.
11. The STATIC area must be declared as the first element of the program.
12. Pseudo-ops parse on first four characters only.
13. All lower case characters in the string of .BYTE will be changed to upper case during assembly.
14. LXPD instruction not implemented. Use ADDR EXT(n), DESTINATION.

The assembler does not fully support operand legality-checking, value range checking, and some soft parsing. Immediate operand values are truncated and no error message is issued.

Certain parsing errors such as balanced parentheses are not checked. Characters other than those allowed by syntax rules are not checked for and may not output an error message.

C.7 TDS MEMORY MAP (DB32000)

The diagram is a graphic representation of the memory space available to TDSs.



*Unless memory available is greater than or equal to 64K, the module table is constructed within the reserved system area.

C.8 TDS ERROR SUMMARY (DB32000)

TDS system error messages are summarized as follows:

BAD_TXT	Bad text input.
ERR_ED#	Error in edit number.
BAD_MEM	Code start <i>addr</i> conflicts with text or debug data. Enter a higher value for <i>addr</i> .
BAD_SEQ	Attempting to assemble null or bad text.

BAD_INS	No such instruction.
BAD_PSU	No such Psuedo-op.
ERR_VAL	Bad number syntax or bad operand.
BAD_LIN	Gross line error or standalone label.
ERR_SEG	Using data allocation Psuedo-ops within PC segment or vice versa.
BAD_NUM	Bad floating-point syntax or value range.
BAD_SYM	Duplicate symbol.
UND_SYM	Undefined symbol.
BAD_TAB	Assembler was not initiated by IT command.
E NMI	Error - Non-maskable interrupt.
E NVI	Error - Non-vectored interrupt (not implemented).
E FPU	Error - FPU trap.
E DVZ	Error - divide by zero.
E UND	Error - undefined op-code (trying to use non-existing MMU or FPU).
E FLG	Error - flag trap.
E BPT	Error - non-debugger BPT instruction.
E ILL	Error - illegal for user instruction.
E EXT	Error - external abort.
E BPR	Error - MMU breakpoint.
E NST	Error - MMU non-sequential trace trap.
E ABT	Error - MMU address translation.
E SRC	Error - memory search failed.
E SVC	Error - unknown SVC.
E CXP	Error - more than one call command.
E VRF	Memory verify error (breakpoint could not be inserted).
TIM_OUT	No data pulses present. This message occurs approximately 90 seconds after a data operation is initiated if no data pulses are detected.
BAD_FRM	Bad tape format. Attempted to read data that does not conform to the established data format.
CRC_ERR	A CRC error was detected.

C.9 TDS COMMAND SUMMARY (DB32000)

The commands used in the TDS system are summarized as follows:

RX [H D]	Select the system base to be hexadecimal or decimal.
OM <i>gn</i>	Set the terminal communications mode. See Section 3.3.2.
IN [<i>n</i>] <i>string</i>	Insert <i>string</i> above or after line <i>n</i> of the text buffer.
TP [<i>n1</i> [/ <i>n2</i>]]	Type line <i>n1</i> of text buffer or type <i>n2</i> lines of buffer beginning with <i>n1</i> .
RP <i>n string</i>	Replace line <i>n</i> of buffer with <i>string</i> .
K <i>n</i>	Kill or delete line <i>n</i> from text buffer.
RS	Reset clear text buffer.
RL [LPT: ASN:][C]	Redirect listing from terminal to printer at parallel port or serial device at port 1, and end listing with a [CR] or cancel previously selected device.
AS [<i>addr</i>]	Begin the assembly of user program at address <i>addr</i> .
B { [Z] mod offset }	Initialize a user program if an explicit address was used as assembly time. If an explicit address was not used, <i>mod</i> and <i>offset</i> may be omitted. The Z option zeros memory from the SB to the SP area.
PM {B W D F L} { <i>addr</i> <i>symbol</i> }	Print memory contents according to selected base. Byte, Word, Double-word, Floating-point or Long.
PAD <i>symbol</i>	Print address of <i>symbol</i> .
PR {0 1 2 3 4 5 6 7}	Print the contents of one of eight general purpose registers.
Pcpureg	Print the contents of the CPU register.
PMS	Print the contents of the MMU MSR register.
PPT {0 1}	Print the contents of the MMU PTB register.
PEI	Print the contents of the MMU EADDR/INVAL register.
PPF {0 1}	Print the contents of the MMU PF register.

PSC	Print the contents of the MMU SC register.
PBP {0 ... 15 f}	Print the contents of one of 16 breakpoint registers.
PBC	Print the contents of one of the MMU BCNT registers.
PF {0 1 2 3 4 5 6 7}	Print the contents of one of eight FPU registers.
PFS	Print the contents of the FPU status register.
PCF	Print configuration.
PMM	Print the contents of the main MSF register.
AR	Print the contents of all general purpose registers.
AC	Print the contents of all CPU registers.
AM	Print the contents of all MMU registers.
AF	Print the contents of all FPU registers.
CM {B W D} {addr symbol=gn}	Change memory contents, at specified location, to an integer value.
CM {F L} {addr symbol=rn}	Change memory contents at <i>addr</i> or <i>symbol</i> to a real number.
Ccpureg = gn	Change contents of the CPU register to <i>gn</i> .
CMS = gn	Change the contents of the MSR register to <i>gn</i> .
CPT {0 1} = gn	Change the contents of the PTB register to <i>gn</i> .
CEI = gn	Change the contents of the EADR/ INVALID register to <i>gn</i> .
CSC = gn	Change the contents of the MMU SC register.
CBP {0 ... 15 f} = {addr symbol}	Change the current <i>addr</i> or <i>symbol</i> of one of sixteen breakpoint registers to <i>addr</i> or <i>symbol</i> .
CBC = gn	Change the memory contents of the MMU NCNT register to <i>gn</i> .
CF {0 1 2 3 4 5 7} = gn	Change the memory contents of one of eight FPU registers to <i>gn</i> .

CFS = <i>gn</i>	Change the memory contents of the floating-point status register to <i>gn</i> .
CCF = <i>gn</i>	Change the memory contents of the configuration register to <i>gn</i> .
CMM = <i>gn</i>	Change the memory contents of the main MSR register to <i>gn</i> .
ST [<i>n</i>]	Step through the user program for <i>n</i> instructions and then break. Default for <i>n</i> is one.
SU { <i>addr</i> <i>gpreg</i> <i>cpureg</i> <i>gn1 gn2 gnmask</i> }	Step through a user program until the contents of memory at <i>addr</i> , or <i>gpreg</i> (R0-R7), or <i>cpureg</i> masked with <i>gnmask</i> , is greater than or equal to <i>gn1</i> and less than or equal to <i>gn2</i> .
SW { <i>addr</i> <i>gpreg</i> <i>cpureg</i> } { <i>gn1 gn2 gnmask</i> }	Step through a user program while the memory contents of either <i>addr</i> , or <i>gpreg</i> , or <i>cpureg</i> masked with <i>gnmask</i> , are greater than or equal to <i>gn1</i> and less than or equal to <i>gn2</i> .
JS <i>addr</i>	Call a subroutine at location <i>addr</i> .
CX <i>mod offset</i>	Call external subroutine at address calculated using <i>mod</i> and <i>offset</i> from the module table.
GO	Begin program execution.
M <i>addr1 addr2</i>	Move <i>gn</i> sequential bytes of data from memory location <i>addr1</i> to <i>addr2</i> .
F <i>addr1 addr2 gn</i> [B W D]	Fill memory locations from <i>addr1</i> to <i>addr2</i> with data <i>gn</i> .
SR <i>addr1 addr2 gn</i> [B W D]	Search memory between <i>addr1</i> and <i>addr2</i> for data <i>gn</i> and print its address according to BWD.
D <i>addr1 gn</i>	Dump <i>gn</i> locations beginning at <i>addr1</i> , to the user terminal.
TW [<i>@n</i>] [<i>addr1 addr2</i>]	Write data between location <i>addr1</i> and <i>addr2</i> to the cassette recorder. Otherwise, write contents of editor text buffer to the cassette recorder. <i>@n</i> specifies timings for boards not equipped with a 14 MHz crystal. See Chapter 9.2 for details.

TR [*addr*]

Read data from the cassette tape and store it in memory beginning at location *addr*. Otherwise, load the data into the editor text buffer.

READER'S COMMENT FORM

In the interest of improving our documentation, National Semiconductor invites your comments on this manual.

Please restrict your comments to the documentation. Technical Support may be contacted at:

(800) 538-1866 - U.S. non CA
(800) 672-1811 - CA only
(408) 733-2600

Please rate this document according to the following categories. Include your comments below.

	EXCELLENT	GOOD	ADEQUATE	FAIR	POOR
Readability (style)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Technical Accuracy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Fulfills Needs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Presentation (format)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Depth of Coverage	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Overall Quality	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

Do you require a response? Yes No PHONE _____


Comments:

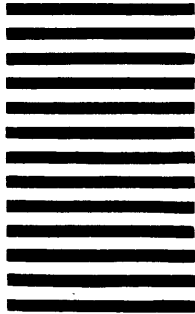


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 409 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

 **National Semiconductor Corporation**
Microcomputer Systems Division
Technical Publications Dept. 8278, M/S 7C261
2900 Semiconductor Drive
Santa Clara, CA 95051





National Semiconductor Corporation

Microcomputer Systems Division

National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051
Tel: (408) 721-5000
TWX: (910) 339-9240

National Semiconductor
5955 Airport Road
Suite 206
Mississauga, Ontario
L4V1R9 Canada
Tel: (416) 678-2920
TWX: 610-492-8863

Electronica NSC de Mexico SA
Hegel No. 153-204
Mexico 5 D.F. Mexico
Tel: (905) 531-1689, 531-0569,
531-8204
Telex: 017-73550

NS Electronics Do Brasil
Avda Brigadeiro Faria Lima 830
8 Andar
01452 Sao Paulo, Brasil
Telex: 1121008 CABINE SAO PAULO
113193 INSBR BR

National Semiconductor GmbH
Furstenriederstraße Nr.5
D-8000 München 21
West Germany
Tel.: (089) 5 60 12-0
Telex: 522772

National Semiconductor (UK), Ltd.
301 Harpur Centre
Horne Lane
Bedford MK40 1TR
United Kingdom
Tel: 0234-47147
Telex: 826 209

National Semiconductor Benelux
Ave. Charles Quint 545
B-1080 Bruxelles
Belgium
Tel: (02) 4661807
Telex: 61007

National Semiconductor (UK), Ltd.
1, Bianco Lunos Allè
DK-1868 Copenhagen V
Denmark
Tel: (01) 213211
Telex: 15179

National Semiconductor
Expansion 10000
28, Rue de la Redoute
F-92 260 Fontenay-aux-Roses
France
Tel: (01) 660-8140
Telex: 250956

National Semiconductor S.p.A.
Via Solferino 19
20121 Milano
Italy
Tel: (02) 345-2046/7/8/9
Telex: 332835

National Semiconductor AB
Box 2016
Stensåtravägen 4/11 TR
S-12702 Skärholmen
Sweden
Tel: (08) 970190
Telex: 10731

National Semiconductor
Calle Nunez Morgado 9
(Esc. Dcha. 1-A)
E-Madrid 16
Spain
Tel: (01) 733-2954/733-2958
Telex: 46133

National Semiconductor Switzerland
Alte Winterthurerstrasse 53
Postfach 567
CH-8304 Wallisellen-Zürich
Tel: (01) 830-2727
Telex: 59000

National Semiconductor
Pasilanraitio 6C
SF-00240 Helsinki 24
Finland
Tel: (90) 14 03 44
Telex: 124854

NS Japan K.K.
POB 4152 Shinjuku Center Building
1-25-1 Nishishinjuku, Shinjuku-ku
Tokyo 160, Japan
Tel: (03) 349-0811
TWX: 232-2015 NSCJ-J

National Semiconductor Hong Kong, Ltd.
1st Floor,
Cheung Kong Electronic Bldg.
4 Hing Yip Street
Kwun Tong
Kowloon, Hong Kong
Tel: 3-899235
Telex: 43866 NSEHK HX
Cable: NATSEMI HX

NS Electronics Pty. Ltd.
Cnr. Stud Rd. & Mtn. Highway
Bayswater, Victoria 3153
Australia
Tel: 03-729-6333
Telex: AA32096

National Semiconductor PTE, Ltd.
10th Floor
Pub Building, Devonshire Wing
Somerset Road
Singapore 0923
Tel: 652 700047
Telex: NATSEMI RS 21402

**National Semiconductor Far East, Ltd.
Taiwan Branch**
P.O. Box 68-332 Taipei
3rd Floor, Apollo Bldg.
No. 218-7 Chung Hsiao E. Rd.
Sec. 4 Taipei Taiwan R.O.C.
Tel: 7310393-4, 7310465-6
Telex: 22837 NSTW
Cable: NSTW TAIPEI

**National Semiconductor (HK) Ltd.
Korea Liaison Office**
6th Floor, Kunwon Bldg.
No. 2. 1-GA Mookjung-Dong
Choong-ku, Seoul, Korea
C.P.O. Box 7941 Seoul
Tel: 267-9473
Telex: K24942