

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A154 340

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NSWC TR 84-93	2. GOVT ACCESSION NO. AD-A154340	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) BP TRICOMP USER'S GUIDE	5. TYPE OF REPORT & PERIOD COVERED Final	
	6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s) The THLL Group, K53	8. CONTRACT OR GRANT NUMBER(s)	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center (Code K53) Dahlgren, VA 22448	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 36801	
11. CONTROLLING OFFICE NAME AND ADDRESS Strategic Systems Program Office Washington, DC 20376	12. REPORT DATE July 1984	
	13. NUMBER OF PAGES 85	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution is unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) TRIDENT Higher Level Language Compiler THLL program TRIDENT Digital Control Computer (TDCC) implementation VAX 11/780 runtime system		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The TRIDENT Higher Level Language (THLL) is implemented on the VAX 11/780 based TRIDENT Software Generation System (SGS) for three target machines: the TRIDENT Basic Processor (BP), the Motorola MC68000, and the VAX 11/780. This document describes implementation features of THLL, which are unique for the BP, and the runtime system for the BP.		

FOREWORD

This document was written in the Operational Support Branch (K53), Submarine Launched Ballistic Missile (SLBM) Software Development Division (K50), of the Strategic Systems Department (K) at the Naval Surface Weapons Center (NSWC), Dahlgren, Virginia.

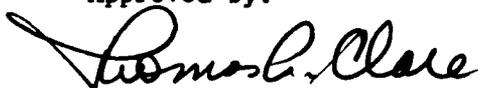
The purpose of this document is to describe implementation features of the TRIDENT Higher Level Language (THLL), which are unique for the TRIDENT BP, and the runtime support system for THLL on the BP. It supplements the THLL Reference Manual for programs designed to run on the BP.

This report is based on NSWC Technical Report TR-3657 (Revised June 1978). Much of the material in TR-3657 was rewritten and reorganized. This report was extensively reviewed by personnel in the Operational Support Software Systems Group in K53, the Quality Assurance Branch (K52), and the Operational Systems Branch (K54).

This project was funded by the Strategic Systems Program Office, Washington, DC 20376, under task number 36801.

Questions, comments, and suggestions concerning the material presented in this document should be directed to the Commander, Naval Surface Weapons Center, ATTN: K53, Dahlgren, Virginia 22448.

Approved by:



THOMAS A. CLARE, Head
Strategic Systems Department

S DTIC ELECTE **D**
APR 24 1985
B

Accession For	
NSWC	<input checked="" type="checkbox"/>
DDIC	<input type="checkbox"/>
Unpublished	<input type="checkbox"/>
Distribution	
Availability Codes	
Dist	Avail and/or Special
A-1	



CONTENTS

CHAPTER 1	INTRODUCTION	1-1
CHAPTER 2	BASIC ELEMENTS OF THLL	2-1
2.1	THLL CHARACTER SET	2-1
2.2	OPERATORS	2-2
2.3	DELIMITERS	2-2
2.4	IDENTIFIERS	2-2
2.5	CONSTANTS	2-2
2.5.1	Integer and Double Constants	2-2
2.5.2	Real Constants	2-3
2.5.3	Pointer Constants	2-3
2.5.4	Boolean Constants	2-3
2.5.5	Strings	2-3
CHAPTER 3	DATA DECLARATIONS	3-1
3.1	TYPES	3-1
3.2	BIT NUMBER CONVENTIONS	3-1
3.3	STORAGE FORMAT	3-1
3.4	ALLOCATION OF STORAGE	3-3
3.5	COMPONENTS	3-4
3.5.1	Indexed Components	3-4
3.5.2	Alpha Components	3-4
3.6	PRESETS	3-5
CHAPTER 4	EXPRESSIONS AND STATEMENTS	4-1
4.1	EXPRESSION TYPES	4-1
4.2	PROCEDURES	4-1
4.2.1	LINK and EXEC Procedures	4-1
4.2.2	Optional Arguments	4-2
CHAPTER 5	THLL I/O ON THE BP	5-1
5.1	THLL FILE CONTROL BLOCK	5-1
5.2	READ AND WRITE POSTING CODES FOR LEVELS 1 AND 2	5-2
5.2.1	KBDSS Codes	5-2
5.2.2	CPRINT and SPRINT Codes	5-3
5.2.3	MDF and MTF Codes	5-3
5.2.4	ICL Codes	5-4
5.3	I/O OPTIONS FOR READ AND WRITE PROCEDURES	5-5
5.4	READ FROM KEYBOARD CODES	5-7
5.5	READ/WRITE FORMAT ERRORS	5-7
CHAPTER 6	STANDARD PROCEDURES	6-1
6.1	FIXH, FIXI, AND FIXD FUNCTIONS	6-1
6.2	MATHEMATICAL FUNCTIONS	6-1
6.3	SHIFT FUNCTIONS	6-1
6.4	BIT FUNCTIONS	6-2
6.5	SWA FUNCTION	6-2

CHAPTER 7	UTILITY PROCEDURES	7-1
7.1	BLOCKIN	7-1
7.2	BLOCKOUT	7-2
7.3	CONVALPH	7-2
7.4	CONVINT	7-5
7.5	PRINT	7-7
7.6	EPRINT	7-7
7.7	TERM	7-8
7.8	REGLINK	7-9
7.9	TRUNC	7-9
7.10	TRUNCD	7-10
CHAPTER 8	DATA COMMUNICATION PROCESSOR PROCEDURES	8-1
8.1	START I/O PROCEDURES	8-1
8.2	TEST I/O PROCEDURES	8-2
8.3	HALT I/O PROCEDURES	8-3
8.4	RESTART I/O PROCEDURE	8-3
8.5	WRITE DIRECT PROCEDURE	8-4
8.6	READ DIRECT PROCEDURE	8-5
8.7	ACKNOWLEDGE STACK PROCEDURES	8-6
8.8	ACTIVATE/DEACTIVATE PORT PROCEDURES	8-6
8.9	RESET PROCEDURES	8-7
8.10	STATUS PROCEDURE	8-7
8.11	MITNOR PROCEDURE	8-8
8.12	ACKNOWLEDGE DEDICATED INTERRUPT PROCEDURES	8-9
CHAPTER 9	REAL TIME OPERATING SYSTEM SERVICE PROCEDURES	9-1
9.1	ATTACH PROCEDURE	9-1
9.2	SCHEDULE PROCEDURE	9-2
9.3	PURGE PROCEDURE	9-3
9.4	DETACH PROCEDURE	9-4
9.5	EXIT PROGRAM PROCEDURE	9-5
9.6	WAITE SERVICE MODULE PROCEDURES	9-6
9.6.1	MARKTIME	9-6
9.6.2	WAITMEMO, WAITKBIN, WAITICL, WAITEEXEC	9-6
9.6.3	WAITTASK	9-7
9.7	CONTINUE PROCEDURE	9-7
9.8	ENABLE, DISABLE, ARM, DISARM, AND TRIGGER PROCEDURES	9-7
9.9	OVERLAY LOADER PROCEDURE	9-8
9.10	ACQUIRE PROCEDURE	9-9
9.11	RELEASE PROCEDURES - RELEASE - ALLMEM	9-10
9.12	SETSTATUS PROCEDURE	9-10
9.13	SYSERR PROCEDURE	9-12
9.14	DATASUM PROCEDURE	9-13
9.15	EES PROCEDURE	9-13
CHAPTER 10	REFERENCES	10-1

APPENDIX A	ASCII CHARACTER SET	A-1
APPENDIX B	RUNTIME ENVIRONMENT ON THE BP	B-1
B.1	CONTROL SECTIONS	B-1
B.2	ALLOCATION OF STACK FRAMES	B-1
B.2.1	Allocation of THLLFCB	B-2
B.2.2	Temporaries	B-3
APPENDIX C	USING BP TRICOMP	C-1
C.1	INTRODUCTION	C-1
C.2	ORGANIZATION OF A THLL PROGRAM	C-1
C.3	THE BP TRICOMP ENVIRONMENT	C-2
C.4	INPUT TO BP TRICOMP	C-2
C.4.1	Compile Units (.THL Files)	C-2
C.4.2	Insert Files (.THI Files)	C-2
C.5	INVOCATION OF BP TRICOMP (.THL, .THI FILES)	C-3
C.6	OUTPUT FROM BP TRICOMP (.TLS, .BIN, .GXR, .TRE FILES)	C-3
C.7	INVOCATION OF TRILOAD (.BIN, .MAP, .LOD FILES)	C-4
C.8	RUNNING A PROGRAM (.LOD FILES)	C-4
C.9	DEBUGGING (.LIS FILES)	C-5
C.10	PRODUCING A GLOBAL CROSS REFERENCE REPORT (.GXR, .MXD, .MXR FILES)	C-5
C.11	PRODUCING A PROCEDURE CALL TREE REPORT (.TRE, .MTD, .MTR FILES)	C-6
C.12	PRODUCING A NESTED PROCEDURE CALL TREE REPORT (.TRE, .MTD, .NTR FILES)	C-6
C.13	DIAGRAM OF A THLL PROGRAM'S FILE RELATIONSHIPS	C-7
C.14	ADVANCED INVOCATIONS OF TRICOMP (OVERRIDING DEFAULTS)	C-7
C.14.1	Suppressing TRICOMP Generated Files	C-8
C.14.2	Parallel Directory Organization	C-8
C.14.3	Overriding Default File Extensions	C-8
C.14.4	Example Compile and Library Command Procedure	C-9
C.14.5	\$SEVERITY Returned from BP TRICOMP	C-9
C.15	EXAMPLE SESSION ON USING BP TRICOMP	C-9
C.15.1	Example THLL Compile Unit (DEMOMAIN.THL FILE)	C-13
C.15.2	Example THLL Compile Unit (DEMOPRIME.THL FILE)	C-14
C.15.3	Example THLL Insert File (EXTEND.THI FILE)	C-16
C.15.4	Example Global Cross Reference Report (DEMO.MXR FILE)	C-17
C.15.5	Example Procedure Call Tree Report (DEMO.MTR FILE)	C-18
C.15.6	Example Nested Procedure Call Tree Report (DEMO.NTR FILE)	C-19
DISTRIBUTION		(1)

CHAPTER 1

INTRODUCTION

BP TRICOMP is a compiler for the TRIDENT Higher Level Language (THLL) that runs on the VAX 11/780 and produces code for the TRIDENT Digital Control Computer (TDCC), also referred to as the Basic Processor (BP). This document describes implementation features of THLL, which are unique for the BP, and the runtime support system for the BP.

Differences between the BP compiler and other THLL compilers are due to differences in the hardware of the target machines and differences in the software runtime environment of the target machines. This document is not self-contained. It is to be used in combination with the THLL Reference Manual (Reference 1). Details about the BP architecture can be found in the TRIDENT Basic Processor Programmers Reference Manual (Reference 2).

BP TRICOMP allows the use of THLL as a high level programming language for the TRIDENT BP, which is included in the TRIDENT Fire Control System. It supports the full THLL language.

The BP architecture views data as 32-bit words and 64-bit doublewords. The address of the data is the word address of the first word of the data. This maps directly into the THLL view of data.

CHAPTER 2

BASIC ELEMENTS OF THLL

2.1 THLL CHARACTER SET

The THLL character set consists of the following ASCII subset:

Uppercase letters A-Z

Numerals 0-9

The following special characters:

<u>Character</u>	<u>Name</u>	<u>Character</u>	<u>Name</u>
	Space	.	Period
!	Exclamation point	/	Slash
"	Quotation mark	:	Colon
#	Number sign	;	Semicolon
\$	Dollar sign	<>	Angle brackets
%	Percent sign	=	Equal sign
&	Ampersand	?	Question mark
'	Apostrophe	@	At sign
()	Parentheses	[]	Square brackets
*	Asterisk	\	Backslash
+	Plus sign	^	Circumflex
,	Comma	_	Underline
-	Minus sign		

There is no plus or minus sign (+) defined in ASCII and the circumflex (^) is used to represent it. The TDCC devices KBDSS, CPRINT, and SPRINT display the circumflex as the + character.

Not included in the THLL character set are the lowercase letters (codes X'60' - X'7E') and the nonprintable characters (codes X'00' - X'1F' and X'7E'). When these characters are used in a THLL source file they are treated as follows:

- A. Lowercase letters are converted to uppercase letters (codes X'40' - X'5E') except when they appear in comments or remarks.

B. Nonprintable characters are converted to question marks (?).

The listing file produced by BP TRICOMP reflects these character conversions.

A table of ASCII characters is included in Appendix A.

2.2 OPERATORS

The value of LOC X is the virtual address of the THLL word or doubleword containing X. A virtual address is a 16-bit quantity of type POINTER. The upper 4 bits identify a base register, and the lower 12 bits represent a displacement.

The value of LOCA X is the absolute address of the THLL word or doubleword containing X. The type of the value is INTEGER. The bit patterns of LOC X and LOCA X are, in general, different.

2.3 DELIMITERS

All THLL delimiters are implemented in BP TRICOMP.

2.4 IDENTIFIERS

The first eight characters of identifiers are significant. The first character of the identifier must be a letter. The remaining characters can be either letters, numerals, or the special character dot (.). A dot is not allowed as the last character of an identifier. A dot that occurs in an external symbol is mapped into a dollar sign (\$).

2.5 CONSTANTS

For a pictorial representation of the layout of the following constants, see Section 3.

2.5.1 Integer and Double Constants

In BP TRICOMP, any decimal integer that exceeds 31 bits of significance is considered to be a double value. Binary, octal, and hexadecimal integers are considered to be double when they exceed 32 bits of significance.

An integer constant is kept in one THLL word (32-bit BP word). A double constant is kept in a THLL doubleword (two 32-bit BP words).

2.5.2 Real Constants

BP TRICOMP uses the 64-bit BP floating point format for THLL real numbers (see Reference 2). The range for real constants on the BP is approximately .353E-9864 through .708E+9864. The precision for real constants on the BP is approximately 15 decimal digits.

2.5.3 Pointer Constants

The only legal pointer constant is 0. Pointer expressions built up using THLL operators are valid at preset time and runtime. A pointer contains the BP virtual address of the designated item. A virtual address on the BP is specified in 16 bits. The upper 4 bits designate the base register, and the lower 12 bits designate the displacement. Therefore, pointer components require at least 16 bits on the BP.

2.5.4 Boolean Constants

TRUE is defined to be a 32-bit integer that has all 32 bits set; however, any non-zero value tests as true. FALSE is represented as X'00000000'.

2.5.5 Strings

A string is represented by a header word followed by a block of words holding the characters of the string. The characters are ASCII as defined for the BP. Each character occupies one 8-bit byte. The characters are packed from left to right within a THLL word.

CHAPTER 3

DATA DECLARATIONS

3.1 TYPES

There are six types for classifying constants, variables, and procedures in THLL. A type may be thought of as a class of values. In general, the user should only have to know about the abstract properties of a type, the rules for the use of data types by operators and procedures, and the rules for type conversion. For portability reasons, the user should make a conscious effort to not take advantage of a particular implementation of types. In some cases, however, it may be necessary to know about the internal representation of the various types of values in memory.

Constants are assigned types as specified in Sections 2.5.1 through 2.5.5. Symbol types are specified in the corresponding declarations. Valueless items, such as statements, are assigned no type (N).

3.2 BIT NUMBER CONVENTIONS

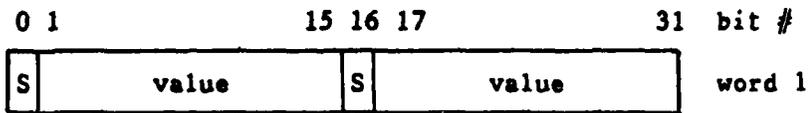
THLL bit numbers run from 0 for the leftmost (most significant) bit to 31 for the rightmost (least significant) bit of a THLL word. For a doubleword the bit numbers run from 0 to 63.

This convention is used for defining the meaning of a component "start bit" and "number of bits." Functions such as TEST.BIT, CLR.BIT, SET.BIT, TGL.BIT, and FIND.BIT use the THLL bit number as an input parameter, and FIND.BIT can also output a THLL bit number.

3.3 STORAGE FORMAT

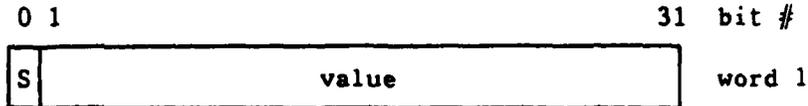
The types INTEGER (I), DOUBLE (D), and REAL (R) have a fixed format as to number of bits, location of sign bit, and number of words required for storage.

The type HALF (H) is equivalent to the type INTEGER for all data except arrays. The elements of a HALF array require only 16 bits of storage. Hence, two such type values from an array can be stored in one word.

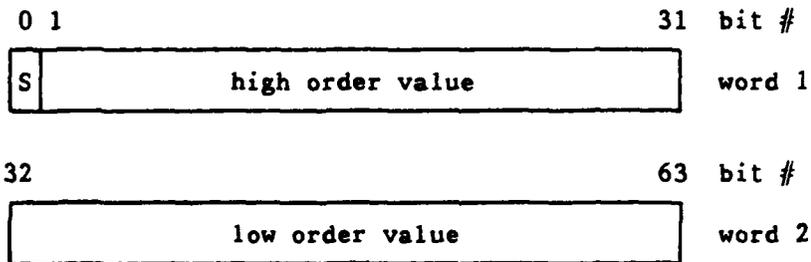


S = sign bit

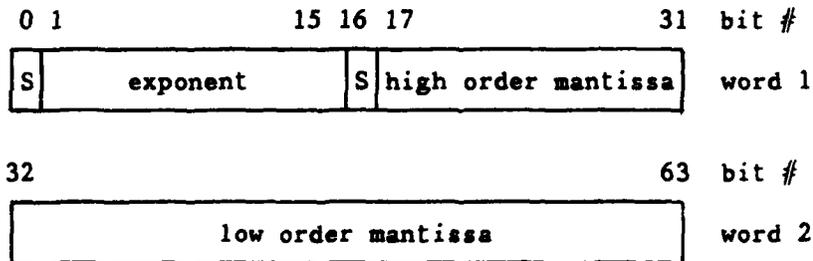
Type INTEGER requires one word, 32 bits.



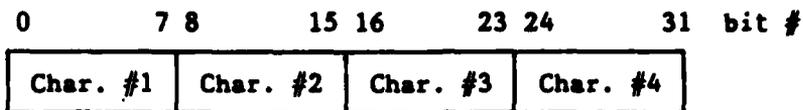
Type DOUBLE requires two words, 64 bits.



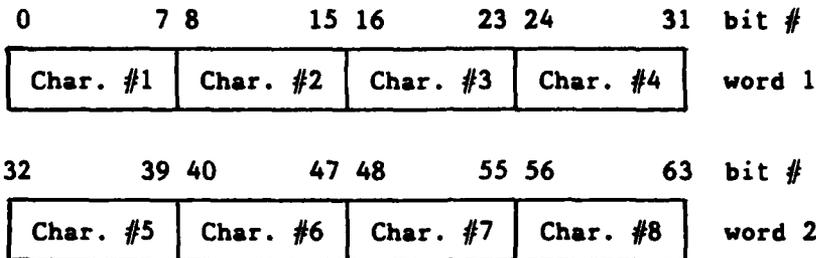
Type REAL requires two words, 64 bits, separated into exponent and mantissa fields. The exponent and the mantissa are both two's complement values. For more information, see Reference 2.



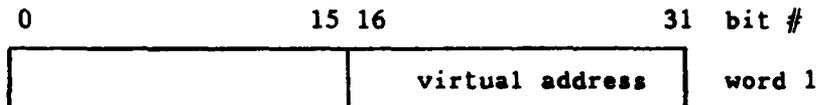
Type ALPHA is stored as four characters per word plus one header word. Thus, the number of words required for a string of n characters is $((n+3)/4) + 1$. However, the programmer does not need to allow for the header when specifying string length.



Eight characters of a string can also be stored in a variable of type DOUBLE.



Type POINTER is stored as a full word with the value in the least significant 16 bits.



3.4 ALLOCATION OF STORAGE

Storage is allocated in the order in which data are declared. All THLL data structures are allocated at THLL word or doubleword addresses. A doubleword address is one which is a multiple of 2. All data structures of type DOUBLE or REAL and all stacks and commons are allocated at doubleword addresses. All other data structures are allocated at word addresses. All constants are allocated at word addresses.

If in allocating a doubleword address, a "hole" arises, the "hole" remains. No attempt is made to fill the hole.

The allocation of storage within a common is the same as the allocation of storage outside the common.

The constant area, the OWN area, and the runtime stack all begin at doubleword addresses. This guarantees that REAL and DOUBLE data structures, as well as stacks and commons, start at doubleword addresses.

3.5 COMPONENTS

THLL components are based on the concept of THLL words. The offset part of the component declaration is in units of THLL words. When an integer expression is added to the pointer in a component reference, it is also considered to be in units of THLL words.

POINTER components require at least 16 bits.

REAL and DOUBLE components reference THLL doublewords. However, they do not need to be aligned at doubleword addresses. The offset in the component declaration is in units of THLL words. Thus, a DOUBLE or REAL component occupies two offset values.

3.5.1 Indexed Components

Full word INTEGER and POINTER components index in units of THLL words, and DOUBLE and REAL components index in units of THLL doublewords.

Partial word indexed components run with increasing index from left to right. For portability reasons, characters in ALPHA or DOUBLE variables should not be accessed with INTEGER indexed components. Instead, ALPHA components should be used.

3.5.2 Alpha Components

The ALPHA component provides a machine-independent method of accessing character data. In effect, it is an indexed INTEGER component that indexes in the same direction as character data on the target computer. The type of the value of an alpha component is INTEGER, not ALPHA!

An ALPHA component variable must always be used as an indexed component variable. The OFFSET information in the component declaration is in units of THLL words. A particular character within the THLL word can be accessed by using an appropriate character number as in the example below.

```
ALPHA COMPONENT CHR ;
OFFSET 0 FOR CHR ; /* OPTIONAL - 0 IF NOT SPECIFIED */
or
ALPHA COMPONENT CHR(OFFSET 0) ;

usage: CHR(P,I) /* P - POINTER TO FIRST WORD THAT
                CONTAINS CHARACTER DATA */
                /* I - CHARACTER NUMBER */
```

Note that the following two declarations are equivalent in BP TRICOMP:

```
ALPHA COMPONENT CHAR (OFFSET 0) ;  
INTEGER COMPONENT BYTE (FIELD(0,8),OFFSET 0) ;
```

The two component variables CHAR(P,I) and BYTE(P,I) always have the same value on the BP. However, the second method is not recommended.

3.6 PRESETS

Compile time expressions evaluated by BP TRICOMP are the same as runtime expression evaluation on the BP. Presets using indexed components work the same as runtime indexed components.

CHAPTER 4

EXPRESSIONS AND STATEMENTS

4.1 EXPRESSION TYPES

A POINTER value is the virtual address of a 32-bit THLL word. A half value is a 16-bit quantity if the expression is a subscripted array variable, a 32-bit quantity otherwise. Therefore, since two half array elements fit into the same 32-bit word, both elements have the same virtual address.

4.2 PROCEDURES

A THLL procedure can be called from another THLL procedure, an assembly program, or the operating system. A THLL procedure can call itself, other THLL procedures, and assembly programs.

THLL allows recursion. THLL procedures running on the BP can be executed in a recursive manner. Variables sensitive to each invocation of a procedure must be shared (non-OWN).

A runtime interrupt occurs when a procedure references a missing parameter passed by reference. This can happen when required actual parameters are missing in the procedure call.

A subscripted array variable of type HALF can only be passed by value.

Appendix B contains details about the runtime environment on the BP. Appendix C shows a method for preparing programs for the BP.

4.2.1 LINK and EXEC Procedures

LINK and EXEC procedures are assumed to be GLOBAL. It is not considered an error if they are also declared to be GLOBAL.

The entry point of a program must be a LINK procedure. Any procedure that may be invoked from a different virtual space must be a LINK procedure. See Sections 6.1, 6.6, and 7.1.5 of the THLL Reference Manual (Reference 1) for complete information on LINK and EXEC procedures.

4.2.2 Optional Arguments

ARGSYNCL is used to determine the syntactic class of the I'th argument of a call to a procedure using optional arguments. More information about ARGSYNCL may be found in the THLL Reference Manual (Reference 1). For the BP, the meaning of the integer value returned by ARGSYNCL is:

<u>Argument Class</u>	<u>Value Returned</u>
one-dimensional array	1
two-dimensional array	2
three-dimensional array	3
simple variable	0
stack	4
procedure	0
device	0
format	0

CHAPTER 5

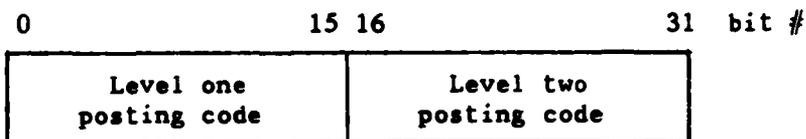
THLL I/O ON THE BP

5.1 THLL FILE CONTROL BLOCK

The THLL runtime system on the BP maintains a THLL File Control Block (THLLFCB) that contains all information concerning the status of the file and the most recent READ/WRITE transfer involving the file. OPEN, CLOSE, READ, and WRITE have an effect on the THLLFCB.

The first word of the THLLFCB contains the level one and two posting codes. The devices for which this code is meaningful are the Magnetic Disk File (MDF) and Magnetic Tape File (MTF). The format of the posting areas and the meaning of these codes for the OPEN procedure is as follows:

THLLFCB



Level one
Posting codes

0

Meaning

Not Used

Level two
Posting codes

0

Meaning

File open to the user

-1

File not found

It is the programmer's responsibility to check posting returned from the OPEN procedure.

The CLOSE procedure releases the THLLFCB for the file being closed. After return from CLOSE(P), the pointer P no longer points to a valid data structure that represents a runtime maintained THLLFCB.

5.2 READ AND WRITE POSTING CODES FOR LEVELS 1 AND 2

The first word of the THLLFCB contains the level one and level two posting codes. Level one is found in bits 0-15; level two is found in bits 16-31. The meanings of these codes are found in the following sections.

5.2.1 KBDSS Codes

<u>Level one Posting codes</u>	<u>Meaning for KBDSS</u>
0	Parameters Valid
-3	Currently Locked
-4	Currently Busy
-5	Not Connected
-53	Output Not Accepted by DCP
<u>Level two Posting codes</u>	<u>Meaning for KBDSS</u>
+1	Processing Completed
0	Processing Incomplete
-3	Has Been Locked
-5	Has Been Disconnected
-6	I/O Timeout
-7	Output Failure
-7	Input Failure
-14	Invalid Alert Interrupt
-53	Output Retry Not Accepted by the DCP
-53	Input Request Not Accepted by the DCP

5.2.2 CPRINT and SPRINT Codes

<u>Level one Posting codes</u>	<u>Meaning for CPRINT and SPRINT</u>
0	All Parameters Valid
-3	Device Locked
-51	No Memory Available
-52	Device Queue Full
<u>Level two Posting codes</u>	<u>Meaning for CPRINT and SPRINT</u>
1	Processing Completed
0	Processing Incomplete

5.2.3 MDF and MTF Codes

<u>Level one Posting codes</u>	<u>Meaning for MDF and MTF</u>
0	Parameters Valid
-3	Device Locked
-5	Device Not Connected
-8	Device Not Normal
-13	Protect Area/Key in Protect
-51	No Memory Available
-52	Device Queue Full
-53	Command Not Accepted by DCP
-54	Disk Down (MDF)/Drive In-op or Illegal Operation (MTF)

<u>Level two Posting codes</u>	<u>Meaning for MDF and MTF</u>
1	Processing Completed
0	Processing Incomplete
-5	Device Not Connected
-6	Device Time Out
-7	Unusual End
-8	Device Not Normal
-10, -54	Disk Down (MDF)/Drive In-op or Illegal Operation (MTF)
-11	Header Failure
-12	Buffer Length Error
-13	Protect Area/Key in Protect
-53	Command Not Accepted

5.2.4 ICL Codes

<u>Level one Posting codes</u>	<u>Meaning for ICL</u>
0	Parameters Valid
-3	ICL Locked
-4	ICL Software (in this computer) is Busy
-4	ICL Software (in the other computer) is Busy
-6	I/O Time Out
-6	Unusual End Encountered
-53	Command Not Accepted by DCP

<u>Level two Posting codes</u>	<u>Meaning for ICL</u>
+1	Process Completed
0	Process Incomplete
-6	I/O Time Out in Sender Task
-6	I/O Time Out in Receiver Task
-7	Unusual End Encountered for Sender Task
-7	Unusual End Encountered for Receiver Task
-9	Disk Failure (Sender)
-9	Disk Failure (Receiver)
-15	Invalid Parameter Table (Sender Task)
-15	Sumcheck Failure
-15	Invalid Parameter Table (Receiver Task)
-50	Memory Not Available (Sender)
-50	Memory Not Available (Receiver)
-53	Sender Command Not Accepted by DCP
-53	Receiver Command Not Accepted by DCP

5.3 I/O OPTIONS FOR READ AND WRITE PROCEDURES

The OPT expression (see the description of READ and WRITE in Reference 1) is used to define certain I/O options in a bit-wise manner. The meaning of each bit is defined as follows:

<u>Bit</u>	<u>Meaning</u>
31 (Least significant bit)	NOWAIT: waiting for an I/O operation to complete is not performed after the initiation. Not applicable to ICL.
30 (for MDF)	NOCOMP: there is no comparison of the transferred data.

- 29 (for MTF) NOBCKSP: the device is not backspaced before the I/O operation.
- 29 (for CPRINT or SPRINT) FORMFEED: a form feed is issued before title and message.
- 28 (for CPRINT or SPRINT) Periodic print is used--entire message is queued.

The following codes are unique to the KBDSS.

- 28 OLDPGE: the old buffer is rewritten.
- 27 Line Select 1 button is enabled.
- 26 Line Select 2 button is enabled.
- 25 Line Select 3 button is enabled.
- 24 Line Select 4 button is enabled.
- 23 Line Select 5 button is enabled.
- 22 Line Select 6 button is enabled.
- 21 Line Select 7 button is enabled.
- 20 Line Select 8 button is enabled.
- 19 Spare, always input 0.
- 18 Advance page button is enabled.
- 17 Back page button is enabled.
- 16 Data Entry button is enabled.
- 15 Initialize button is enabled.
- 14 Spare, always input 0.
- 13 Execute button is enabled
- 12 Information/Action display. The current write is for an information display or operator action.

5.4 READ FROM KEYBOARD CODES

When reading from the keyboard, the following codes are returned in the third word of the THLLFCB.

<u>Code</u>	<u>Meaning</u>
0	No buttons pressed
1	Line Select 1 pressed
2	Line Select 2 pressed
3	Line Select 3 pressed
4	Line Select 4 pressed
5	Line Select 5 pressed
6	Line Select 6 pressed
7	Line Select 7 pressed
8	Line Select 8 pressed
10	Advance page
11	Back page pressed
12	Execute pressed
13	Initialize pressed

5.5 READ/WRITE FORMAT ERRORS

When a format error occurs during a read, an error code is returned in the fourth word of the THLLFCB. The meaning of this code follows:

<u>Error Code</u>	<u>Meaning</u>
-N	The N'th item in the expanded format contains an improper value.
0	No errors.

During a write, if the converted value is too large for the field specified in the format statement, the field is filled with asterisks.

CHAPTER 6

STANDARD PROCEDURES

All standard procedures described in the THLL Reference Manual (Reference 1) are available on the BP.

Most of the standard functions and a number of operators such as `**` are implemented as routines in the THLL runtime library, SYSLIB. Appendix C shows how to access this library.

The following sections describe BP machine dependencies for the standard procedures.

6.1 FIXH, FIXI, AND FIXD FUNCTIONS

The value returned is the greatest integer less than or equal to the argument. Thus $\text{FIXI}(-4.5) = -5$ and $\text{FIXI}(4.5) = 4$. This is also the case when the compiler automatically generates a REAL to INTEGER type conversion.

6.2 MATHEMATICAL FUNCTIONS

The mathematical routines such as the trigonometric functions, SQRT, LN, EXP, etc. are supported either by BP instructions or by the runtime library, SYSLIB. If the arguments of these functions are not in the required THLL range, the BP illegal operand fault is set.

The argument for SQRT must be greater than or equal to 0. The argument of the trigonometric functions must be radians in the range $-\pi$ to π . The argument of ARCSIN and ARCCOS must be in the range -1 to 1. The argument of LN must be greater than 0. The angle argument for POCA, ROAX, and ROTA must be radians in the range $-\pi$ to π .

6.3 SHIFT FUNCTIONS

A 16-bit operand of type HALF is extended to a 32-bit INTEGER quantity and then shifted.

6.4 BIT FUNCTIONS

If the bit number argument for any of the bit functions (TEST.BIT, CLR.BIT, etc.) is not in the range 0 to 31, the nonexistent bit fault is set.

6.5 SWA FUNCTION

The argument N is an integer. The function value is N considered as a pointer with base register $N/(2^{**}12)$ and displacement $N \text{ MOD } 2^{**}12$.

CHAPTER 7
UTILITY PROCEDURES

Each utility procedure that is used must be declared external. The form for that external declaration appears with each procedure. The effect, restrictions, and applications are briefly defined. These procedures can be found in the THLL runtime library, SYSLIB. Appendix C shows how to access this library.

7.1 BLOCKIN

BLOCKIN is a procedure which calls the RTOS MONITOR to read a buffer of specified length into a designated data area of the user's program.

The form of the procedure call is:

```
BLOCKIN(DEV,NAME,PTR,POST,SIZE,INDEX)
```

where

DEV - device identifier.

NAME - a character string of one to eight characters or a double variable naming the file to be read.

PTR - pointer to data area to receive data.

POST - pointer to a posting buffer of at least 16 words.

SIZE - integer expression specifying number of words to be read.

INDEX - integer expression specifying file index to be read.

The device argument is restricted to identifiers assigned to the MDF, MTF, or ICL. It is not necessary to open the device used in this procedure.

The external declaration is:

```
EXTERNAL PROCEDURE BLOCKIN(DEVICE,DOUBLE VALUE,POINTER VALUE,  
POINTER VALUE,INTEGER VALUE,INTEGER VALUE) ;
```

See Sections 5.2, 5.2.3, and 5.2.4 for posting codes.

7.2 BLOCKOUT

BLOCKOUT is a procedure which calls the RTOS MONITOR to write a buffer of specified length from a designated area of the user's program.

The form of the procedure call is:

```
BLOCKOUT(DEV,NAME,PTR,POST,SIZE,INDEX)
```

where

- DEV - device identifier.
- NAME - a character string of one to eight characters or a double variable naming the file to receive the data.
- PTR - pointer to data area to be written.
- POST - pointer to a posting buffer of at least 16 words.
- SIZE - integer expression specifying number of words to be written.
- INDEX - integer expression specifying file index to be written.

The device argument is restricted to identifiers assigned to the MDF, MTF, or ICL. It is not necessary to open the device used in this procedure.

The external declaration is:

```
EXTERNAL PROCEDURE BLOCKOUT(DEVICE,DOUBLE VALUE,POINTER VALUE,  
POINTER VALUE,INTEGER VALUE,INTEGER VALUE) ;
```

See Sections 5.2, 5.2.3, and 5.2.4 for posting codes.

7.3 CONVALPH

CONVALPH is an integer procedure which converts a substring of an ALPHA variable to an integer. This integer is returned as the value of the procedure unless the substring is not convertible. A substring is convertible if it contains only characters 0 through 9. The procedure allows the THLL programmer to designate the character position P of the ALPHA string on which to start the conversion and the length L of the ALPHA string to be converted. If the position is not specified, then 0 is assumed. If the length is not specified, then the current length of the entire ALPHA string minus P is

assumed. If P+L exceeds the current length of the ALPHA variable, the string is not convertible.

The form of the procedure call is:

```
CONVALPH(NAME[,P[,L]])
```

where

NAME - the identifier of the ALPHA string or a pointer to the ALPHA.

P - the start character position within the ALPHA at which to begin conversion (first position is 0).

L - the number of characters to be converted.

The returned value of the procedure is:

<u>Returned Value</u>	<u>Condition</u>
desired integer	the ALPHA is convertible
-1	the ALPHA is not convertible

The external declaration is:

```
EXTERNAL INTEGER PROCEDURE CONVALPH(ALPHA,OPTARG) ;
```

For details on OPTARG see Reference 1.

Examples of CONVALPH:

A. If the declaration is:

```
ALPHA NAME(13) ;
```

and the assignment is:

```
NAME-#/NUMBER IS 1234/ ;
```

then CONVALPH(NAME,10,4) returns 1234.

B. If the declaration is:

ALPHA NUMB(3) ;

and the assignment is:

NUMB-#/1234/ ;

then CONVALPH(NUMB) returns 1234.

C. If the declaration is:

ALPHA MESSAGE(14) ;

and the assignment is:

MESSAGE-#/HEX VALUE IS D8/ ;

then CONVALPH(MESSAGE,13,2) returns -1 because D8 is not convertible.

D. If the declaration is:

ALPHA B(3) ;

and the assignment is:

B-#/-250/ ;

then CONVALPH(B) returns -1 because the negative sign is not convertible.

E. If the declaration is:

ALPHA B(15) ;

and the assignment is:

B-#/SPEED IS 250 MPH/ ;

then CONVALPH(B,9,3) returns 250.

7.4 CONVINT

CONVINT is a procedure which converts an integer INT into an ALPHA string of a specified length L and substitutes it into a given ALPHA variable starting at a specified position P. Only non-negative integers are convertible. If the position P is not specified, then 0 is assumed. If the length is not specified, then the current length of the ALPHA variable minus P is assumed. If P+L exceeds the current length of the ALPHA variable, the number is not convertible.

The string produced from INT is defined as follows:

If $INT < 0$ or $INT \geq 10^{**}L$, then the string is a sequence of L asterisks:

`#!/*...*/`

Otherwise, the string is a sequence of L characters consisting of the right-justified decimal digits resulting from the conversion. If necessary, the field is padded with leading zeros.

`#/0 ... 0 dk ... d1 d0/`,

The d_i are the decimal digits of the number:

$$INT = d_0 + d_1 * 10 + \dots + d_k * 10^{**}k$$

The form of the procedure call is:

`CONVINT(INT,NAME[,P[,L]])`

where

INT - the integer to be converted.

NAME - the identifier of the ALPHA variable or a pointer to the ALPHA.

P - the start character position within the ALPHA.

L - the number of characters into which the integer is to be converted.

The returned value of the procedure is:

<u>Returned Value</u>	<u>Condition</u>
0	the integer is not convertible or the string produced is <code>#!/*...*/</code>
-1	the integer is convertible

The external declaration is:

EXTERNAL INTEGER PROCEDURE CONVINT(INTEGER VALUE,ALPHA,OPTARG) ;

For details on OPTARG see Reference 1.

Examples of CONVINT. (b indicates a blank)

A. If the declarations are:

INTEGER INT ;
ALPHA NOTE(8) ;

and the assignments are:

INT = 20 ;
NOTE = #/MSLbNObbb/ ;

then CONVINT(INT,NOTE,7,2) produces NOTE = #/MSLbNOb20/

B. If the declarations are:

INTEGER INT ;
ALPHA NOTE(5) ;

and the assignments are:

INT = 10 ;
NOTE = #/bbbbbb/ ;

then CONVINT(INT,NOTE) produces NOTE = #/000010/

C. If the declarations are:

INTEGER INT ;
ALPHA N(8) ;

and the assignments are:

INT = -250 ;
N = #/bbbbbbbbb/ ;

then CONVINT(INT,N) produces N = #/*****/

D. If the declarations are:

```
INTEGER INT ;
ALPHA M(8) ;
```

and the assignments are:

```
INT = 500 ;
M = #/bb/ ;
```

then CONVINT(INT,M) produces M = #/**/

7.5 PRINT

PRINT calls an RTOS MONITOR service routine to print a message on a designated device. A wait option is used.

The form of the procedure call is:

```
PRINT(DEV,POST,MSG)
```

where

DEV - device identifier.

POST - pointer to a posting buffer of at least 16 words.

MSG - the ALPHA string to be printed or a pointer to the ALPHA.

The device argument is restricted to identifiers assigned to either the system printer SPRINT or the computer printer CPRINT. The programmer is responsible for ensuring that the ALPHA string produces the desired appearance (Reference 4).

The external declaration is:

```
EXTERNAL PROCEDURE PRINT(DEVICE, POINTER VALUE, ALPHA) ;
```

See Sections 5.2 and 5.2.2 for posting codes.

7.6 EPRINT

EPRINT calls an RTOS MONITOR service routine to print a message on a designated device. The no-wait option is used.

The form of the procedure call is:

```
EPRINT(DEV,POST,MSG)
```

where

DEV - device identifier.

POST - pointer to a posting buffer of at least 16 words.

MSG - the ALPHA string to be printed or a pointer to the ALPHA.

The device argument is restricted to identifiers assigned to either the system printer SPRINT or the computer printer CPRINT. The programmer is responsible for ensuring that the ALPHA string produces the desired appearance (Reference 4).

The external declaration is:

```
EXTERNAL PROCEDURE EPRINT(DEVICE, POINTER VALUE, ALPHA) ;
```

See Sections 5.2 and 5.2.2 for posting codes.

7.7 TERM

TERM calls the RTOS MONITOR periodic print routines to print a message on a designated device.

The form of the procedure call is:

```
TERM(DEV,POST,MSG)
```

where

DEV - device identifier.

POST - pointer to a posting buffer of at least 16 words.

MSG - the ALPHA string to be printed or a pointer to the ALPHA.

The device argument is restricted to identifiers assigned to either the system printer SPRINT or the computer printer CPRINT. The ALPHA string must be less than 160 characters in length, including printer control characters (Reference 4).

The external declaration is:

EXTERNAL PROCEDURE TERM(DEVICE, POINTER VALUE, ALPHA) ;

See Sections 5.2 and 5.2.2 for posting codes.

7.8 REGLINK

REGLINK is an integer procedure whose value is the contents of the specified register at the time the calling procedure is entered.

The form of the procedure call is:

REGLINK(N)

where

N - integer expression designating the general register (0-15).

The use of this procedure is restricted to EXEC procedures to obtain the values contained in the registers when the procedure is called.

The external declaration is:

EXTERNAL INTEGER PROCEDURE REGLINK(INTEGER VALUE) ;

7.9 TRUNC

TRUNC is an integer procedure that can be used to obtain the singleword representation of the integer portion of a real expression. For example, TRUNC(2.5) is 2, and TRUNC(-2.5) is -2. This is in contrast to the FIX functions described in Section 6.1.

The form of the procedure call is:

TRUNC(R)

where

R - any real valued expression

It should be noted that no attempt is made to determine whether the value of the integer obtained can fit into a singleword.

The external declaration is:

```
EXTERNAL INTEGER PROCEDURE TRUNC(REAL VALUE) ;
```

7.10 TRUNCD

TRUNCD is a double procedure that can be used to obtain the doubleword representation of the integer portion of a real expression.

The form of the procedure call is:

```
TRUNCD(R)
```

where

R - any real valued expression

The external declaration is:

```
EXTERNAL DOUBLE PROCEDURE TRUNCD(REAL VALUE) ;
```

CHAPTER 8

DATA COMMUNICATION PROCESSOR PROCEDURES

A privileged program is able to communicate with the external equipment through the interface provided by the Data Communication Processor (DCP). The following procedures aid the THLL programmer in writing privileged state programs by translating DCP-related I/O requests into DCC I/O instructions (Reference 3). Additional information on the DCP is also available in the DCP Protocol document (Reference 4). The DCP related procedures are described in the following sections. They can be found in the THLL runtime library, SYSLIB. Appendix C shows how to access this library.

All DCP procedures are integer procedures. They return as value the Condition Designator (CD) which is a 4-bit quantity. These bits are right-justified (bits 28 through 31) in the returned value. Only the CD bits that have meaning are described in the sections that follow.

8.1 START I/O PROCEDURES

The SIO1 and SIO2 integer procedures enable the THLL programmer to initiate port activity. In particular, SIO1 initiates I/O on a single port. SIO2 initiates I/O on a port pair. The forms for these procedure calls are:

```
SIO1(DCW,PORT)
SIO2(DCW,PORT)
```

where

DCW - pointer to an array, stack, or subscripted variable which contains the Data Control Words (DCWs) that indicate the activity to be performed.

PORT - integer expression specifying the port or port pair on which I/O activity is to be initiated.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>SIO1 CD Meaning</u>
30	If set, port number is in DCW stack
28	If set, port is busy
<u>Bit</u>	<u>SIO2 CD Meaning</u>
28	If set, at least one port of pair is busy

8.2 TEST I/O PROCEDURES

The TIO1 and TIO2 integer procedures enable the THLL programmer to test the busy status of a single port (using TIO1) or port pair (using TIO2). The forms of the procedure calls are:

TIO1(PORT)
TIO2(PORT)

where

PORT - integer expression specifying the port or port pair being tested.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>TIO1 CD Meaning</u>
30	If set, port number is in DCW stack
29	If set, interrupt on specified port is pending
28	If set, port is busy
<u>Bit</u>	<u>TIO2 CD Meaning</u>
29	If set, output port is busy
28	If set, input port is busy

8.3 HALT I/O PROCEDURES

The HIO1 and HIO2 integer procedures enable the THLL programmer to halt I/O activity between the DCP and the DCC on either a single port or a port pair. HIO1 is used for a single port; HIO2 is used for a port pair. The forms of the procedure calls are:

```
HIO1(PORT)
HIO2(PORT)
```

where

PORT - integer expression which specified the port or port pair on which activity is to be halted.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>HIO1 CD Meaning</u>
30	If set, port number is in DCW stack
28	If set, port is busy
<u>Bit</u>	<u>HIO2 CD Meaning</u>
29	If set, output port is busy
28	If set, input port is busy

8.4 RESTART I/O PROCEDURE

The RSIO integer procedure enables the programmer to continue I/O transfers that have been halted. The form of the RSIO procedure call is:

```
RSIO(PORT)
```

where

PORT - integer expression which specifies the port that is to be restarted.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>RSIO CD Meaning</u>
28	If set, port is busy

8.5 WRITE DIRECT PROCEDURE

The DOIO integer procedure enables the THLL programmer to transfer information from the CPU controlled Executive Interface directly to a peripheral. The form of the procedure call is:

DOIO(EXP,PORT,TAG)

where

- EXP - integer expression whose value is sent to the peripheral.
- PORT - integer expression that specifies the output port over which the transfer is made.
- TAG - integer expression that represents the type of transfer to be performed.

The allowable TAG values and their interpretations are listed below:

<u>Value</u>	<u>Meaning</u>
0	Select Command Transfer (without interrupt)
1	Select Data Transfer (without interrupt)
2	Override Command Transfer (without interrupt)
3	Error
4	Select Command Transfer (with interrupt)
5	Select Data Transfer (with interrupt)
6	Override Command Transfer (with interrupt)
7	Error

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>DOIO CD Meaning</u>
29	If set, TAG error detected
28	If set, port is busy

8.6 READ DIRECT PROCEDURE

The DIIO integer procedure enables the THLL programmer to transfer information from a peripheral directly into the CPU controlled Executive Interface. The form of the DIIO procedure call is:

DIIO(RTN,PORT,TAG)

where

- RTN - integer variable to contain the peripheral information.
- PORT - integer expression that specifies the input port over which the transfer is made.
- TAG - integer expression that represents the type of transfer to be performed.

The allowable values for TAG and their meanings are:

<u>Value</u>	<u>Meaning</u>
0	Without interrupt
1	With interrupt

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>DIIO CD Meaning</u>
28	If set, port is active, waiting for input transfer

8.7 ACKNOWLEDGE STACK PROCEDURES

The ASTK1, ASTK2, and ASTK3 integer procedures enable the THLL programmer to retrieve interrupt information that is held in the DCP stacks. In particular, ASTK1 acknowledges DCP interrupt stack one, ASTK2 acknowledges DCP interrupt stack two, and ASTK3 acknowledges DCP interrupt stack three.

The forms of the procedure calls are:

```
ASTK1(INTFOR)
ASTK2(INTFOR)
ASTK3(INTFOR)
```

where

INTFOR - integer variable to contain the interrupt information that is to be retrieved.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>CD Meaning</u>
30	If set, at least one interrupt is pending
29	If set, more than one interrupt is pending
28	Should be set to indicate port is busy

8.8 ACTIVATE/DEACTIVATE PORT PROCEDURES

The APORT and DPORT integer procedures enable the THLL programmer to activate (using APORT) or deactivate (using DPORT) a specific port. The forms of these procedure calls are:

```
APORT(PORT)
DPORT(PORT)
```

where

PORT - integer expression which specifies the port to be activated or deactivated.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>CD Meaning</u>
30	If set, interrupt is pending
28	Should be set to indicate port is busy

8.9 RESET PROCEDURES

The RESETCIU, RESETDCP, and RESET integer procedures enable the THLL programmer to initialize the CIU, initialize the DCP, or reset a port, respectively. The forms of these procedure calls are:

```
RESETCIU
RESETDCP
RESET(PORT)
```

where

PORT - integer expression which specifies the port to be reset.

The returned values for RESETCIU and RESETDCP are meaningless. The meaning of the CD bits returned by RESET is as follows:

<u>Bit</u>	<u>RESET CD Meaning</u>
30	Port is in DCW stack for RESET call
29	Error, if set
28	Indicates port is busy if set on RESET call

8.10 STATUS PROCEDURE

The STATUS integer procedure enables the THLL programmer to retrieve the status indications that are internal to the DCP. The form of the procedure call is:

```
STATUS(PORT, TAG, STAFOR)
```

where

- PORT - integer expression which specifies the port from which the status is to be retrieved.
- TAG - integer expression which indicates the type of information to be returned.
- STAFOR - integer variable in which a status indication is returned.

The allowable values for TAG and their meanings are:

<u>Value</u>	<u>Meaning</u>
0	X (not used)
1	Device Code and Buffer's Next Address
2	Buffer Control and Current Buffer Count
3	Port Status and Next DCW Address
4	Interrupt Status
5	X (not used)
6	Bootstrap information
7	Bootstrap information

The format of the integer variable 'STAFOR' for the various tag values is defined in Reference 4.

Additional special usage of the tag field for the special ports (EXEC, SDS, IDLE) is also defined in Reference 4.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>STATUS CD Meaning</u>
29	TAG error, if set

8.11 MITNOR PROCEDURE

The MITNOR integer procedure enables the THLL programmer to prepare a DCP input port for receiving unsolicited interrupt words from peripherals on the port. The form of the procedure call is:

MITNOR (EXP, PORT)

where

EXP - integer expression whose value (the Buffer control field) is sent to the DCP.

PORT - integer expression which specifies the input port to be affected.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>MITNOR CD Meaning</u>
28	Indicates port busy status

8.12 ACKNOWLEDGE DEDICATED INTERRUPT PROCEDURES

The ADED1 and ADED2 integer procedures enable the THLL programmer to retrieve interrupt information from ports that have been assigned interrupt levels which are not associated with a DCP stack. In particular, ADED1 retrieves all 32 bits from the interrupting port. ADED2 retrieves 24 bits from the interrupting port (most significant bits of the returned value) and eight bits from the DCP (least significant bits of the returned value). The forms for the ADED1 and ADED2 procedure calls are:

ADED1 (PORT, INTFOR)
 ADED2 (PORT, INTFOR)

where

PORT - integer expression which specifies the port to be acknowledged.

INTFOR - integer variable to contain the interrupt information returned.

The meaning of the returned CD bits is as follows:

<u>Bit</u>	<u>CD Meaning</u>
30	If set, interrupt is pending
28	Should be set indicating port is busy

CHAPTER 9

REAL TIME OPERATING SYSTEM SERVICE PROCEDURES

The Real Time Operating System (RTOS) serves as the runtime interface between a program and its external environment. The details of RTOS services are documented in the Real Time Operating System Monitor Design Disclosure Document (Reference 3). For the THLL programmer, this interface is implemented as a set of service procedures. The descriptions of these service procedures follow. These procedures can be found in the THLL runtime library, SYSLIB. Appendix C shows how to access this library.

9.1 ATTACH PROCEDURE

The ATTACH procedure serves as the interface for the Attach Service Module contained within the RTOS MONITOR and is used by a programmer whenever a task is to be connected to the Multitask Set (MTS). The form for the ATTACH procedure call is:

ATTACH(NAME,OPT,POST,LNKWD)

where

- NAME - a character string of from one to eight characters which specifies the name of the task to be connected to the MTS.
- a double array element containing the name of the task to be connected to the MTS.
- a double variable whose first word contains the number of tasks to be connected to the MTS and whose second word contains a pointer to the list of task names.
- OPT - an integer expression which specifies certain ATTACH options.
- POST - a pointer to a data area whose length is $16 + 2(N-1)$ where N is the number of tasks being attached with this ATTACH call.
- LNKWD - an integer identifier which defines the linkage word for exit processing.

The OPT expression defines options in a bit-wise manner. The effect of a bit being set is defined as follows:

<u>Bit</u>	<u>Effect</u>
31 (least significant bit)	LNKWD specifies an address of exit processing.
30	NOLOAD: the loading of the task is to be deferred.

The first word of the POST area contains the ATTACH processing codes. The meaning of these codes is defined as follows.

<u>Posting Code</u>	<u>Meaning</u>
0	Parameters Valid, Processing Complete
-1	Task Already Connected
-50	Insufficient Main Memory to Load Task

The second and third words of the POST area contain the name of the task.

9.2 SCHEDULE PROCEDURE

The SCHEDULE procedure serves as the interface for the Schedule Service Module contained within the RTOS MONITOR and is used by a programmer whenever a task is to be initiated for execution. The form for the SCHEDULE procedure call is:

SCHEDULE (NAME, OPT, POST, PREC)

where

- NAME - a character string of from one to eight characters which specifies the name of the task to be initiated for execution.
- a double array element containing the name of the task to be initiated for execution.
- a double variable whose first word contains the number of tasks to be initiated for execution and whose second word contains a pointer to the list of task names.
- OPT - an integer expression which specifies certain SCHEDULE options.

POST - a pointer to a data area whose length is $16 + 2(N-1)$ where N is the number of tasks.

PREC - an integer value; each bit set in the binary representation of this value indicates the rank of a predecessor task.

The OPT expression defines options in a bit-wise manner. The effect of a bit being set is defined below:

<u>Bit</u>	<u>Effect</u>
30	PREC: the PREC expression is used to effect predecessor requirements.
29	REL: the memory assigned to the task is released at the end of its execution.

The first word of the POST area contains the SCHEDULE processing codes. The meaning of these codes is defined as follows:

<u>Posting Code</u>	<u>Meaning</u>
0	Parameters Valid, Processing Complete
-1, -2	Task Activity Already in Progress, where -1 indicates task already scheduled
-50	Insufficient Main Memory to Load Task

The second and third words of the POST area contain the name of the task.

9.3 PURGE PROCEDURE

The PURGE procedure serves as the interface for the Purge Service Module contained within the RTOS MONITOR and is used by a programmer to indicate an abnormal or forced end of execution for a task. The form for the PURGE procedure call is:

PURGE(NAME,OPT,POST,FAULT)

where

- NAME - a character string of from one to eight characters which specifies the name of the task to be purged from execution.
- a double array element containing the name of the task to be purged from execution.
 - a double variable whose first word contains the number of tasks to be purged from execution and whose second word contains a pointer to the list of task names.
- OPT - an integer expression which specifies certain PURGE options.
- POST - a pointer to a data area whose length is $16 + 2(N-1)$ where N is the number of tasks.
- FAULT - an integer expression in which bits 8 through 31 identify the nature of the abnormal end.

The OPT expression defines options in a bit-wise manner. The effect of a bit being set is defined below:

<u>Bit</u>	<u>Effect</u>
30	FAULT: the fault expression is used to specify the nature of the abnormal end.

The first word of the POST area contains the PURGE processing codes. The meaning of these codes is defined as follows:

<u>Posting Code</u>	<u>Meaning</u>
0	Parameters Valid; Processing Complete
-1	Attempt to PURGE a Non-Active Task
-2	Task Termination Already in Progress

The second and third words of the POST area contain the name of the task.

9.4 DETACH PROCEDURE

The DETACH procedure serves as the interface for the Detach Service Module contained within the RTOS MONITOR and is used by a programmer whenever a task is to be disconnected from the MTS. The form for the DETACH procedure call is:

DETACH(NAME, POST)

where

- NAME - a character string of from one to eight characters which specifies the name of the task to be detached from the MTS.
- a double array element containing the name of the task to be detached from the MTS.
 - a double variable whose first word contains the number of tasks to be detached from the MTS and whose second word contains a pointer to the list of task names.
- POST - a pointer to a data area whose length is $16 + 2(N-1)$ where N is the number of tasks.

The first word of the POST area contains the DETACH processing codes. The meaning of these codes is defined as follows:

<u>Posting Code</u>	<u>Meaning</u>
0	Parameters Valid; Processing Complete
-2	Task Termination Already in Progress

The second and third words of the POST area contain the name of the task.

9.5 EXIT PROGRAM PROCEDURE

The EXITPGM procedure serves as the interface for the Exit Service Module contained within the RTOS MONITOR and is used by a programmer to relinquish BP control or to signal normal end of execution. The form for the EXITPGM procedure call is:

EXITPGM

This procedure should NEVER be used within an interrupt procedure as it is designed to signal the end of task execution.

9.6 WAITE SERVICE MODULE PROCEDURES

The MARKTIME, WAITMEMO, WAITKBIN, WAITICL, WAITTASK, and WAITEEXEC procedures serve as the interface for the Waite Service Module contained within the RTOS MONITOR and are used by a programmer to suspend a program's activity until a selected event has occurred. The event that is selected for each procedure is as follows:

<u>Procedure</u>	<u>Event</u>
MARKTIME	Time Interval Elapse
WAITMEMO	Main Memory Availability
WAITKBIN	Keyboard Input from Operator
WAITICL	Completion of ICL Transfer
WAITTASK	Completion of another Task
WAITEEXEC	Executive Action

The form for each procedure call follows.

9.6.1 MARKTIME

MARKTIME (DELTA [, WS])

where

DELTA - a real expression which indicates the number of seconds that are to elapse until that task can be continued.

WS - an optional pointer to a working storage area with a length of at least 16 words.

9.6.2 WAITMEMO, WAITKBIN, WAITICL, WAITEEXEC

WAITMEMO [(WS)]
 WAITKBIN [(WS)]
 WAITICL [(WS)]
 WAITEEXEC [(WS)]

where

WS - an optional pointer to a working storage area with a length of at least 16 words.

9.6.3 WAITTASK

WAITTASK(TECKEY[,WS])

where

TECKEY - an integer value; each bit set in the binary representation of this value indicates the rank of a predecessor task.

WS - an optional pointer to a working storage area with a length of at least 16 words.

9.7 CONTINUE PROCEDURE

The CONTINUE procedure serves as the interface for the Continue Service Module contained within the RTOS MONITOR and is used by a STATE EXECUTIVE programmer to continue the execution of those tasks that are waiting for Executive Action. The form for the CONTINUE procedure call is:

CONTINUE(PGM[,WS])

where

PGM - an integer value; each bit set in the binary representation of this value indicates the rank of a task to be continued.

WS - an optional pointer to a working storage area with a length of at least 16 words.

9.8 ENABLE, DISABLE, ARM, DISARM, AND TRIGGER PROCEDURES

The ENABLE, DISABLE, ARM, DISARM, and TRIGGER procedures serve as the interfaces to the Enable, Disable, Arm, Disarm, and Trigger RTOS MONITOR Service Modules, respectively. They are used by a programmer to change the interrupt characteristics of the DCC by manipulating the contents of the Interrupt Disable Register, the Interrupt Mask Register, and the Interrupt Occurrence Register. The particular characteristic of each procedure is:

<u>Procedure</u>	<u>Characteristic</u>
ENABLE	Enable Interrupts
DISABLE	Disable Interrupts
ARM	Arm Interrupts
DISARM	Disarm Interrupts
TRIGGER	Simulate the Occurrence of Interrupts

The form for each procedure call is:

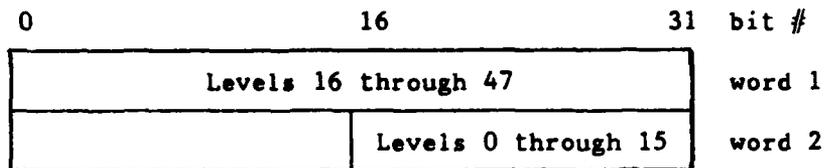
```
ENABLE(MASK[,WS])
DISABLE(MASK[,WS])
ARM(MASK[,WS])
DISARM(MASK[,WS])
TRIGGER(MASK[,WS])
```

where

MASK - a double expression which indicates the interrupt levels that are to be affected.

WS - an optional pointer to a working storage area with a length of at least 16 words.

The MASK expression indicates, in a bit-wise manner, the interrupt levels to be affected. Each bit set in this doubleword indicates action for the corresponding interrupt level. The format of this double expression is as follows:



9.9 OVERLAY LOADER PROCEDURE

The OVLOAD procedure serves as the interface for the Overlay Loader contained within the RTOS MONITOR and is used by a Task programmer to request the loading of secondary overlays. The form of the OVLOAD procedure call is:

OVLOAD(OVLY,POST)

where

OVLY - an integer expression which indicates the overlay number of the overlay that is to be loaded.

POST - a pointer to a data area with a length of at least 36 words.

The first word of the POST area contains the OVLOAD processing summary codes. The meaning of these codes is defined as follows:

<u>Code</u>	<u>Meaning</u>
0	Parameter Valid; Processing Complete
-9	DIO to MDF Failure

The second word of the Post area contains the level one and level two DIO processing codes. These codes are the same as those for the MDF READ and WRITE procedures (see Section 5.2.3).

9.10 ACQUIRE PROCEDURE

The ACQUIRE procedure serves as the interface to the Acquire Service Module contained within the RTOS MONITOR and is used by a programmer whenever main memory must be allocated. The form of the ACQUIRE procedure call is:

ACQUIRE(MEM,SIZE[,WS])

where

MEM - an integer variable to contain the absolute address of the memory block allocated.

SIZE - an integer expression which specifies the number of words to be allocated.

WS - an optional pointer to a working storage area which contains at least 16 words.

If main memory cannot be allocated, then a value of -1 is returned in MEM.

9.11 RELEASE PROCEDURES - RELEASE - ALLMEM

The RELEASE and ALLMEM procedures serve as interfaces to the Release Service Module contained within the RTOS MONITOR. The RELEASE procedure is used to return a main memory block that was dynamically acquired. The form of the RELEASE procedure call is:

```
RELEASE(MEM[,WS])
```

where

MEM - an integer variable which contains the absolute address of the main memory block to be deallocated.

WS - an optional pointer to a working storage area which contains at least 16 words.

The ALLMEM procedure is used to return or deallocate all main memory allocated to the program via the ACQUIRE procedure. The form of the ALLMEM procedure call is:

```
ALLMEM[(WS)]
```

where

WS - an optional pointer to a working storage area which contains at least 16 words.

9.12 SETSTATUS PROCEDURE

The SETSTATUS integer procedure serves as the interface for the F/C Status Service Module. The value of the procedure is the posting code. It enables the THLL programmer to change a status controlled by the MONITOR. The form of the SETSTATUS procedure call is:

```
SETSTATUS(ICODE,ISUB[,POST])
```

where

ICODE - an integer expression which specifies the device for which the status is to be changed.

ISUB - an integer expression which specifies the condition for which status is to be modified.

POST - an optional pointer to a data area with a length of at least 16 words.

The values for ICODE and the device each represents are defined below:

<u>ICODE Value</u>	<u>Device</u>
1	DCG - Data Control Group
2	DCC - Digital Control Computer
3	MDF - Magnetic Disk File
4	DCSS - Data Converter Subsystem/ Time of Day Subsystem
5	SPR - System printer
6	CPR - Computer printer

The values for ISUB are device-dependent.

<u>Device</u>	<u>ISUB Value</u>	<u>Condition</u>
DCG	1	DCG Equipment Failure
	2	DCG Configuration Incomplete
	3	DCG Alert
	4	DCG Task Control Bus Alarm
	5	DCG System Mode Bus Alarm
	6	DCG Peripheral Configuration Alarm
	7	DCG Peripheral Equipment Failure
DCC	0	DCC Normal
	1	DCC Hardware Fault
	2	Computations Invalid
MDF	0	MDF Normal
	1	MDF Alarm
DCSS	0	DCSS Normal
	1	DCSS Alarm

SPR	1	System Printer Alert
CPR	1	Computer Printer Alert

The first word of the optional POST area also contains the SETSTATUS posting code. The meaning of these codes are defined as follows:

<u>Posting Code</u>	<u>Meaning</u>
0	All Parameters Valid, Processing Complete
-1	Requested Status Already Set
-2	Invalid Code Specified
-3	Invalid Device Specified

9.13 SYSERR PROCEDURE

SYSERR is a procedure which calls the RTOS MONITOR system error printout routine.

The form of the SYSERR procedure call is:

```
SYSERR(TYPE, FAULT[, SUBFAULT])
```

where

TYPE - an integer expression which indicates the software diagnosing the error.

0	MONITOR diagnosed system error
otherwise	Assignment of this integer is mode executive dependent.

FAULT - an integer expression which specifies the fault code number.

SUBFAULT - an optional integer expression which specifies the subfault code number.

9.14 DATASUM PROCEDURE

The DATASUM integer procedure serves as the interface to the RTOS SUM routine. It enables the THLL programmer to sum a block of data. The value of the procedure is the sum of all words in the data block specified. The form of the DATASUM procedure call is:

```
DATASUM(PTR,SIZE[,POST])
```

where

PTR - a pointer to the data block to be summed.

SIZE - an integer expression for the number of words to be summed.

POST - an optional pointer to a data buffer of at least 16 words.

The first word of the optional POST area also contains the sum of the data block.

9.15 EES PROCEDURE

The Enter Executive State (EES) procedure enables the THLL programmer to call any RTOS EES service routine or write his own EES handler. The form of the EES procedure call is:

```
EES(N,FPT)
```

where

N - integer number of the EES procedure to be called.

FPT - pointer to the Function Parameter Table (FPT) to be used by the EES procedure.

Many of the RTOS EES routines have been implemented for THLL and are described in Sections 9.1 through 9.14. This procedure provides the mechanism for a programmer to write his own handler after studying the RTOS requirements. See Reference 3, paragraph 6.14.6 for information on EES functions.

Table 6.14-1 of Reference 3 contains the values which N may assume. Additional information on FPTs is also available in Reference 3.

CHAPTER 10

REFERENCES

1. Hartmut G. Huber, THLL Reference Manual, NSWC TR 84-101, Jul 1984.
2. TRIDENT Basic Processor Programmers Reference Manual, Hughes Aircraft Company, Jul 1974.
3. NAVSEA OD 45601, Volume II, Part 1, TRIDENT-I and TRIDENT-I Backfit Fire Control Software Real Time Operating System Monitor Design Disclosure Document, Nov 1976.
4. NAVSEA OD 46359, Revision 2, Subset, Data Communication Processor Protocol.
5. Hartmut G. Huber, TRILOAD Reference Manual, 1984. (To be published)
6. J. Gaines, N. Raines, J. Schneider, and W. McCoy, TRIDENT On-Line Aid in Debugging Compiler (TOADC) User's Guide, NSWC/DL TR 3757, November 1977 (Revised August 1979).

APPENDIX A

ASCII CHARACTER SET

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	c	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1
STX	Start of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3
EOT	End of Transmission	DC4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tabulation	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tab	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
SP	Space	DEL	Delete

The table above represents the ASCII character set. At the top of the table are hexadecimal digits (0 to 7), and to the left of the table are hexadecimal digits (0 to F). To determine the hexadecimal value of an ASCII

character, use the hexadecimal digit that corresponds to the row in the "units" position, and use the hexadecimal digit that corresponds to the column in the "16's" position. For example, the value of the character representing the equal sign is 3D.

APPENDIX B

RUNTIME ENVIRONMENT ON THE BP

B.1 CONTROL SECTIONS

THLL programs for the BP are broken into three control sections. Control sections are used to identify characteristics associated with a data area or instruction area of a program. The following table defines the BP control sections and also indicates the memory alignment and protections associated with each section.

<u>Section</u>	<u>Purpose</u>	<u>Alignment</u>	<u>Protections</u>
O\$\$	Own Data Area	doubleword address	read,write
I\$\$	Executable Area	word address	execute only
C\$\$	Constant Area	doubleword address	read only

Each control section is relocatable. The BP loader, TRILOAD, concatenates the same control section from all compile units.

Argument lists are constant on the BP. Either the called procedure evaluates the address of the argument (passed by reference) or it copies the value of the argument (passed by value). These values are placed in the runtime stack frame for that procedure invocation.

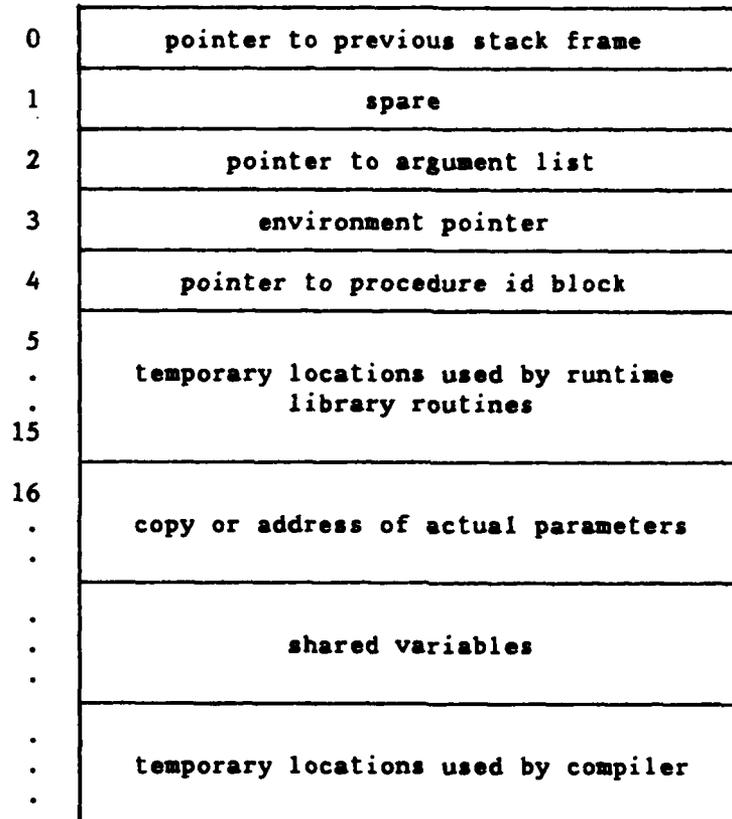
B.2 ALLOCATION OF STACK FRAMES

On entry to a procedure, a stack frame is allocated providing memory space for parameters, local variables, and temporaries. On exit from the procedure, the stack frame is released. X7 contains the address of the stack frame. X6 contains half the address of the stack frame (for accessing DOUBLE and REAL values).

The first 16 THLL words on the stack frame are used to support the THLL procedure environment. This area also serves as a scratch area for THLL library routines. The parameters to a procedure start at offset 16 on the stack frame. Shared variables are next on the stack frame, and temporary variables are normally last on each stack frame. The cross reference for a procedure indicates the hexadecimal byte offset of all user-defined items on the stack frame.

Stack frames on the BP are adjacent to each other in a memory block known as RSTACK\$. The interrupt stack, ISTACK\$, is used to keep track of all currently active stack frames and of the next available stack frame. The BP executive stack is used to store procedure linkage information, such as the return address.

The following diagram describes the structure of the stack frame. The offsets shown are in terms of THLL words.



B.2.1 Allocation of THLLFCB

Before a file can be used for reading or writing, it must be opened by calling the procedure OPEN. This procedure opens the file, creates a THLL File Control Block (THLLFCB), and returns the address of the THLLFCB as its value.

Eight words for the THLLFCB are allocated by extending the stack frame of the procedure in which the call to OPEN appears. This means that this storage area is released on exit from that procedure. Therefore, all read and write operations on the opened file must be done before returning from that

procedure. For more information on OPEN, see the THLL Reference Manual (Reference 1).

B.2.2 Temporaries

During the compilation process, it sometimes becomes necessary to employ temporary locations on the runtime stack (as shown in the diagram in Section B.2) for storage of intermediate results. Normally the temporaries follow the shared variables. However, if the amount of storage required for the whole stack frame exceeds 4096 words, the area for the temporaries is moved before the formal parameter area and the other areas are moved to higher displacements. This is done to ensure 'easy' access for temporaries since a base register can span only 4096 words.

APPENDIX C

USING BP TRICOMP

C.1 INTRODUCTION

By default, BP TRICOMP translates THLL code into BP binary code. This implies a two-step process in order to create an executable file. These steps are:

1. Compile THLL code producing BP binary code.
2. Link the binary code into an executable file.

This appendix explains these steps. Additionally, it briefly describes how to produce the various reports available for THLL programs.

Note that filenames under VMS have the format Basename.EXT. Basename is the base filename for a set of related files that have different .EXT's extensions. Most VAX programs have a set of conventions governing the .EXT extensions. This is true for BP TRICOMP also. The user can override these conventions, but this is discouraged for configuration management reasons. The organization of a THLL program as presented in this appendix is not the only organization, but it is a logical one that should be considered seriously before deviating to another.

C.2 ORGANIZATION OF A THLL PROGRAM

A program written in THLL (targeted for the BP) consists of one or more compile units, all of which must be compiled, assembled, linked, and then executed. One usually thinks of a compile unit as a module, that is, a set of related procedures and data that can be compiled separately. On the VAX, a single compile unit maps nicely into a single EDT file having a .THL extension. Just as a compile unit maps into a file, a program (which is a collection of compile units) maps into a directory (which is a collection of files). Thus, on the VAX, a program written in THLL consists of one or more .THL files (each of which contains one compile unit) in a common directory. These .THL files are then compiled, assembled, linked, and then executed. Each step of this process creates additional files, most of which have the same base filenames as the .THL files.

C.3 THE BP TRICOMP ENVIRONMENT

The BP TRICOMP environment is part of the VAX/VMS environment. It consists of a set of commands to produce an executable THLL program, a global cross reference report, and procedure call tree reports. The set of commands consists of the following:

1. TRICOMP - The command used to compile a THLL compile unit.
2. TRILOAD - The command used to link the BP binary files into an executable image.
3. GXRUPDATE - The command used to combine .GXR files into a .MXD file.
4. GXRREPORT - The command used to produce a global cross reference report (.MXR file) from a .MXD file.
5. TREUPDATE - The command used to combine .TRE files into a .MTD file.
6. TREReport - The command used to produce a procedure call tree report (.MTR file) from a .MTD file.
7. NTREReport - The command used to produce a nested procedure call tree report (.NTR file) from a .MTD file.

These commands are available as all VMS commands are available. The THLL Reference Manual (Reference 1) and the VMS HELP command may be used to receive additional information about the above commands.

C.4 INPUT TO BP TRICOMP

C.4.1 Compile Units (.THL Files)

By default BP TRICOMP expects a compile unit in a file with a .THL extension. Each .THL file contains one THLL compile unit. A .THL file is an input source file that can be maintained with a VAX editor. It is recommended that each .THL file have a base filename corresponding to the compile unit name contained in the file.

C.4.2 Insert Files (.THI Files)

In addition to the .THL files that correspond to compile units, a THLL program often includes INSERT files. An insert file is an input source file that must have a .THI extension. It can be created simply by editing a file. In order to locate a .THI file associated with the insert declaration:

```
INSERT FILENAME(DIRNAME)
```

BP TRICOMP searches for DIRNAME in the following order:

1. The VMS logical name table is searched for a definition of DIRNAME. If it exists, that name is the name of the directory that contains the FILENAME.THI file. The DEFINE command in VMS is used to establish this relationship in the logical name table.
2. If DIRNAME is the default identifier, DEFAULT, the current default directory contains the FILENAME.THI file. DEFAULT could be defined in the logical name table in which case the check above would have associated DEFAULT to a (possibly different) directory.
3. Finally, it is assumed that DIRNAME is a subdirectory in the current default directory.

DIRNAME and FILENAME are truncated to 8 characters. This approach gives the program designer quite a bit of flexibility.

C.5 INVOCATION OF BP TRICOMP (.THL, .THI FILES)

After having created .THL and .THI files, BP TRICOMP can be invoked by the following command:

```
$ TRICOMP/BP Filename
```

where Filename is the base filename of a .THL file. It is recommended that the current default directory is set to be the directory containing the .THL file before invoking BP TRICOMP. Otherwise, the association is lost between the .THL file and the files generated by BP TRICOMP.

C.6 OUTPUT FROM BP TRICOMP (.TLS, .BIN, .GXR, .TRE FILES)

The two file types, .THL and .THI, described above are created by the users as they develop their program. A .THL file contains a compile unit and it is the input file to a single invocation of BP TRICOMP. The .THI files contain information to be INSERTed into .THL files during a THLL compilation. BP TRICOMP compiles a single .THL file containing a single compile unit and produces four files. The default file extensions of the four files created by BP TRICOMP are:

- .TLS - Compiled listing produced by BP TRICOMP
- .BIN - BP binary code file corresponding to the THLL compile unit
- .GXR - Global cross reference data for the THLL compile unit
- .TRE - Procedure call tree data for the THLL compile unit

By default the base filename of each of these files is that of the .THL file containing the compile unit. BP TRICOMP creates the four output filenames by appending the above extensions to the base filename of the .THL file. See Section C.14 for ways to override the defaults.

The .TLS files produced by BP TRICOMP can be examined by the programmers to ensure that their development is proceeding as planned. The .BIN, .GXR, and .TRE files produced by BP TRICOMP are used as input to other programs.

C.7 INVOCATION OF TRILOAD (.BIN, .MAP, .LOD FILES)

The .BIN files created by BP TRICOMP contain BP binary code corresponding to the THLL compile units contained in the .THL files. The .BIN files are input to the BP linker-loader, TRILOAD, which collects them into an executable load module. Using TRILOAD has many variations. All (or part) of the .BIN files can be combined into an object library (via the LIBRARY command) and TRILOAD can select them from there, or TRILOAD can simply link the .BIN files. The details of using TRILOAD can be found in the TRILOAD Reference Manual (Reference 5).

The following is an example invocation of TRILOAD in which several .BIN files (FILE1, FILE2, FILE3) are collected into an executable load module.

Forthcoming.

This example selects the base filename of the executable load module and the map file to be OUT. Thus, the output of this command is two files with the base filename OUT. The default file extensions of these two files are:

.LOD - This file is an executable load module which may be transported to the BP for execution

.MAP - This file contains the program's load map

Note that up until this .LOD file, four files have accumulated for each compile unit. For each Basename.THL file containing a compile unit, the THLL user derives Basename.TLS, Basename.GXR, Basename.TRE, and Basename.BIN. The ultimate goal is to combine a collection of .THL files into one executable load module. Thus, even starting with three .THL files in a directory, only one .LOD file results (in the same directory).

C.8 RUNNING A PROGRAM (.LOD FILES)

After having created the .LOD file on the VAX, it must be transported to the BP for execution. The process of going from an executable load module on the VAX to actually executing on the BP is long and complicated.

C.9 DEBUGGING (.LIS FILES)

For information on debugging, see the TOADC User's Guide (Reference 6).

C.10 PRODUCING A GLOBAL CROSS REFERENCE REPORT (.GXR, .MXD, .MXR FILES)

The global cross reference report presents an overview of the global symbols of a program. The actual individual compile unit line references for those symbols (as well as symbols local to a compile unit) can be found in the local cross reference at the end of the .TLS file. The global cross reference report is useful in determining which procedures and modules reference a particular global symbol. Symbols defined within an insert file are in a sense global; thus they are also included in the global cross reference report. The global cross reference report is developed from .GXR files in a two-step process.

The first step is to combine the .GXR files into a master cross reference data file (.MXD). This is done by invoking GXRUPDATE. The details of GXRUPDATE are presented in the THLL Reference Manual (Reference 1). The following command invokes GXRUPDATE:

```
$ GXRUPDATE Filename
```

where Filename is the base filename of the desired .MXD file. GXRUPDATE combines the .GXR; files into the Filename.MXD file. The Filename.MXD file is the input file to the second step of this process.

The second step is to transform the .MXD file into a readable global cross reference report. This is done by invoking GXRREPORT. The details of GXRREPORT are presented in the THLL Reference Manual. The following command invokes GXRREPORT:

```
$ GXRREPORT Filename
```

where Filename is the base filename of the .MXD file. The file produced by GXRREPORT is Filename.MXR. This report can be examined and/or printed.

This two-step process allows for incremental updates to the global cross reference data. As the need arises to change selected compile units, the .THL files are changed, compiled using BP TRICOMP, assembled, and linked. Later, the global cross reference data files are gathered by invoking GXRUPDATE to do a partial update to the .MXD file. GXRREPORT is used to form the new report. The changed compile units are reflected accordingly in the new report.

C.11 PRODUCING A PROCEDURE CALL TREE REPORT (.TRE, .MTD, .MTR FILES)

The procedure call tree report shows which procedures call which procedures. This report is produced in a two-step process that mirrors the two-step process for producing the global cross reference report.

The first step is to combine the .TRE files into a master procedure call tree data file (.MTD). This is done by invoking TREUPDATE. The details of TREUPDATE are presented in the THLL Reference Manual. The following command invokes TREUPDATE:

```
$ TREUPDATE Filename
```

where Filename is the base filename of the desired .MTD file. TREUPDATE combines the .TRE; files into the Filename.MTD file. The Filename.MTD file is the input file to the second step of this process.

The second step is to transform the .MTD file into a readable procedure call tree report. This is done by invoking TREReport. The details of TREReport are presented in the THLL Reference Manual. The following command invokes TREReport:

```
$ TREReport Filename
```

where Filename is the base filename of the .MTD file. The file produced by TREReport is Filename.MTR. This report can be examined and/or printed.

This two-step process allows for incremental updates to the procedure call tree data. As the need arises to change selected compile units, the .THL files are changed, compiled using BP TRICOMP, assembled, and linked. Later, the procedure call tree data files are gathered by invoking TREUPDATE to do a partial update to the .MTD file. TREReport is used to form the new report. The changed compile units are reflected accordingly in the new report.

C.12 PRODUCING A NESTED PROCEDURE CALL TREE REPORT (.TRE, .MTD, .NTR FILES)

The nested procedure call tree report shows which procedures call which procedures in a nested format. This report is produced in a two-step process that mirrors the two-step process for producing the regular procedure call tree report.

The first step is exactly the same as the first step of producing a procedure call tree report. The first step is to combine the .TRE files into a master procedure call tree data file (.MTD). This is done by invoking TREUPDATE as shown above. If the .MTD file has been created for a regular procedure call tree report, it can be used for a nested procedure call tree report.

The second step is to transform the .MTD file into a readable nested procedure call tree report. This is done by invoking NTREREPORT. The details of NTREREPORT are presented in the THLL Reference Manual. The following command invokes NTREREPORT:

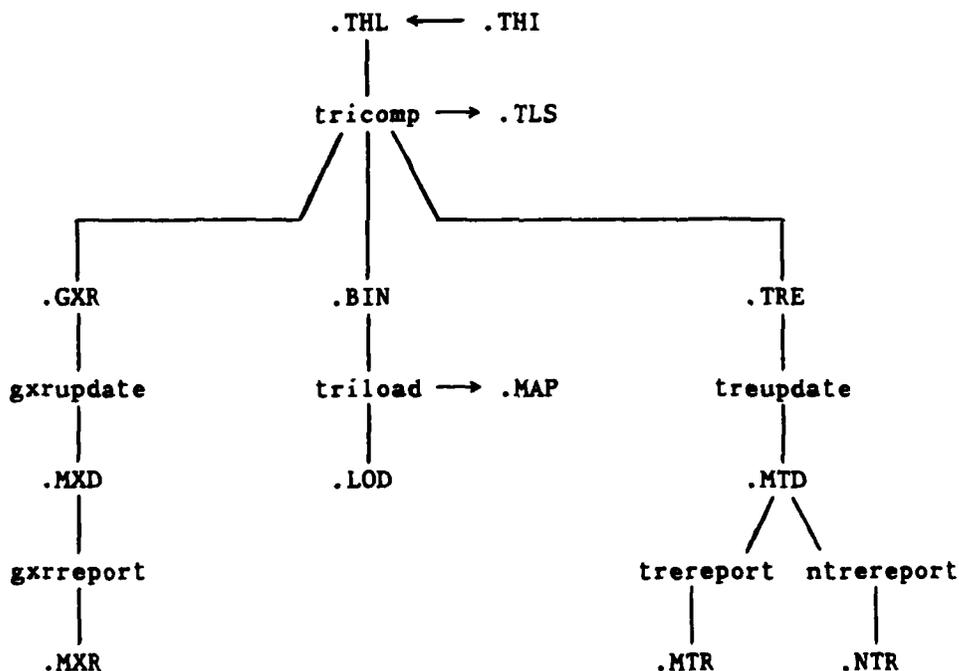
```
$ NTREREPORT Filename
```

where Filename is the base filename of the .MTD file. The file produced by NTREREPORT is Filename.NTR. This report can be examined and/or printed.

As in producing a regular procedure call tree report, this two-step process allows for incremental updates to the nested procedure call tree data.

C.13 DIAGRAM OF A THLL PROGRAM'S FILE RELATIONSHIPS

The preceding description is summarized pictorially in the following diagram. Flow is downward.



C.14 ADVANCED INVOCATIONS OF TRICOMP (OVERRIDING DEFAULTS)

The preceding sections of this appendix describe how to use BP TRICOMP, accepting the defaults. Sometimes defaults are not sufficient and sometimes defaults constrict a programmer's creativity. If the need arises, BP TRICOMP provides, in the form command qualifiers, ways to override the defaults. All

command qualifiers can be seen in the THLL Reference Manual or via the HELP TRICOMP command.

C.14.1 Suppressing TRICOMP Generated Files

Often utility programs consist of a small number (sometimes one) of compile units. In these cases, the .GXR and .TRE files are cumbersome (or not necessary). The following command does not produce a .GXR and a .TRE file.

```
$ TRICOMP/BP/NOGXR/NOTRE UTIL
```

C.14.2 Parallel Directory Organization

Sometimes a program is so large that keeping all of the files in the same directory would be unmanageable. A valid approach is to view a program as a collection of parallel directories where:

1. [PROGRAM.THL] contains the .THL files,
2. [PROGRAM.TLS] contains the .TLS files,
3. [PROGRAM.BIN] contains the .BIN files,
4. [PROGRAM.GXR] contains the .GXR files, and
5. [PROGRAM.TRE] contains the .TRE files.

The following command sprays the .TLS, .BIN, .GXR, and .TRE files into the appropriate directories for [PROGRAM.THL] FILE1.THL.

```
$ TRICOMP/BP/TLS=[PROGRAM.TLS]/BIN=[PROGRAM.BIN]-
$_/GXR=[PROGRAM.GXR]/TRE=[PROGRAM.TRE] [PROGRAM.THL] FILE1
```

C.14.3 Overriding Default File Extensions

Occasionally programmers believe that their own conventions are more expressive than the default conventions. The following command compiles a THLL compile unit contained in file PROG.SRC and places the BP binary code in file PROG.BNY.

```
$ TRICOMP/BP/NOTRE/NOGXR/BIN=.BNY PROG.SRC
```

Note that the THLL listing is placed in the default PROG.TLS file.

C.14.4 Example Compile and Library Command Procedure

BP TRICOMP produces BP binary code which must be linked by TRILOAD into an executable load module. Very often, binary libraries are used to collect the .BIN files, which implies there is an intermediate step of placing the .BIN files into a library prior to creating the executable load module. If there are no compile errors in the .THL file, the following command procedure places the .BIN file into the THLL library PROGRAM.TLB.

```
$ ON WARNING THEN EXIT
$ TRICOMP/BP/NOGXR/NOTRE 'P1'
$ MNAME = F$PARSE(P1,,,"NAME")
$ LIBRARY/TEXT PROGRAM 'MNAME'.BIN
$ DEL 'MNAME'.BIN;
$ EXIT
```

C.14.5 \$SEVERITY Returned from BP TRICOMP

The preceding example command file relies upon the \$SEVERITY system symbol. When TRICOMP exits to VMS, \$SEVERITY is set to indicate how the compilation went. The following values can be found in \$SEVERITY.

1. \$SEVERITY = 1 implies a normal compilation
2. \$SEVERITY = 0 implies normal compilation with compile errors
3. \$SEVERITY = 2 implies abnormal compilation
4. \$SEVERITY = 4 implies one of the pertinent files could not be found or opened

Note that if the \\ ABORT directive is contained within a compile unit, then a \$SEVERITY of 0 becomes a 2 and a \$SEVERITY of 2 becomes a 4.

C.15 EXAMPLE SESSION ON USING BP TRICOMP

The following is an example session using the BP TRICOMP environment. All of the features discussed above are covered in the example. Users confused by the above descriptions may want to create the necessary files and actually perform this session. The necessary files are DEMOMAIN.THL, DEMOPRIME.THL, and EXTEND.THI. These files are listed in this appendix following this example. An understanding of VAX facilities for creating files is assumed. A slight twist on the example is to try putting the EXTEND.THI file in the same (or parallel) directory as the .THL files.

```
$ SD.
```

```
UDISK1:[username]

$ CREATE/DIR [.DEMO]
$ SD.DEMO

UDISK1:[username.DEMO]

$
$ ! create DEMOMAIN.THL and DEMOPRIME.THL (listed in appendix)
$ ! using one of the VAX editors
$
$ DIR

Directory UDISK1:[username.DEMO]

DEMOMAIN.THL;1 DEMOPRIME.THL;1

Total of 2 files.
$ CREATE/DIR [.INSERTS]
$ SD.INSERTS

UDISK1:[username.DEMO.INSERTS]

$
$ ! create EXTEND.THI (listed in appendix) using one of the
$ ! VAX editors
$
$ SD. ! sets home directory as default directory

UDISK1:[username]

$ SD [.DEMO]

UDISK1:[username.DEMO]

$
$ ! show the program's structure
$
$ DIR

Directory UDISK1:[username.DEMO]

DEMOMAIN.THL;1 DEMOPRIME.THL;1 INSERTS.THI;1

Total of 3 files.
$ SD.INSERTS ! subdirectory containing insert files

UDISK1:[username.DEMO.INSERTS]

$ DIR

Directory UDISK1:[username.DEMO.INSERTS]
```

EXTEND.THI;1

Total of 1 file.

\$ SD ^

UDISK1:[username.DEMO]

\$
 \$! compile .THL files (each has one compile unit)
 \$
 \$ TRICOMP/BP DEMOMAIN
 BEGIN BP TRICOMP
 NORMAL COMPILATION DEMOMAIN
 END BP TRICOMP
 \$ TRICOMP/BP DEMOPRIME
 BEGIN BP TRICOMP
 NORMAL COMPILATION DEMOPRIME
 END BP TRICOMP
 \$ DIR

Directory UDISK1:[username.DEMO]

DEMOMAIN.BIN;1	DEMOMAIN.GXR;1	DEMOMAIN.THL;1	DEMOMAIN.TLS;1
DEMOMAIN.TRE;1	DEMOPRIME.BIN;1	DEMOPRIME.GXR;1	DEMOPRIME.THL;1
DEMOPRIME.TLS;1	DEMOPRIME.TRE;1	INSERTS.DIR;1	

Total of 11 files.

\$
 \$
 \$! Invoke TRILOAD to produce an executable load module.
 \$! TRILOAD is forthcoming.
 \$! The process creates files DEMO.MAP and DEMO.LOD.
 \$
 \$
 \$! Produce a Global Cross Reference Report
 \$
 \$ GXRUPDATE DEMO
 \$ DEL *.GXR; ! Not needed after the update
 \$ DIR

Directory UDISK1:[username.DEMO]

DEMO.LOD;1	DEMO.MAP;1	DEMO.MXD;1	DEMOMAIN.BIN;1
DEMOMAIN.THL;1	DEMOMAIN.TLS;1	DEMOMAIN.TRE;1	DEMOPRIME.BIN;1
DEMOPRIME.THL;1	DEMOPRIME.TLS;1	DEMOPRIME.TRE;1	INSERTS.DIR;1

Total of 12 files.

\$ GXRREPORT/TITLE="Demonstration Use of BP TRICOMP" DEMO
 \$ DIR

Directory UDISK1:[username.DEMO]

```

DEMO.LOD;1      DEMO.MAP;1      DEMO.MXD;1      DEMO.MXR;1
DEMOMAIN.BIN;1  DEMOMAIN.THL;1  DEMOMAIN.TLS;1  DEMOMAIN.TRE;1
DEMOPRIME.BIN;1 DEMOPRIME.THL;1 DEMOPRIME.TLS;1 DEMOPRIME.TRE;1
INSERTS.DIR;1
    
```

Total of 13 files.

```

$
$ ! Produce a Procedure Call Tree Report
$
$ TREUPDATE DEMO
$ DEL *.TRE;      ! Not needed after the update
$ DIR
    
```

Directory UDISK1:[username.DEMO]

```

DEMO.LOD;1      DEMO.MAP;1      DEMO.MTD;1      DEMO.MXD;1
DEMO.MXR;1      DEMOMAIN.BIN;1  DEMOMAIN.THL;1  DEMOMAIN.TLS;1
DEMOPRIME.BIN;1 DEMOPRIME.THL;1 DEMOPRIME.TLS;1 INSERTS.DIR;1
    
```

Total of 12 files.

```

$ TREREPORT/TITLE="Demonstration Use of BP TRICOMP" DEMO
$ DIR
    
```

Directory UDISK1:[username.DEMO]

```

DEMO.LOD;1      DEMO.MAP;1      DEMO.MTD;1      DEMO.MTR;1
DEMO.MXD;1      DEMO.MXR;1      DEMOMAIN.BIN;1  DEMOMAIN.THL;1
DEMOMAIN.TLS;1  DEMOPRIME.BIN;1 DEMOPRIME.THL;1 DEMOPRIME.TLS;1
INSERTS.DIR;1
    
```

Total of 13 files.

```

$
$ ! Produce a Nested Procedure Call Tree Report
$ ! Note the .MTD file already exists
$
$ NTREREPORT/TITLE="Demonstration Use of BP TRICOMP" DEMO
$ DIR
    
```

Directory UDISK1:[username.DEMO]

```

DEMO.LOD;1      DEMO.MAP;1      DEMO.MTD;1      DEMO.MTR;1
DEMO.MXD;1      DEMO.MXR;1      DEMO.NTR;1      DEMOMAIN.BIN;1
DEMOMAIN.THL;1  DEMOMAIN.TLS;1 DEMOPRIME.BIN;1  DEMOPRIME.THL;1
DEMOPRIME.TLS;1 INSERTS.DIR;1
    
```

Total of 14 files.

C.15.1 Example THLL Compile Unit (DEMOMAIN.THL FILE)

```

DEMOMAIN
  BEGIN

  COMMENT THIS PROGRAM DETERMINES THE PRIME NUMBERS FROM 2 TO 63.
  THE METHOD USED IS A FORM OF SIEVE ERATOSTHENES. THE
  NUMBERS TO SEARCH ARE IN THE ARRAY NUM. EACH BIT IN
  THE ARRAY CAND CORRESPONDS TO AN ENTRY IN NUM. AS LONG
  AS THE CAND BIT IS SET, THE NUM ENTRY IS A CANDIDATE
  FOR A PRIME NUMBER. IF AN ENTRY IN NUM IS FOUND TO BE
  A COMPOSITE NUMBER, ITS BIT IN CAND IS CLEARED. ;

  INSERT EXTEND(INSERTS) ;

  GLOBAL CAND,NUM ;

  OWN INTEGER ARRAY CAND(1),NUM(63) ;

  EXTERNAL PROCEDURE PRIME ;
  EXTERNAL PROCEDURE PRINT(INTEGER) ;

  INTEGER COMPONENT PRIMECAND (OFFSET 0,FIELD(0,1)) ;

  GLOBAL MAIN ;

  DEFINE PROCEDURE MAIN ;
    BEGIN
      INTEGER I ;
      POINTER P ;
      FOR I=0 STEP 1 UNTIL 63 DO
        NUM(I) = I+2
      ENDDO ;
      FOR I=0 STEP 1 UNTIL 1 DO
        CAND(I) = X'FFFFFFFF'
      ENDDO ;
      PRIME ;
      P = LOC CAND(0) ;
      FOR I=0 STEP 1 UNTIL 63 DO
        IF PRIMECAND(P,I) EQL 1 THEN
          PRINT(NUM(I))
        IFEND
      ENDDO ;
      END ;
    END FINIS

```

C.15.2 Example THLL Compile Unit (DEMOPRIME.THL FILE)

```

DEMOPRIME
  BEGIN
  COMMENT  THIS MODULE CONTAINS THE PROCEDURE FOR CALCULATING THE
           PRIME NUMBERS, ALONG WITH THE MODULE USED FOR PRINTING
           THE RESULTS. ;

  INSERT EXTEND(INSERTS) ;

  EXTERNAL PROCEDURE TERM(DEVICE, POINTER VALUE, ALPHA) ;

  EXTERNAL INTEGER ARRAY CAND(1) ;
  EXTERNAL INTEGER ARRAY NUM(63) ;

  ALPHA COMPONENT A0(OFFSET 1);
  INTEGER COMPONENT P0(FIELD(0,4), OFFSET 0);
  INTEGER COMPONENT PRIMECAND(OFFSET 0, FIELD(0,1)) ;

  GLOBAL PRIME ;

  DEFINE PROCEDURE PRIME ;
    BEGIN
    INTEGER CURPRIME, J, STOPVAL ;
    POINTER P ;
    P = LOC CAND(0) ;
    STOPVAL = 8 ;          /* SQRT(64)=8 */
    CURPRIME = 0 ;
    WHILE NUM(CURPRIME) LEQ STOPVAL DO
      BEGIN
      /* REMOVE MULTIPLES OF CURRENT PRIME FROM CANDIDATES */
      FOR J = CURPRIME+1 STEP 1 UNTIL 63 DO
        IF PRIMECAND(P, J) EQL 1 THEN
          IF (NUM(J) MOD NUM(CURPRIME)) EQL 0 THEN
            PRIMECAND(P, J) = 0
          IFEND
        IFEND
      ENDDO ;
      /* FIND NEXT PRIME TO PROCESS */
      CURPRIME = CURPRIME + 1 ;
      END
    ENDDO ;
  END ;

  GLOBAL PRTINT;

  DEFINE PROCEDURE PRTINT(INT) ;
    VALUE INT;
    INTEGER INT;

    BEGIN
    DEVICE SPR=SPRINT ;

```

```
OWN ALPHA PRTBUF(7) ;
OWN POINTER PTR,PTR1 ;
OWN INTEGER I ;
OWN INTEGER TEMP ;
OWN INTEGER ARRAY POSTBUF(15) ;
PRESET PRTBUF=#'12345678' ;
PTR = LOC INT ;
PTR1 = LOC PRTBUF ;
FOR I = 0 STEP 1 UNTIL 7 DO
  BEGIN
    TEMP = PO(PTR,I) ;
    IF TEMP GRT 9 THEN
      AO(PTR1,I) = TEMP + 55
    ELSE
      AO(PTR1,I) = TEMP + 48
    IFEND;
  END
ENDDO ;
TERM(SPR,LOC POSTBUF(0),PRTBUF);
END;
END FINIS
```

C.15.3 Example THLL Insert File (EXTEND.THI FILE)

\\ BOUNDS = 0 , XREF = 3

COMMENT THESE SYNONYM DECLARATIONS ARE USED TO EXTEND THLL ;

SYNONYM

```
BEGIN
NEXTCASE = / , / ;
ELSEIF = / , / ;
BOOLEAN = / INTEGER / ;
ENDDO = / / ;
END ;
```

C.15.4 Example Global Cross Reference Report (DEMO.MXR FILE)

GLOBAL SYMBOL NAME	CROSS REFERENCE ATTRIBUTES	Demonstration DEFINED IN DECK	Use of BP TRICOMP	USED IN DECK	USED IN PROCEDURE	PAGE 1 REFERENCES USED	CHNG
CAND	IA(1)	DEMOMAIN		DEMOMAIN DEMOPRIME	MAIN PRIME	1 1	1
ENDDO	SYNONYM	EXTEND	I	DEMOMAIN DEMOPRIME	MAIN PRIME PRTINT	3 2 1	
GEM	SYNONYM	EXTEND	I	DEMOMAIN DEMOPRIME		1 1	
MAIN	P(0)	DEMOMAIN					
NUM	IA(63)	DEMOMAIN		DEMOMAIN DEMOPRIME	MAIN PRIME	1 3	1
PRIME	P(0)	DEMOPRIME		DEMOMAIN	MAIN	1	
PRTINT	P(1)	DEMOPRIME		DEMOMAIN	MAIN	1	
TERM	X			DEMOPRIME	PRTINT	1	

C.15.5 Example Procedure Call Tree Report (DEMO.MTR FILE)

PROCEDURE CALL TREE Demonstration Use of BP TRICOMP

PAGE 1

MAIN	G	DEMOMAIN
	PRIME	*
	PRTINT	
PRIME	G	DEMOPRIME
PRTINT	G	DEMOPRIME
	TERM	X

C.15.6 Example Nested Procedure Call Tree Report (DEMO.NTR FILE)

PROCEDURE CALL TREE Demonstration Use of BP TRICOMP. PAGE 1
MAIN TREE 10-JUL-1984 10:18:01

<u>LINE</u>	<u>LEVEL</u>	<u>ROUTINE</u>
1	1	MAIN
2	2	PRIME
3	3	PRTINT
4	4	-TERM
5	2	PRTINT (3)

DISTRIBUTION

Library of Congress
Attn: Gift and Exchange Division
Washington, DC 20540 (4)

General Electric Company
Ordnance Systems
100 Plastics Avenue
Pittsfield, MA 01201
Attn: G. Desmarais (6)
J. Fenton (6)

Strategic Systems Program Office
Department of the Navy
Washington, DC 20376
Attn: SP-23115 (C. Chappell) (1)
SP-23423 (H. Cook) (1)

EG&G Washington Analytical
Services Center
P.O. Box 552
Dahlgren VA 22448
Attn: IMC (2)

Local:
K50-GE (1)
K51 (2)
K52 (12)
K53 (50)
K54 (20)
E31 (GIDEP Office) (1)
E431 (10)

END

FILMED

6-85

DTIC

END

FILMED

6-85

DTIC