IAC Doc. No. UG-I1050-0000-B

I4DDT USER'S GUIDE

JUNE 1973

FOREWORD

This document is a user's manual for a version of DDT (Dynamic Debugging Technique) referred to as I4. I4DDT operates under the TENEX time-sharing system, which runs on the DEC PDP-10 computer. I4DDT can be used for on-line checkout and testing of programs written in either assembly or compiler languages for PDP-10, PDP-11, and ILLIAC IV software systems.

Section 1 is an overview of the I4DDT program with more detailed descriptions of the features and commands contained in Sections 2 and 3, respectively. Since I4DDT's command syntax is similar to that of earlier versions of DDT, the experienced user may need only to refer to the command summary in Appendix A.

The following conventions are used throughout this manual:

- Upper-case letters, digits, and special characters must be entered literally as shown in the format representations.
- Information in lower-case letters in the format representations is variable data.
- Underlined text in the examples indicates information typed out by I4DDT, as distinguished from user-typed input.

The special terminal symbols used in this manual are:

SYMBOL	CHARACTER REPRESENTED
↳	Carriage Return
↑	The Control Key
$	ALTMODE or ESCAPE (ALT or ESC)
→|	Tab

CONTENTS

Page

APPENDICES

SECTION 1

INTRODUCTION

1.1 GENERAL DESCRIPTION

I4DDT is a generalized test-driver subsystem which includes an interactive control language that facilitates on-line checkout and testing of object programs. I4DDT resides in central PDP-10 memory, but it provides the user control and interaction with programs (to be debugged) which occupy any of several different address spaces: PDP-10, PDP-11, ILLIAC array memory, ILLIAC disk, or Test Maintenance Unit (TMU). Structurally I4DDT operates on user programs that run in inferior forks; as such, I4DDT simulates many executive functions. The user interacts with I4DDT and through I4DDT with the object program by entering commands from a terminal keyboard.

Examples of some of the checkout features of I4DDT follow: after the source program has been compiled or assembled, the binary object program with its symbol table is loaded under I4DDT. The user can specify locations in his program (breakpoints) where I4DDT is to suspend execution in order to accept further commands. In this way, the user can check out his program section by section and if an error occurs, insert the corrected code immediately. Either before I4DDT begins execution or at breakpoints, the user can examine and modify the contents of any memory location. Insertions and deletions can be in source language code or in various numeric and text modes. I4DDT also performs searches and calls user-coded debugging subroutines at breakpoint locations.

1.1.1 FEATURES

I4DDT's command language enables the user to quickly and easily locate, examine, and change the contents of any memory location. These facilities include the following capabilities:

- The user can specify for typeout the contents of any location for examination or alteration in a variety of formats; e.g., a location can be typed out as a symbolic instruction, numeric constant, floating-point number, ASCII or SIXBIT text, or half-word format. The output format can be set either permanently or temporarily.

- The user can type in modifications in a variety of formats, e.g., symbolic instructions, octal values, decimal floating-point numbers, ASCII text.
- The user can locate a word by combining symbols and numeric quantities into arithmetic expressions.
- The user can run all or any portion of the object program using the breakpoint feature. A breakpoint is set at a memory location to suspend program operation at that location, thus permitting the user to examine partial results and debug a program section by section. The user may also request that the contents of a specified memory location be typed out when the breakpoint is reached. Breakpoints can be set conditionally, so that a program stop occurs only if the condition is satisfied. In addition, a counter can be set up specifying the number of times a breakpoint is passed before a program stop occurs.
- The user can search through the object program for any quantity or effective address, and type out the addresses and contents of all locations where the quantity or effective address is located.
- The user can reference symbols in one address space from another address space, and can reference local symbols within a particular program. The user can also assign values to symbols and restrict or eliminate the use of defined symbols.

1.1.2 COMMANDS

I4DDT's operations are controlled by commands typed in by the user. Each command consists of one or more symbols or characters and, in many cases, requires a user-supplied parameter (normally an absolute or symbolic address). For example, the user may examine the contents of any memory location by typing the address of the desired location followed immediately by a slash (/). To type out the contents of a location whose symbolic address is CAT, the user types:

```
CAT/
```

I4DDT now types out the contents (preceded and followed by spaces) on the same line:

```
          CAT/ MOVE AC,DOG
```

The location is now opened; once opened, the user may change its contents by typing in the desired new contents and a carriage return immediately following I4DDT's typeout. For example:

```
          CAT/ MOVE AC,DOG MOVE AC2,DOG+3↵
```

All I4DDT commands are described in detail in Section 3.

SECTION 2

USING I4DDT

2.1 OPERATING PROCEDURES

The user loads the I4DDT program into PDP-10 central memory; the program to be debugged is loaded into the appropriate address space (see Section 2.2.1 and 3.1.1). The procedures are as follows:

1. Log in to the TENEX Operating System (refer to the TENEX Executive Language manual).
2. Load and start I4DDT (type I4DDT).
3. Select appropriate address space.
4. Use I4DDT to load the program to be debugged (type ;L, ;;L, or ;G). These commands are described in Section 3.1.3.

2.2 EXECUTIVE FUNCTIONS

I4DDT controls the execution of programs to be tested which run, under I4DDT, as inferior forks. Since I4DDT is the superior fork in the structure, it simulates many functions normally performed by an executive. The I4DDT commands that invoke these functions are described in Section 3.1. The I4DDT executive command mode is entered by typing a semicolon (;) as the first character of the command.

One set of executive commands is used to select an address space, i.e., specify where the program to be debugged resides. The user can select any of five address spaces: PDP-10, PDP-11, ILLIAC array, ILLIAC disk, and Test Maintenance Unit (TMU). When the user selects the TMU address space, the object program is actually loaded into PDP-10 memory; i.e., TMU programs occupy PDP-10 address space. Features and characteristics of the TMU address space are described in Appendix E.

Another set of executive commands enables the user to move the contents of files into and out of address spaces and to perform various file manipulation functions. Other executive commands are available that enable the user to perform miscellaneous functions such as selecting a new escape character and changing page protection.

2.2.1 SELECTING ADDRESS SPACES AND I/O MODES

When I4DDT is loaded, the user must issue a command to select an address space, i.e., specify the instruction and word formats of the program to be executed. Normally the address space specified corresponds to the machine in which the program executes. The user may, however, interpret locations in one address space as if they were in another. The input/output mode commands enable the user to select any combination of address space and mode (see Section 3.1.2).

Address spaces and input/output modes may be permanently or temporarily selected. A permanent mode is in effect until it is replaced with another permanent mode or until I4DDT is reloaded. A temporary mode is in effect until the user types a carriage return or re-enters I4DDT.

2.2.2 FILE OPERATIONS

The I4DDT executive commands described in Section 3.1.3 enable the user to perform the following file manipulation functions:

- Load a file into an address space.
- Load a symbol table into an address space.
- Move the contents of an address space to a file.
- Merge a file with what is already in an address space.
- Bring the loader into the current address space.
- Execute I4DDT command strings from a file.

2.3 INTERNAL REGISTERS

A distinct set of internal registers is associated with each address space in I4DDT. When the current address space is changed to a new address space by an appropriate I4DDT command, the corresponding set of internal registers is automatically mapped into the correct locations. It should be noted that some internal registers are only meaningful within certain address spaces, and others are only meaningful to I4DDT and are of no concern to the user.

The user may examine and explicitly set the internal registers as if they were in his address space; the registers are also set upon the execution of certain commands.

All internal registers begin with an ampersand (&) and contain the values shown below.

REGISTER	CONTENTS
&nB	Address of breakpoint n.
&nB+1	Conditional breakpoint instruction at breakpoint n.
&nB+2	Proceed count at breakpoint n (number of times to proceed past the breakpoint).
&nB+3	String pointer at breakpoint n.
&nB+4	Saved instruction while running user's program.
&nB+5	-1 if automatic proceed mode; otherwise zero.
&nB+6	ASCII name assigned to this breakpoint by I4DDT (usually &nB).
&nB+7	Trace value. If -1, trace location zero in the user fork. If the left half is -1, the value of the current location is added to the trace value. The trace value may also be an internal register.
&C	The radix (2, 8, 10, or 16) used for the contents of a location in numeric typeouts (see Section 3.2).
&DDT	Location zero in I4DDT memory.

REGISTER	CONTENTS
&F	Bit size (32, 36, or 64) used in floating-poing typeouts (see Section 3.2).
&FH	Fork name used by I4DDT to reference the user.
&ICA	Interrupt channels assigned for user.
&ICB	Interrupt channels with breaks waiting.
&IS	Zero if user's interrupt system is off; otherwise nonzero.
&L	Left half of last quantity typed.
&LL	Lower limit used for searches (see Section 3.10).
&LMB	Lower memory boundary used in moving a TMU address space to a file.
&LS	Left half of last quantity typed (swapped extended).
&M	Mask used in searches (see Section 3.10).
&M+1	High-order word of mask.
&O	Byte size (1-36) used in typeouts (see Section 3.2).
&PC	The user's Program Counter, i.e., the address of the next instruction to be executed.
&PL	Patch location, i.e., location where patch instructions are placed.
&PTS	Pager trap status word at memory violation (saved state information).
&PWD	Pager write data at memory violation (saved state information).
&Q	Last quantity typed.
&QS	Last quantity typed (halves swapped).
&R	The radix (2, 8, 10, or 16) used for the address of a location in numeric typeouts (see Section 3.2).
&S	Instruction class used in symbolic typeouts (see Section 3.2), where 0=PDP-10, 1=short ILLIAC, 2=long ILLIAC, 3=TMU, 4=PDP-11.

REGISTER	CONTENTS
&T	Text mode used in typeouts (see Section 3.2), where 5=Radix-50, 6=Sixbit text, 7=7-bit ASCII, 8=8-bit ASCII.
&UL	Upper limit used for searches (see Section 3.10).
&UMB	Upper memory boundary used in moving a TMU address space to a file.
&UN	Saved user subsystem name.
&V	Current register pair used in field typeouts — default 1 (see Section 3.2).
&nVB	Boundary bits used in field typeouts (n=current register pair).
&nVB1	High-order word of field boundary bits (n=current register pair).
&nVM	Mask bits used in field typeouts (n=current register pair).
&nVM1	High-order word of field mask bits (n=current register pair).
&W	Word length used in typeouts (see Section 3.2).
&X	The address of the first of the four words used by the Execute command ($X). This register initially contains 777774 so that the top four words are used.
•	Current location.
@	Indirect bit used to reference an effective address in a MACRO=10 statement.

2.4 TYPEIN FORMATS

To change the contents of a memory location, the user may type in symbolic instructions (PDP-10 or ILLIAC IV), numbers (octal integers, hexadecimal integers, fixed or floating-point decimal), and text characters (ASCII, SIXBIT, or Radix 50). I4DDT will test the data typed in to determine how it should be interpreted, i.e., to determine the typein format. Therefore, the user must follow the syntax rules described below when typing in changes. The typein format is not affected by the typeout format settings described in Section 2.5.

2.4.1 SYMBOLIC INSTRUCTIONS

2.4.1.1 PDP-10 Instructions

PDP-10 instructions are typed in by the user in the same format used in writing a MACRO-10 assembler language source program statement. For example, in the following MACRO-10 statement,

```
ADD AC1,DATE
```

a space terminates the operation field, and a comma terminates the accumulator field. The operation code determines the interpretation of the accumulator field. If an I/O instruction is used, I4DDT inserts the I/O device number in the correct place; indexed and indirect addresses are written with an at sign (@) preceding the address to set the indirect bit, and the index register in parentheses. For example:

```
ADD 4,@NUM(17)
```

To type in two halfwords, the left and right expressions are separated by two commas. For example:

```
-6,,BEGIN-I
```

2.4.1.2 ILLIAC IV Instructions

ILLIAC IV instructions are typed in by the user with the same mnemonic operation codes used in writing an ASK assembler language source program statement. However, significant syntactic differences exist between I4DDT's ILLIAC instructions and ASK instructions. These differences are described in Appendix F.

2.4.2 NUMBERS

The user may type in octal integers, or fixed or floating-point decimal integers. A typed-in number that does not contain a decimal point is always interpreted as octal. The following are examples of octal typeins:

```
1234
-10101
772
777777777777
```

Fixed point decimal integers must contain a decimal point with no digits following. For example:

```
1234.
-99.
877.
```

Floating-point numbers may be written in either of two formats:

(1) With a decimal point and one or more digits following the decimal point, e.g.,

```
101.1
999.0
-2.71828
```

(2) In fixed point notation, with E indicating exponentiation, e.g.,

```
12.0E+2
12.34E2
31.4159E-1
```

2.4.3 TEXT CHARACTERS

The user may type in any number of ASCII or SIXBIT characters, and up to six redix-50 characters.

To type in a single ASCII character, right-justified in an opened word, the user types a quotation mark, followed by a single ASCII text character, then by an ALTMODE ($). For example:

```
"0$
"/$
"?$
```

To type in more than one ASCII character, left-justified in consecutive locations,the user types a quotation mark, followed by an arbitrary delimiting character (any non-blank printing character not used in the text), then the text characters, terminated by the delimiting character. For example:

```
"/TEXT/ (the slash is the delimiting character)
"ABCDEFA (the letter A is the delimiting character)
"/VERY LONG TEXT STRINGS CAN BE ENTERED/
```

If a location is not open, I4DDT types out the delimiting character when the fifth character is typed, and no other characters can be entered.

To type in a single SIXBIT character, right-justified in an opened word, the user types an ALTMODE ($), followed by a quotation mark, then by a single SIXBIT character, terminated by an ALTMODE. For example:

```
$"Q$
$"M$
$"$$
```

To type in more than one SIXBIT character, left-justified in consecutive locations, the user types an ALTMODE ($) and a quotation mark, followed by any delimiting character, then the text characters, terminated by the delimiting character. For example:

```
$"/DIVIDE/ (the slash is the delimiting character)
$"EXXXXXXE (the letter E is the delimiting character)
```

If a location is not open, I4DDT types out the delimiting character when the sixth character is typed, and no other characters can be entered.

To type in up to six radix-50 characters (in a PDP-10 fork), the user types the characters followed by ↑A. For example:

```
100/ 0 XYZ↑A
$5T100/ OXYZ (set typeout format to radix-50)
```

2.5 TYPEOUT FORMATS

When I4DDT is loaded, it is initially set to type out the contents of locations in symbolic instruction format, with addresses relative to symbolic locations. However, it may often be desirable to change the typeout format depending upon the information that is stored at a particular location. For example, if numeric data is stored in a location, a symbolic typeout would be meaningless; instead, the user may wish to have I4DDT type out 7-bit ASCII text characters, SIXBIT text characters, floating-point numbers, etc. Or, it may be more useful to examine an instruction by having the address parts typed out as absolute addresses rather than as relative to symbolic locations. Therefore, a set of commands is available that enables the user to change the typeout format at any time during the debugging session. These commands are described in detail in Section 3.2.

In addition to having the ability to change the typeout format, the user also can specify how long the change is to be in effect. The format change may have one of the following durations:

1. Prevailing or semipermanent. The user sets a prevailing format change by preceding the typeout format command with two ALTMODE's. A prevailing format is changed by replacing it with another prevailing format or by reinitializing the system.

2. Temporary. The user sets a temporary format change by preceding the typeout format command with a single ALTMODE. Temporary formats remain in effect until the user types a carriage return or reenters I4DDT.

TABLE 2-1. TYPEIN/TYPEOUT FORMATS IN VARIOUS I/O MODES

TYPEIN FORMATS

Format	Syntax	Bits	PDP-10 (;$P)	Bits	Short Illiac (;$S)	Bits	Long Illiac (;$L)	Bits	TMU (;$T)
Symbolic	opcode operand	36	PDP-10 opcodes	32	Illiac 4 opcodes	32	Illiac 4 opcodes	48	TMU opcodes
Halfword	lefthalf,,righthalf	36	Two halfwords	2x32	32-bit mode integers	2x32	32-bit mode integers	16,32	16- and 32-bit integers
Octal	n	36	Octal	32	Single word octal	64	Double word octal	48	48-bit octal
Decimal	n.	36	Decimal	32	Single word decimal	64	Double word decimal	48	48-bit decimal
Hexadecimal	!n!	36	Hexadecimal	32	Single word hexadecimal	64	Double word hexadecimal	48	48-bit hexadecimal
Text	"/TEXT/	35	7-bit left*	32	8-bit left*	64	8-bit left*	42	8-bit left*
Character	"c$	36	7-bit right	32	8-bit right, single	64	8-bit right, double	48	8-bit right
Sixbit character	$"c$	36	6-bit right	30	6-bit right	60	6-bit right	48	6-bit right
Sixbit text	$"/TEXT/	36	6-bit left*	30	6-bit left*	60	6-bit left*	48	6-bit left*
Radix-50 symbol	SYMBOL↑A	32	Radix-50, tag field zero	32	Radix-50 symbol	32	Radix-50 symbol (in second word)	32	Radix-50 right
Floating point	n.n or n.nEn	36	PDP-10 floating	64	Illiac floating	64	Illiac floating	32	32-bit right
32-bit mode	n.nEn,,n.nEn	2x32	32-bit mode floating	2x32	32-bit mode floating	2x32	32-bit mode floating	2x32	32-bit mode floating

TYPEOUT FORMATS

Format	Syntax	Bits	PDP-10 (;$P)	Bits	Short Illiac (;$S)	Bits	Long Illiac (;$L)	Bits	TMU (;$T)
Symbolic	$S or $nS	36	PDP-10 opcodes	32	Illiac 4 opcodes	32	Illiac 4 opcodes	48	TMU opcodes
Numeric	$C or $nC	36	Current radix	32	Current radix	64	Current radix	48	Current radix
Floating point	$F or $nF	36	PDP-10 floating	32	Illiac floating	64	Illiac floating	32	Illiac floating
Text	$7T	35	7-bit text	32	8-bit text	64	8-bit text	42	8-bit text
Sixbit text	$6T	36	6-bit text	30	6-bit text	60	6-bit text	48	6-bit text
Radix-50	$5T	32	Radix-50	64	Radix-50	64	Radix-50	32	Radix-50 text
Halfwords	$H	36	Halfwords 2x18	64	Integers 2x32	64	Integers 2x32	16,32	Two integers 16 and 32 bits
Bytes	$O or $nO	36	Bytes	32	Bytes	64	Bytes	48	Bytes

*As many words as are required.

TS IN VARIOUS I/O MODES

ort Illiac (;$S)	Bits	Long Illiac (;$L)	Bits	TMU (;$T)
liac 4 opcodes	32	Illiac 4 opcodes	48	TMU opcodes
-bit mode integers	2x32	32-bit mode integers	16,32	16- and 32-bit integers
ngle word octal	64	Double word octal	48	48-bit octal
ngle word decimal	64	Double word decimal	48	48-bit decimal
ngle word hexadecimal	64	Double word hexadecimal	48	48-bit hexadecimal
bit left*	64	8-bit left*	42	8-bit left*
bit right, single	64	8-bit right, double	48	8-bit right
bit right	60	6-bit right	48	6-bit right
bit left*	60	6-bit left*	48	6-bit left*
dix-50 symbol	32	Radix-50 symbol (in second word)	32	Radix-50 right
liac floating			32	32-bit right
-bit mode floating	64	Illiac floating	2x32	32-bit mode floating
	2x32	32-bit mode floating		

liac 4 opcodes	32	Illiac 4 opcodes	48	TMU opcodes
rrent radix	64	Current radix	48	Current radix
liac floating	64	Illiac floating	32	Illiac floating
bit text	64	8-bit text	42	8-bit text
bit text	60	6-bit text	48	6-bit text
dix-50	64	Radix-50	32	Radix-50 text
tegers 2x32	64	Integers 2x32	16,32	Two integers 16 and 32 bits
tes	64	Bytes		
			48	Bytes

2.6 SYMBOLS

I4DDT allows the user to create defined or undefined symbols, to restrict or eliminate the use of symbols, and to reference symbols in other programs.

A symbol in I4DDT contains from one to six characters from the following set:

- 26 letters, A-Z
- Ten digits, 0-9
- Three special characters: $(dollar sign)
%(percent)
.(period)

If the symbol contains numerals and only one letter, that letter must not be B, D, or E . These letters are reserved for binary-shifted and floating-point numbers. A symbol may actually have more than six characters, but characters after the sixth are ignored.

I4DDT accepts both lower and upper case letters. Lower case letters are internally converted to upper case; except when they are used within strings.

2.6.1 DEFINING AND REFERENCING SYMBOLS

I4DDT includes a set of commands that enable the user to insert a symbol in the symbol table and assign it a specified value, to delete a symbol from the symbol table or restrict its use, and to reference symbols in other programs. These commands are described in Section 3.9.

Symbols that can be referenced in one program from another are called global symbols, e.g., symbols defined by the MACRO-10 statements INTERN or ENTRY. Symbols that can be referenced only within the program in which they are defined are called local symbols.

I4DDT permits the user to reference all global and local symbols. However, to reference a local symbol that appears with different values in different programs, the user must first type the command

```
name$:
```

where name is the name of the desired program in which the symbol is defined, followed by an ALTMODE and a colon. For example, the command

```
GEORGE$:
```

"unlocks" the symbol table associated with program GEORGE; the user can subsequently reference all local symbols in program GEORGE until another name$: command is given, where name represents another program whose symbol table the user wishes to access.

The user can insert (or redefine) a symbol in the symbol table by typing a symbol, followed by a colon. The symbol will have a value equal to the address of the location pointer (.). For example,

```
400/ ADD 3,N TAG:
```

puts the symbol TAG in I4DDT's symbol table and assigns it the value 400, the address of the last location opened.

The user can also directly assign a value to a symbol by typing the value, a left angle bracket (<) and the symbol, terminated by a colon. For example:

```
305<TAG:
```

TAG is now defined to have the value 305.

Undefined symbols may be used in a program by following the symbol with a number sign (#). The symbol will then be saved by I4DDT, and subsequent uses of the symbol will cause I4DDT to type out #. When an undefined symbol is assigned a value, all previous occurrences of the symbol will be replaced with the defined value. Undefined symbols may be used only in the address field, and only in operations involving addition or subtraction.

2.6.2 SYMBOL TABLES

When the user types in a symbol, I4DDT attempts to evaluate it by searching the following two symbol tables in order:

1. A built-in operation table containing the machine language instructions and unimplemented operation codes (UUOs).
2. A symbol table constructed by the loader during the loading process that contains all the user-defined symbols.

2.7 EXPRESSIONS

2.7.1 FORMING EXPRESSIONS WITH ARITHMETIC OPERATORS

I4DDT permits the user to combine symbols and numbers into expressions by using the following characters to indicate arithmetic operators.

Operator	Meaning
+	Two's complement integer addition
-	Two's complement integer subtraction
*	Integer multiplication
'	Integer division with remainder discarded

Expressions are evaluated from left to right, with multiplication and division performed first, followed by addition and subtraction. The following examples show how expressions may be formed:

AC1+7, RHO'3

A-3, BETA*5

2.7.2 EXPRESSION EVALUATION

Parentheses are used to denote an index field or to interchange the left and right halves of an expression inside the parentheses. If an arithmatic operator immediately precedes the left parenthesis, the expression is treated as a term in the larger expression being assembled. If an arithmetic operator does not immediately precede the left parenthesis, the swapped quantity within the parentheses is added to the storage word being generated. For example, in a PDP-10 program, the expression (17) is effectively 000017000000_8. This 36-bit quantity is added to the word.

2.8 BREAKPOINTS

Breakpoint instructions are used to help the user monitor the progress of the program being executed. Up to eight breakpoints may be set subject to the restrictions described in Section 2.8.5. When a breakpoint is set, I4DDT replaces the contents of the breakpoint location with a trap instruction so that when the program is executed and the breakpoint is encountered, program execution is suspended, the state of the user's program is saved, and the programmer's original instructions are restored to the breakpoint location. I4DDT types out the number of the breakpoint, a symbol indicating the reason for the break, and the address at which the break occurred. (The breakpoints are automatically restored when execution is resumed.)

When the user sets a breakpoint, he may request that the contents of a word be typed out when the breakpoint is reached. When the program stops, the user may then enter other commands to examine and debug his program.

Breakpoints may be set either unconditionally or conditionally.

When an unconditional breakpoint is set, the program will automatically stop when the breakpoint address is reached (but before the instruction at the breakpoint address is executed).

Conditional breakpoints may be set in two ways. The user may provide his own instruction or subroutine to determine whether to stop, or he may set a proceed counter to specify the number of times a breakpoint is passed before a program stop occurs. Each of the eight breakpoints has seven registers associated with it. Three of these registers contain the breakpoint address, the conditional breakpoint instruction (if any), and the proceed counter. Thus, to set a conditional breakpoint, the user must first insert the instruction or proceed count in the proper registers. These registers are described below in Section 2.8.1.

To set a breakpoint, the user types the symbolic or absolute address of the location at which he wants the program to stop, followed by $B. For example:

```
6004$B
```

The user may also assign a number to the breakpoint by typing $nB, where n is the breakpoint number. For example:

```
CAT$4B
```

If no number is specified, I4DDT will assign breakpoint numbers in sequence from 1 to 8.

To have I4DDT type out the contents of a word when the breakpoint is reached, the user types the address of the word to be examined, followed by a left angle bracket (<) before the breakpoint address. For example:

```
DOG <CAT$4B
```

All of I4DDT's breakpoint commands are described in detail in Section 3.8.

2.8.1 BREAKPOINT REGISTERS

The user may set a conditional breakpoint or check the status of a breakpoint by setting or examining the following registers (where n is the breakpoint number):

REGISTER	CONTENTS
&nB	Address of the breakpoint.
&nB+1	The conditional breakpoint instruction.
&nB+2	Proceed count.
&nB+7	Trace location.

When a breakpoint location is reached, I4DDT enters its breakpoint analysis routine (see Figure 2-1). This routine consists of the following instructions:

```
SKIPE      &nB+1              Is the conditional break instruction 0?
XCT        &nB+1              No, execute conditional break instruction
SOSG       &nB+2              Decrement and test proceed counter
JRST       (Break routine)
JRST       (Proceed routine)
```

2.8.2 CONDITIONAL BREAKPOINTS

To set a conditional breakpoint, the user must insert a conditional instruction or a subroutine call in register &nB+1. He does this by entering an I4DDT command to change the contents of the word. For example, to insert a conditional instruction at breakpoint 3:

```
&3B+1/ 0  CAIGE ACC,15↵
```

When the breakpoint is reached, this instruction is executed.

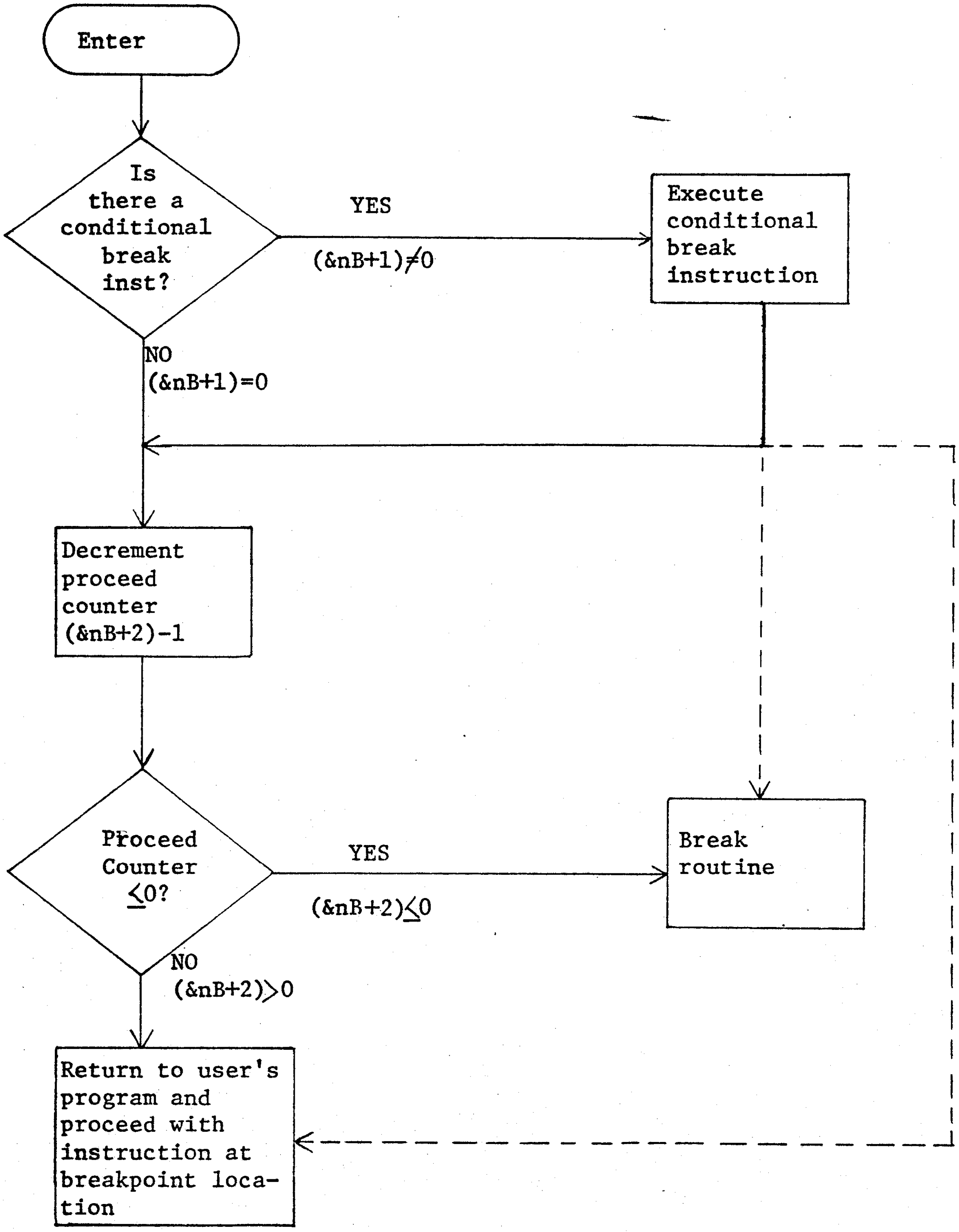
Enter
Is there a conditional break inst?
YES
(&nB+1)≠0
Execute conditional break instruction
NO
(&nB+1)=0
Decrement proceed counter (&nB+2)-1
Proceed Counter ≤0?
YES
(&nB+2)≤0
Break routine
NO
(&nB+2)>0
Return to user's program and proceed with instruction at breakpoint location

Figure 2.1. I4DDT's Breakpoint Analysis Routine

If the instruction does not skip or the subroutine returns to the next sequential location in the breakpoint analysis routine, the proceed counter will be decremented and tested. Therefore, if the user wishes a break to occur based solely on the conditional instruction, he should set the proceed counter to a large positive number so that the proceed counter will never reach zero.

If the instruction or subroutine causes one instruction to be skipped, i.e., that which decrements and tests the proceed counter, the program breaks.

If the conditional break instruction transfers to a subroutine which returns skipping over two instructions, a break will never occur regardless of the proceed counter, and the user's program proceeds with the instruction at the breakpoint address.

Conditional breakpoints cannot be set in the ILLIAC address space.

2.8.3 THE PROCEED COUNTER

If the user wishes to proceed past a breakpoint a specified number of times, and then stop, he inserts the number of passes in register &nB+2, which contains the proceed count. When the breakpoint location is reached, I4DDT will decrement and test the proceed counter. If it is less than or equal to zero, a break occurs; if it is greater than zero, execution of the user's program proceeds with the instruction where the break occurred.

The proceed counter may be set in two ways:

1. Insert the count directly, e.g., &nB+2/ <u>0</u> 20 sets the counter to 20.
2. After stopping at a breakpoint, set or reset the counter by typing the count before the proceed command, e.g., 20$P.

2.8.4 BREAKPOINT TYPEOUTS

When the breakpoint location is reached, I4DDT types out the following:

- The breakpoint number, assigned by I4DDT or explicitly by the user in the form &nB.
- A symbol indicating the reason for the break:

  >for the conditional break instruction,

  >>for an unconditional break or for the proceed counter being equal to zero.
- The Program Counter value at the time the breakpoint is reached, i.e., the address in the user's program where the break occurred.
- The contents of an address (if any) specified by the user to be typed out when the breakpoint is reached.

For example, assume that the user has set an unconditional breakpoint as follows:

```
DOG <CAT$3B
```

I4DDT will type out the following when breakpoint 3 is reached:

```
&3B >> CAT DOG/ SOJGE 3,ALPHA
```

where CAT is the breakpoint address and location DOG contains the instruction SOJGE 3,ALPHA.

2.8.5 RESTRICTIONS

The locations where breakpoints are set may not

a. be modified by the program.

b. be used as data or literals.

c. be used as part of an indirect addressing chain (PDP-10 address space).

d. contain the user mode monitor command INIT (PDP-10 address space).

e. be accumulator 0.

2.9 ERROR CORRECTION AND RECOVERY

2.9.1 DELETING TYPING ERRORS

The user can delete any partially typed command by pressing the Control E (↑E) key. This causes I4DDT to ignore the unexecuted command, suspend program operation, and type out the following message:

XXX:addr/inst

where addr is the address of the next instruction to be executed, and inst is the instruction at that address. For example, if the following message is typed:

XXX:LOC+5/ MOVE A,DAT+21

the interrupt will have occurred immediately before this instruction, and if a $P command is typed to restart the program, this instruction will be the next one executed by the user.

2.9.2 ERROR MESSAGES

Appendix B contains the error messages typed out by I4DDT, their causes, the recovery procedures.

2.10 CONTROL CHARACTERS

Control characters are input by depressing the CTRL key in conjunction with the character. The notation for the control key used in this manual is the symbol ↑. For example, ↑B is input by simultaneously depressing the CTRL key and the B key.

Six control characters are usable under I4DDT: ↑B, ↑E, ↑F, ↑H, ↑T, and ↑Z. The functions of these characters are explained below.

CHARACTER	DESCRIPTION
↑B	Type a report on core memory assigned to the user, i.e., number of pages, location and length of entry vector, memory map (page numbers with associated file names and type of access allowed — read, write, or execute).
↑E	Delete a partially typed command (may be replaced by using the ;;A command — see Section 3.1.4).
↑F	Type a report on all inferior forks under I4DDT.
↑H	Change the contents of a location and subtract 1 from the location pointer (see Section 3.6).
↑T	Alternate escape character (may be replaced by using the ;A command — see Section 3.1.4).
↑Z	If the last quantity typed is zero, ignore all characters typed in until the next ↑Z; if the last quantity typed is non-zero, ignore the next ↑Z. (Used in I4DDT command files to ignore or execute selected commands.)

SECTION 3

I4DDT COMMANDS

3.1 EXECUTIVE FUNCTIONS

3.1.1 ADDRESS SPACE SELECTION

The following commands enable the user to select the following address spaces: PDP-10, PDP-11, ILLIAC IV array, ILLIAC IV disk, or TMU. These address spaces may be selected permanently (in effect until replaced by another permanent address space command or the system is reinitialized), or temporarily (in effect until the user types a carriage return or re-enters I4DDT). Permanent address space commands are preceded by two semicolons (;;). Temporary address space commands are preceded by one semicolon (;).

Options

a. Select permanent PDP-10 address space.

   Syntax: ;;P

b. Select temporary PDP-10 address space.

   Syntax: ;P

c. Select permanent ILLIAC IV array address space.

   Syntax: ;;I

d. Select temporary ILLIAC IV array address space.

   Syntax: ;I

e. Select permanent ILLIAC IV disk address space.

   Syntax: ;;D

f. Select temporary ILLIAC IV disk address space.

   Syntax: ;D

g. Select permanent PDP-11 address space.

Syntax: ;;E

h. Select temporary PDP-11 address space.

Syntax: ;E

i. Select permanent TMU address space.

Syntax: ;;T

j. Select temporary TMU address space.

Syntax: ;T

3.1.2 INPUT/OUTPUT MODE SELECTION

The following commands enable the user to interpret locations in one address space as if they were in another address space. Regardless of the address space he has selected, the user may specify a different input/output mode, e.g., he may wish to examine ILLIAC instructions in a PDP-10 address space. Any combination of address space and input/output mode may be selected, although many combinations will have little practical use.

Input/output modes are selected by entering one or two semicolons and an ALTMODE ($), followed by the address space selection letter. As with the address space commands, input/output modes may be selected permanently or temporarily.

Options

a. Select permanent PDP-10 mode.

Syntax: ;;$P

b. Select temporary PDP-10 mode.

Syntax: ;$P

c. Select permanent short ILLIAC IV mode (32 bits).

Syntax: ;;$S

d. Select temporary short ILLIAC IV mode (32 bits).

Syntax: ;$S

e. Select permanent long ILLIAC IV mode (64 bits).

Syntax: ;;$L

f. Select temporary long ILLIAC IV mode (64 bits).

Syntax: ;$L

g. Select permanent PDP-11 mode.

Syntax: ;;$E

h. Select temporary PDP-11 mode.

Syntax: ;$E

i. Select permanent TMU mode (48 bits).

Syntax: ;;$T

j. Select temporary TMU mode (48 bits).

Syntax: ;$T

k. Select permanent short ILLIAC disk mode.

Syntax: ;;$D (equivalent to ;;$S)

l. Select temporary short ILLIAC disk mode.

Syntax: ;$D (equivalent to ;$S)

m. Select permanent short ILLIAC array mode.

Syntax: ;;$I (equivalent to ;;S)

n. Select temporary short ILLIAC array mode.

Syntax: ;$I (equivalent to ;S)

3.1.3 FILE OPERATIONS

The following executive commands enable the user to load the contents of files into address spaces and to move the contents of address spaces into files.

Options

a. Load a file and the symbol table into the current address space.

Syntax: ;G GET FILE: filename

where filename is the name of a SAVE CORE file, created by the TENEX SAVE command (refer to the TENEX Executive Language Manual).

b. Load a new symbol table without changing the contents of the address space.

Syntax: ;;G GET SYMBOL FILE: filename

where filename is the name of a SAVE file.

c. Move the contents of the current address space to a file.

Syntax: ;S SAVE FILE: filename

where filename is the name of a SAVE file.

NOTE: If the current address space is the TMU, pages specified by &LMB to &UMB will be moved to the designated file.

d. Move a symbol table to a file.

Syntax: ;;S SAVE SYMBOL FILE: filename

where filename is the name of a SAVE file.

e. Merge a file with what is already in the current address space.

Syntax: ;M MERGE FILE: filename

where filename is the name of a SAVE file.

f. Merge a symbol table with what is already in the current address space.

Syntax: ;;M MERGE SYMBOL FILE: filename

where filename is the name of a SAVE file.

g. Bring the PDP-10 loader into the current address space and return control to I4DDT.

Syntax: ;L

h. Bring the PDP-10 loader into the current address space and start the user's program at a previously specified starting address.

Syntax: ;;L

i. Execute I4DDT command strings from a file.

Syntax: ;X EXECUTE FROM FILE: filename

where commands will be executed from the specified file (default extension .CMD) until end of file is reached or a ;X in the file returns input to the terminal. Execute files may be nested and/or recursive to any level.

3.1.4 MISCELLANEOUS EXEC FUNCTIONS

a. Insert the lower limit into register &LL and the upper limit into register &UL, and zero memory using these limits. Do not zero the symbol table.

Syntax: a<b;Z

where a is the lower limit and b is the upper limit. If a is omitted, the current value of &LL will be used; if b is omitted, the current value of &UL will be used.

b. Zero memory and the symbol table within specified limits.

Syntax: a<b;;Z

where a and b have the values described above.

c. Change protection on pages a through b inclusive according to the three-bit value of n.

Syntax: a<b;nU

where n=4 means allow read access, n=2 means allow write access, and n=1 means allow execute access. n values may be summed to specify combinations of access, e.g., n=6 means allow read and write access. If n is omitted, it is assumed to be 7.

Example: 50<77;6U

where the protection on pages 50 through 77 has been changed to allow read and write access.

d. Change protection on page a.

Syntax: a;nU

where the bit values of n are as described above.

e. Change protection on current page.

Syntax: ;nU

where the bit values of n are as described above.

f. Open access to ILLIAC and copy virtual ILLIAC image to ILLIAC memory.*

Syntax: ;0

g. Open access to ILLIAC. *

Syntax: ;;0

h. Copy ILLIAC memory to virtual ILLIAC image and close access to ILLIAC.

Syntax: ;C

i. Close access to ILLIAC.

Syntax: ;;C

*If the ILLIAC is busy, I4DDT will type out a message indicating who the current user is, and will continue trying to acquire ILLIAC.

j. Replace ↑T with a new escape character.

Syntax: ;A ESCAPE CHR IS: c

where c is the new escape character (c must be ≤ ↑Z).

k. Replace ↑E with a new escape character.

Syntax: ;;A ESCAPE CHR IS: c

where c is the new escape character (c must be ≤ ↑Z).

3.2 CHANGE TYPEOUT FORMAT

The Change Typeout Format commands are used to change the format in which memory contents are typed out by I4DDT, i.e., as symbolic instructions, numeric constants, etc. When I4DDT is loaded, the typeout formats are initialized to output symbolic instructions with addresses relative to symbolic locations. For numeric typeouts, the radix is initially set to octal. Table 3-1 shows the default values for various typeout formats.

The user also sets the duration of a typeout format to be either prevailing (in effect until it is changed to another prevailing format or the system is reinitialized), or temporary (in effect until the user types a carriage return or reenters I4DDT). Prevailing format commands are preceded by two ALTMODES ($$); temporary format commands are preceded by one ALTMODE($).

In all I4DDT typeouts, leading zeros are suppressed.

Options

a. Type out addresses relative to symbolic locations (I4DDT's initial setting).

Syntax: $R or $$R

Example: $R addr/ ADD TABL+14,TAB:+13

b. Type out the address parts of instructions, and both addresses when the format is halfword, as absolute numbers in the current radix.

Syntax: $A or $$A

Example: $A addr/ ADD 4002

c. Type out symbolic instructions (I4DDT's initial setting) and insert n (the instruction class) into register &S.

Syntax: $nS or $$nS

where n=0 (PDP-10), 1 (short ILLIAC), 2 (long ILLIAC), 3 (TMU), or 4 (PDP-11).

Example: $0S addr/ AC1,TABLE+3

d. Type out symbolic instructions, with the instruction class determined by the current contents of register &S (see the values of n, above).

Syntax: $S or $$S

e. Type out the contents of locations as constants, i.e., as numbers in the current radix, and insert n (the radix) into register &C.

Syntax: $nC or $$nC

where n=2, 8, 10, or 16. If the radix is octal, the location typed out will be divided into halves, separated by two commas.

f. Type out the contents of locations as constants, with the radix determined by the current contents of register &C.

Syntax: $C or $$C

g. Type out memory location addresses using radix n.

Syntax: $nR or $$nR

h. Type out the contents of locations as floating-point numbers, and insert n (the bit length) into register &F.

Syntax: $nF or $$nF

where n=32, 36, or 64.

i. Type out the contents of locations as 32-, 36-, or 64-bit floating-point numbers, depending on the current contents of register &F.

Syntax: $F or $$F

j. Type out the contents of locations as text, and insert n (the text type) into register &T.

Syntax: $nT or $$nT

where n=8 (8-bit ASCII), 7 (7-bit ASCII), 6 (SIXBIT text), or 5 (Radix-50).

k. Type out the contents of locations as text (7- or 8-bit ASCII, SIXBIT, or Radix-50) as determined by the current contents of register &T.

Syntax: $T or $$T

l. Type out the contents of locations as n-bit bytes, and insert n in register &O.

Syntax: $nO or $$nO

where n is a value from 1 to 64.

m. Type out the contents of locations as n-bit bytes, where n is determined by the current contents of register &O.

Syntax: $O or $$O

n. Type out the contents of locations as n-bit bytes, where n is half the value of the word-length register (&W).

Syntax: $H or $$H

Table 3-1. Default Values for Typeout Formats

Register	Contents	Default Values				
		PDP-10	Array	I4 Disk	PDP-11	TMU
&S (instruction)	0,1,2,3, or 4	0	1	1	4	3
&C (data radix)	2,8,10, or 16	8	8	8	8	8
&T (text)*	5,6,7, or 8	7	8	8	7	8
&F (floating point)	32,36, or 64	36	64	64	32	64
&O (byte)	1 — 36	18	8	8	8	8
&W (word length)	16,32,36, or 64	36	32	32	16	48
&R (address radix)	2,8,10, or 16	8	8	8	8	8

*5 = Radix-50

6 = SIXBIT Text

7 = 7-bit ASCII

8 = 8-bit ASCII

o. Type out the contents of locations in a field format specified by boundary register &nVB and mask register &nVM, where n (the current register pair) is contained in register &V.

Syntax: $V or $$V

Example: See Section 3.2.1.

p. Type out the contents of locations in a field format specified by boundary register &nVB and mask register &nVM, and insert n (the current register pair) into register &V.

Syntax: $nV or $$nV

Example: See Section 3.2.1.

3.2.1 FIELD FORMAT TYPEOUTS

The field typeout format enables the user to examine the contents of a location divided into a number of fields.

Eight pairs of registers are associated with field typeouts, and one register (&V) contains the number of the current pair to be used when the field typeout format is selected.

Each of the eight pairs consists of a boundary register (&nVB) and a mask register (&nVM), where n is a number from 1 to 8 designating the current pair in register &V. The default value in register &V is 1.

Each bit set in the boundary register specifies the right-most bit of a field. For example, the following bit configuration:

```
0010001
```

would specify a 3-bit field starting at bit position 0 and a 4-bit field starting at bit position 3.

Each bit in the mask register is ANDed with the corresponding bit in the memory location being examined to form the result (see Example 1).

Example 1:

```
;;P                             Select PDP-10 address space.

&2VB/ 0 40,,200011              Set boundary bits using register pair 2.

&2VM/ 0 17743,777771            Set mask bits.

20/ 133104,,325                 Examine location 20 (current typeout format
                                is octal).

$$2C                            Set typeout format to binary.

20/ 1011011001000100,,11010101  Examine location 20.

$$2V                            Set typeout format to field, using

                                register pair 2.

20/ 1011001,0,11010,1           Examine location 20 in field format.

&C/ 2 10                        Change typeout radix to decimal.

20/ 131,0,32,1                  Examine location 20.
```

The user may also set field boundary bits by typing the following:

```
&nV/ leftbit,length;leftbit,length;...;
```

where leftbit is a decimal integer specifying the left-most bit of a field, and length is a decimal integer specifying the length of the field (see Example 2).

Example 2:

```
;;I                             Select ILLIAC array address space.

;;$L                            Set word size to 64 bits (long mode).

$$8C                            Set typeout format to octal constants.
```

Example 2: (cont.)

```
100/ 0 -1ƀ                          Put -1 in the 64-bit word at location 100.

100/ 1777777777777777777777         Examine location 100 in octal format.

&2V/ 0 0,37;37,27;ƀ                 Set field registers with two fields: one
                                    37-bit field starting at bit position 0,
                                    and one 27-bit field starting at bit
                                    position 37.

$$2V                                Set typeout format to field, using
                                    register pair 2.

100/ 1777777777777,777777777        Examine location 100 in field format.

&2V/ 0,37;37,27; 0,5;42,22;ƀ        Set a different field format.

100/ 37, 17777777                   Examine location 100 in the new field format.
```

3.3 START, CONTINUE, AND EXIT

The following commands are used to start the user's program at a specified address, to restart the program (i.e., resume execution) from a breakpoint stop, and to exit from I4DDT. When breakpoints occur, program execution is suspended and each address space may have its registers manipulated, addresses and proceed counts changed, etc, at the user's option. To resume execution, the user selects the proper address space and gives a command to proceed.

Options

a. Start at a previously specified starting address. This is usually the address from the MACRO-10 or ASK END statement.

   Syntax: $G

b. Start at a specified address.

   Syntax: addr$G

c. Resume execution of the program at the instruction location where the break occurred.

Syntax: $P

d. Set the proceed count and proceed.

Syntax: n$P

where n is the proceed count.

e. Proceed from a breakpoint and then proceed automatically when the program breaks again.

Syntax: $$P

f. Increment breakpoint location by 1 and proceed ($P).

Syntax: $I

g. Increment breakpoint location by n and proceed ($P).

Syntax: n$I

h. Increment breakpoint location by 1 and proceed automatically ($$P).

Syntax: $$I

i. Increment breakpoint location by n and proceed automatically ($$P).

Syntax: n$$I

j. Execute a single instruction and return control to the user. I4DDT types out one to three dollar signs to indicate the number of locations skipped; register &X contains the address of the first word of a four-word block in the user's fork that is used to execute the instruction (normally 777774).

Syntax: instr$X

k. Execute the given instruction in a new fork for PDP-10. Other address spaces default to register &X.

Syntax: instr$$X

1. Exit from I4DDT.

   Syntax: $$Q

3.4 TYPE OUT THE CONTENTS OF A LOCATION

The command to type out the contents of a memory location does the following:

1. Causes the contents of the location in the current address space to be typed out on the terminal in the current typeout format, initially symbolic instruction (unless the format has been changed by a Change Typeout Format command).
2. Causes I4DDT's location pointer to point to the specified location.
3. Opens the location so that its contents can be changed by typing the new contents and a carriage return* following the typeout by I4DDT.

Options

a. Type out the contents of a location in current typeout format, open the location, and set the location pointer to this address.

   Syntax: addr/

   where addr is the symbolic or absolute address of the location to be typed out.

   Example: LOC/ MOVE A,CC1

b. Examine the current location.

   Syntax: ./

   Example: ./ MOVE A,CC1

*The line feed (↓) or vertical arrow (↑) may be used instead of the carriage return if the user wishes to increment or decrement the location pointer. These commands are described in Section 3.6.

c. Type out the contents of a location relative to the current value of the location pointer.

Syntax: .±n/

Example: .+5/ ADD B,SUM

d. Type out the contents of the location addressed in the last typeout.

Syntax: /

Example: LOC/ MOVE A,CC1 / ADD 2,SUM

where location LOC contains the instruction MOVE A,CC1 and CC1 contains the instruction ADD 2,SUM.

e. Change the temporary typeout format to constant, type out the contents of a location, open the location, and set the location pointer to this address.

Syntax: addr [

Example: LOC [9.

f. Change the temporary typeout format to symbolic instruction format, type out the contents of a location, open the location, and set the location pointer to this address.

Syntax: addr]

Example: LOC] MOVE 15,LIST+2

g. Open a location to modification, but suppress typeout of contents until a slash, left bracket, or right bracket is typed by the user.

Syntax: addr\

Example: LOC\MOVE AC,555↓

(The location LOC is open to modification, the user has typed in the new contents, but the reverse slash suppresses typeout.)

LOC+1\

(The line feed (↓) has incremented the location pointer, but the contents of LOC+1 are not typed out because the reverse slash is still in effect.)

LOC/ MOVE AC,555

(The slash terminates the effect of the reverse slash, and the new contents of the location are typed out.)

3.5 RETYPE THE CONTENTS OF A LOCATION IN A SPECIFIED FORMAT

Three commands are available that enable the user to have I4DDT retype the last quantity typed in a different specified format. These format changes are in effect only temporarily; when a carriage return is typed, I4DDT reverts to the prevailing format previously in effect.

Options

a. Retype the last expression in numeric format, with numbers in the current radix (initially octal).

Syntax: addr/ contents=

Example: =25411,,4050

b. Retype the last expression as a symbolic instruction. The address mode is determined by the typeout format in effect, i.e., absolute numeric address or relative to symbolic address (see Section 3.2).

Syntax: addr/ contents←

Example: ←ADD 4, TAB+1

c. Retype the last expression in the current format (normally used when the typeout format has been changed.)

Syntax: addr/ contents;/

Example: TEXT/ANDM 1,342212 (10) $T;/

ABCDE

3.6 CHANGE THE CONTENTS OF A LOCATION

In order to change the contents of a memory location, the user must first issue a command to cause the current contents of that location to be typed out. He does this by typing the symbolic or absolute address of the location followed by a slash. The location is now considered opened, and the user may change its contents by typing the desired new contents immediately following the typeout produced by I4DDT. A carriage return (↲) then commands I4DDT to make the indicated modification, and the location is now considered closed.

The contents of a location can also be changed by typing a LINE FEED (↓), vertical arrow (↑), or Control H (↑H) in place of a carriage return. A LINE FEED makes the desired modification, but then adds 1 to the location pointer if the word length (contained in register &W) is 36 or less, or adds 2 to the location pointer if the word length is greater than 36, and types out the resulting address and the contents of the new address. A vertical arrow or Control H makes the desired modification, but then subtracts 1 from the location point if the word length is 36 or less, or subtracts 2 from the location pointer if the word length is greater than 36, and types out the resulting address and the contents of the new address. If the carriage return, LINE FEED, vertical arrow, or Control H is preceded by an ALTMODE($), then the next-to-last location examined is used for setting the new value of the location pointer instead of the last location examined.

Options

a. Change the contents of a location, and close the location without moving location pointer.

Syntax: addr/current contents new contents↵

Example: LOC/MOVE A,CC1 MOVE A,CC2↵

b. Cause a carriage return, add 1 to the location pointer, type out the resulting address and the contents of the new address.

Syntax: addr/current contents new contents↓

Example: LOC/MOVE A, CC1 MOVE A,CC2↓

LOC + 1/ADD 3,CC3

c. Cause a carriage return, subtract 1 from the location pointer, type out the resulting address and the contents of the new address.

Syntax: addr/current contents new contents↑

Example: LOC + 1/ADD 3,CC3↑ ADD 4,CC4↑

LOC/MOVE A,CC2

3.7 INSERT A CHANGE AND EXAMINE CONTENTS OF LAST TYPED ADDRESS

The commands in this category cause I4DDT to close the current open location (making any modification typed in) and to open the following related locations, causing them to be typed out in the current typeout format.

Options

a. Type out the contents of the location specified by the address of the last quantity typed, and set the location pointer to this address.

Syntax: addr/ current contents new contents⇥

Example: LOC/MOVE A,CC1 MOVE A,CC2⇥

CC2/ 666

b. Open the contents of the location specified by the address of the last quantity typed, but do not change the location pointer.

Syntax: addr/ current contents new contents\

Example: LOC/ MOVE A,CC2 JRST X\

(The location pointer continues to point to LOC, but location X is now open and may be modified if desired.)

3.8 SET AND REMOVE BREAKPOINTS

The following commands enable the user to set and remove breakpoints and check the status of a breakpoint. Section 2.8 contains additional information on conditional breakpoints, breakpoint typeouts, and restrictions on breakpoint locations.

Options

a. Set a breakpoint at a specified location and assign a number to the breakpoint.

Syntax: addr$nB

where addr is the symbolic or absolute address of the location at which the user wishes to stop the program, and n is a breakpoint number from 1 through 8.

Example: CAT+3$4B (when CAT+3 is reached, I4DDT types out &4B >> CAT+3)

The user may also reassign a breakpoint, e.g., if he has set breakpoint 2 at location CAT (CAT$2B), he may reassign breakpoint 2 to location CAT+1 by typing CAT+1$2B.

b. Set a breakpoint at a specified location and assign it the next unused breakpoint number.

Syntax: addr$B

Example: 4002$B

c. Set a breakpoint at a specified location and set automatic proceed.*

Syntax: addr$$nB

If n is omitted, set the next unused breakpoint with automatic proceed.

d. Set a breakpoint and open and examine a specified location.

Syntax: x<addr$nB

where addr is the address of the breakpoint and x is the address of the location to be opened and examined when the breakpoint is reached. If x=-1, examine location 0. If n is omitted, set the next unused breakpoint and open and examine location x.

e. Set a breakpoint with automatic proceed and open and examine a specified location.

Syntax: x<addr$$nB

where x, addr, and n have the values described above.

f. Remove a specific breakpoint.

Syntax: 0$nB

where n is the number of the breakpoint to be removed.

*To get out of automatic proceed mode, type any teletype key during the typeout, then remove the breakpoint or reassign it with a single ALTMODE. It may be necessary to type ↑C and REENTER to return to I4DDT to remove or reassign the breakpoint.

g. Remove all breakpoints.

Syntax: $B

h. Type out all the set breakpoints and their locations.

Syntax: $$B

i. Check the status of a specific breakpoint (n).

Syntax: &nB/

Location &nB contains the address of the breakpoint. (If &nB equals zero, the breakpoint is not in use.)

j. Insert a conditional instruction or subroutine call at breakpoint n.

Syntax: &nB+1/

Example: &3B+1/ 0 CAIGE ACC,15ƀ

If the conditional instruction does not cause a skip, the proceed counter is decremented and checked; if 0, a break occurs. If the conditional instruction causes one skip, a break occurs. If the conditional instruction causes two skips, execution of the program proceeds.

k. Set the proceed count at breakpoint n.

Syntax: &nB+2/

Example: &3B+2/ 0 20ƀ

l. Execute a string of I4DDT commands when a breakpoint is reached.

Syntax: $nBXccc...cccX

where n is the breakpoint number, X is an arbitrary delimiting character, and ccc...ccc is a text string to be fed to I4DDT for execution as I4DDT commands when breakpoint n is reached.

3.9 SYMBOL MANIPULATION

The following commands enable the user to reference a local symbol within a particular program, assign a value to a symbol, and restrict or eliminate the use of a defined symbol.

Options

a. Reference local symbols within a particular program, i.e., unlock the symbol table associated with that program.

Syntax: name$:

where name is the program name, e.g., the name specified in the MACRO-10 or ASK TITLE statement.

Example: PAYROL$:

where PAYROL is the name of the program whose symbol table the user wishes to access.

b. Insert a symbol in the symbol table and assign a numeric value to the symbol.

Syntax: value<symbol:

where value is any numeric value and symbol is any defined or undefined symbol.

Example: 305<XVAR:

XVAR is now defined to have the value 305.

c. Insert or redefine a symbol in the symbol table and assign it a value equal to the address of the location pointer.

Syntax: symbol:

Example: TAG:

TAG is now defined to have the value equal to the address of the location pointer.

d. Remove a symbol from the symbol table, i.e., prevent the symbol from being used for input or output.

Syntax: symbol$$K

Example: TAG$$K

TAG is now removed from the symbol table.

e. Prevent I4DDT from using a symbol for typeout.

Syntax: symbol$K

Example: TAG$K

The symbol TAG will be replaced in subsequent typeouts by a symbol that has the same numeric value.

f. Prevent I4DDT from using the last symbol typed out as a typeout symbol, and retype the symbol as a quantity in the current typeout mode.

Syntax: $D

Example: A/ MOVE AC,LOC $D MOVE AC,ABC+1

g. Declare an undefined symbol for later definition.

Syntax: symbol#

Example: MOVE 2,SAVE#

SAVE is now saved by I4DDT, and when finally defined, all previous occurrences of the symbol will be replaced with the defined value.

h. Type out a list of all undefined symbols (created by symbol#).

Syntax: ?

Example: ?

SAVE

UNDEF

i. Type out a list of all symbols and their values.

Syntax: $?

j. Type out a symbol and its value.

Syntax: symbol$?

k. Type out a list of all symbols beginning with a specified character string.

Syntax: string$$?

where a string consists of one to five alphanumeric characters.

l. Type out a list of all symbols whose values are equal to n.

Syntax: $n?

3.10 SEARCH FOR A SPECIFIED VALUE

I4DDT's search commands enable the user to search a specified portion of memory for (a) a particular quantity (word search), (b) the absence of a particular quantity (not-word search), or (c) an effective address. The lower and upper limits of the search may be specified by the user, or, if one or both of these parameters are omitted, the values currently in internal registers &LL and &UL are used.

The word search compares the contents of each memory location with a specified value. If the comparison shows an equality, the address and contents of the location are typed out; if the comparison shows an inequality, nothing is typed and the search continues until the upper limit is reached. The value searched for may be modified by a mask word (located at &M) which specifies which bit positions are to be compared. (In double-word searches, register &M1 contains the high-order mask word.) A one bit in the mask word causes the corresponding bit position in each memory location to be compared;

a zero bit in the mask word causes the corresponding bit position in each memory location to be ignored. The mask word initially contains all ones, so that all bit positions are compared; however, the contents of the mask word may be set by the user.

The not-word search types out the address and contents of each location where the comparison shows an inequality; if the comparison shows an equality, nothing is typed and the search continues until the upper limit is reached.

The effective address search types out the address and contents of all locations that contain a specified effective address (the actual address modified by indexing or indirect addressing); if the comparison shows an inequality, nothing is typed and the search continues until the upper limit is reached.

Options

a. Insert lower limit in register &LL and upper limit in register &UL, and type out the address and contents of all locations that contain a specified value.

Syntax: a <b> c$W

where a is the address of the lower limit of the search, b is the address of the upper limit of the search, and c is the value being searched for.

```
Example:  INPT <INPT+10> X$W
          INPT+2/ X
```

b. Using the lower limit contained in register &LL and the upper limit contained in register &UL, type out the address and contents of all locations that contain a specified value.

Syntax: c$W

where c is the value being searched for.

c. Insert lower limit in register &LL and upper limit in register &UL, and type out the address and contents of all locations that do not contain a specified value.

Syntax: a <b> c$N

where a, b, and c are the addresses and value described above.

```
Example:  INPT <INPT+10> DATA$N
          INPT/ 4503,,4502
          INPT+5/ 202400,,6736
          INPT+7/ 50500,773400
```

d. Using the lower limit contained in register &LL and the upper limit contained in register &UL, type out the address and contents of all locations that do not contain a specified value.

Syntax: c$N

where c is the value being searched for.

e. Insert lower limit in register &LL and upper limit in register &UL, and type out the address and contents of all locations that contain a specified effective address, following all indirect and index-register chains to a maximum depth of 64_{10} levels.

Syntax: a<b>c$E

where a,b, and c are the addresses described above.

```
Example:  4517<5000>X$E
          4517/ SETZM X
          4727/ MOVE 2,X
          5000/ MOVE 3, @4721 (address 4721 indirectly
                                         addresses X)
```

f. Using the lower limit contained in register &LL and the upper limit contained in register &UL, type out the address and contents of all locations that contain an effective address.

   Syntax: c$E

   where c is the effective address searched for.

g. Type out the contents of the mask register, which is then open for modification.

   Syntax: &M/
   &M1/

   Example: &M/ 777777777777
   &M1/777777777777

h. Insert a quantity into the mask register.

   Syntax: n$M

   Example: 46$M

3.11 PATCHING

I4DDT's patch commands enable the user to insert a series of instructions at the currently opened location in his program. The procedures are as follows:

1. Open register &PL and type in the address of the location where the patch instructions will be inserted (if address 775000 is not available).

   &PL/ 775000 PCHSPC↲

2. Open the location where the patch is to be inserted and type in

   $Y

   to insert a JRST (Jump) instruction in the location currently open.

3. Enter the proper instructions, open the location following the last instruction in the patch, and type in

```
$$Y
```

This concludes the patch.

APPENDIX A

I4DDT COMMAND SUMMARY

A.1 EXECUTIVE FUNCTIONS

A.1.1 ADDRESS SPACE SELECTION

Syntax	Description
;P	Temporary PDP-10
;;P	Permanent PDP-10
;I	Temporary ILLIAC IV array
;;I	Permanent ILLIAC IV array
;D	Temporary ILLIAC IV disk
;;D	Permanent ILLIAC IV disk
;E	Temporary PDP-11
;;E	Permanent PDP-11
;T	Temporary TMU
;;T	Permanent TMU

A.1.2 INPUT/OUTPUT MODE SELECTION

Syntax	Description
;$P	Temporary PDP-10 mode
;;$P	Permanent PDP-10 mode
;$S	Temporary short ILLIAC IV mode (32 bits)
;;$S	Permanent short ILLIAC IV mode (32 bits)
;$L	Temporary long ILLIAC IV mode (64 bits)
;;$L	Permanent long ILLIAC IV mode (64 bits)
;$E	Temporary PDP-11 mode
;;$E	Permanent PDP-11 mode
;$T	Temporary TMU mode
;;$T	Permanent TMU mode
;$D	Temporary short ILLIAC disk mode (equivalent to ;$S)
;;$D	Permanent short ILLIAC disk mode (equivalent to ;;$S)

Syntax	Description
;$I	Temporary short ILLIAC array mode (equivalent to ;$S)
;;$I	Permanent short ILLIAC array mode (equivalent to ;;$S)

A.1.3 FILE OPERATIONS

Syntax	Description
;G	Load a file and the symbol table into the current address space.
;;G	Load a new symbol table without changing the contents of the address space.
;S	Move the contents of the current address space to a file.
;;S	Move a symbol table to a file.
;M	Merge a file into the current address space.
;;M	Merge a symbol table into the current address space.
;L	Bring the PDP-10 loader into the current address space and return control to I4DDT.
;;L	Bring the PDP-10 leader into the current address space and start the user's program at a previously specified starting address.
;X	Execute command strings from a file.

A.1.4 MISCELLANEOUS EXEC FUNCTIONS

Syntax	Description
a<b;Z	Set lower limit (a), upper limit (b), and zero memory using these limits. Do not zero symbol table.
a<b;;Z	Set lower limit (a), upper limit (b), and zero memory and the symbol table using these limits.

Syntax	Description
a<b;nU	Change protection on pages a through b according to the 3-bit value of n: 4-bit = allow read access, 2-bit = allow write access, 1-bit = allow execute access. Default = allow all.
a;nU	Change protection on page a according to the above values of n.
;nU	Change protection on current page according to above values of n.
;O	Open access to ILLIAC and copy virtual ILLIAC image to ILLIAC memory.
;;O	Open access to ILLIAC.
;C	Copy ILLIAC memory to virtual ILLIAC image and close access to ILLIAC.
;;C	Close access to ILLIAC.
;A	Replace ↑T with new escape character.
;;A	Replace ↑E with new escape character.

A.2 CHANGE TYPEOUT FORMAT - TEMPORARY ($) OR PREVAILING ($$)

Syntax	Description
$R or $$R	Address relative to symbolic locations.
$nR or $$nR	Non-data numeric output radix.
$A or $$A	Addresses as absolute numbers.
$nS or $$nS	Symbolic instructions, where n=0 (PDP-10), 1 (short ILLIAC), 2 (long ILLIAC), 3 (TMU), or 4 (PDP-11).
$S or $$S	Symbolic instructions, with the class determined by register &S.
$nC or $$nC	Location contents as numbers in the current radix, where n=2, 8, 10, or 16.
$C or $$C	Location contents as numbers in the current radix, with the radix determined by register &C.
$nF or $$nF	n-bit floating-point numbers, where n=32, 36, or 64.

Syntax	Description
$F or $$F	32-, 36-, or 64-bit floating-point numbers, depending on the contents of register &F.
$nT or $$nT	Text characters, where n=8 (8-bit ASCII), 7 (7-bit ASCII), 6 (SIXBIT), or 5 (Radix-50).
$T or $$T	Text characters, with the type determined by register &T.
$nO or $$nO	n-bit bytes, where n is a value from 1 to 36.
$O or $$O	n-bit bytes, with n determined by register &O.
$H or $$H	n-bit bytes, where n is half the value of register &W.
$V or $$V	Interpret location according to boundary field register &nVB and mask field register &nVM, where n (the current register pair) is contained in register &V.
$nV or $$nV	Change typeout format to field, and insert n (field register pair 1 through 8) into register &V.

A.3 START, CONTINUE, AND EXIT

Syntax	Description
$G	Start at a previously specified address.
addr$G	Start at a specified address.
$P	Restart the program at a breakpoint location.
n$P	Set the proceed count and proceed.
$$P	Proceed from a breakpoint and then proceed automatically when the program breaks again.
$I	Increment breakpoint location by 1 and proceed.
n$I	Increment breakpoint location by n and proceed.
$$I	Increment breakpoint location by 1 and proceed automatically.

Syntax	Description
n$$I	Increment breakpoint location by n and proceed automatically.
instr$X	Execute a single instruction and return control to the user.
instr$$X	Execute the given instruction in a new fork for PDP-10.
$$Q	Exit from I4DDT.

A.4 TYPE OUT THE CONTENTS OF A LOCATION

Syntax	Description
addr/	Type the contents of location addr in the current typeout format.
./	Examine the current location.
.±n/	Type the contents of a location relative to the current value of the location pointer.
/	Type out the contents of the location addressed in the last typeout.
addr[	Change the temporary typeout format to constant and type out the contents of location addr.
addr]	Change the temporary typeout format to symbolic instruction and type out the contents of location addr.
addr\	Open a location to modification, but suppress typeout of contents until a slash, left bracket, or right bracket is typed by the user.

A.5 RETYPE THE CONTENTS OF A LOCATION

Syntax	Description
addr/contents=	Retype the last expression in numeric format, with numbers in the current radix.
addr/contents←	Retype the last expression as a symbolic instruction.

Syntax	Description
addr/contents;/	Retype the last expression in the current format.

A.6 CHANGE THE CONTENTS OF A LOCATION

Syntax	Description
addr/ contents change↵	Substitute new contents without moving location pointer.
addr/ contents change$↵	Substitute new contents and move location pointer to the most previous location examined.
addr/ contents change↓	Move location pointer to addr+1, type the address and contents, and open the address.
addr/ contents change$↓	Move location pointer to the most previous location examined + 1, and type the address and contents.
addr/ contents change↑	Move location pointer to addr-1, type the address and contents, and close the address.
addr/ contents change$↑	Move location pointer to the most previous location examined -1, and type the address and contents.

A.7 INSERT A CHANGE AND EXAMINE LAST LOCATION

Syntax	Description
addr/ contents change→|	Type out the contents of the location specified by the address of the last quantity typed, and set the location pointer to this address.
addr/ contents change\	Open the contents of the location specified by the address of the last quantity typed, but do not change the location pointer.

A.8 SET AND REMOVE BREAKPOINTS

Syntax	Description
addr$B	Set the next unused breakpoint at location addr.
addr$nB	Set and assign a breakpoint number to location addr (n=1-8).
addr$$B	Set a breakpoint with automatic proceed.
addr$$nB	
addr1<addr2$B	Set breakpoint at addr2 and type contents of addr1 when the breakpoint is reached.
addr1<addr2$nB	
addr1<addr2$$B	
addr1<addr2$$nB	
0$nB	Remove a specific breakpoint.
$B	Remove all breakpoints.
$$B	Type out current settings of all breakpoints.
&nB/	Check the status of a specific breakpoint.
&nB+1/	Insert a conditional instruction or subroutine call at breakpoint n.
&nB+2/	Set proceed count at breakpoint n.
$nBXccc...X	Submit a command string delimited by arbitrary character X to I4DDT for execution when breakpoint n is reached.

A.9 SYMBOL MANIPULATION

Syntax	Description
name$:	Reference local symbols within a named program.
value<symbol:	Insert a symbol in the symbol table and assign it a numeric value.
symbol:	Insert or redefine a symbol in the symbol table and assign it the value of the location pointer.
symbol$$K	Remove a symbol from the symbol table.
symbol$K	Prevent a symbol from being used for typeouts.

Syntax — Description

$D
Prevent the last symbol typed out from being used as a typeout symbol, and retype the symbol as a quantity in the current typeout mode.

symbol#
Declare a symbol for later definition.

?
Type out a list of all undefined symbols.

$?
Type out a list of all symbols and their values.

symbol$?
Type out a symbol and its value.

string$$?
Type out a list of all symbols beginning with a specified character string.

$n?
Type out a list of all symbols whose values are equal to n.

A.10 SEARCH FOR A SPECIFIED VALUE

Syntax — Description

a<b>c$W
Set lower limit (a), upper limit (b), a location to be searched for (c), and type out the address and contents of all locations equal to c.

c$W
Using the lower limit in register &LL and the upper limit in register &UL, type out the address and contents of all locations equal to c.

a<b>c$N
Set lower limit (a), upper limit (b), a value to be searched for (c), and type out the address and contents of all locations not equal to c.

c$N
Using the lower limit in register &LL and the upper limit in register &UL, type out the address and contents of all locations not equal to c.

Syntax — Description

a<b>c$E — Set lower limit (a), upper limit (b), an address to be searched for (c), and type out the address and contents of all locations that contain the effective address c.

c$E — Using the lower limit in register &LL and the upper limit in register &UL, type out the address and contents of all locations that contain the effective address c.

n$M — Insert the quantity n into mask register &M, and &M1.

A.11 PATCHING

Syntax — Description

$Y — Begin a patch by inserting a JRST (Jump and Restore) instruction in the currently opened location, and open register &PL and insert the proper instructions.

$$Y — End a patch by inserting the patched instruction in the current location.

APPENDIX B

DIAGNOSTIC MESSAGES

Message	Meaning
U	The user has typed an undefined symbol which cannot be interpreted by I4DDT.
?	The user has committed one of the following errors: (1) typed an illegal I4DDT command. (2) referenced a location in non-existent or read-protected memory. (3) attempted to write into a location that is inside a write-protected memory segment.
XXX: addr/ inst	The user has pressed the ↑E key while typing in a command. The next instruction to be executed (inst) is at location addr.

APPENDIX C

GLOSSARY

Address space

The memory space(s) associated with a process run under I4DDT. The specification of address space designates to I4DDT the machine in which a given program executes. Under I4DDT, the user may specify one of four different address spaces: PDP-10, PDP-11, ILLIAC IV array, or ILLIAC IV disk. Address space specification may be changed during the running of a process under I4DDT.

Array memory

Working storage for the ILLIAC IV processor, consisting of 128K 64-bit words or 256K 32-bit words. Each of the 64 processing elements in the ILLIAC IV processor has access to one 2K (64-bit) block of array memory.

ASK

The ILLIAC IV assembler that accepts ILLIAC IV symbolic instructions as input, and generates relocatable object code for execution on the ILLIAC IV.

Breakpoint

A point in the program at which execution is suspended and control is given to I4DDT, so that the user may intervene in the execution of the program making modification, insertions, and deletions where necessary. See Sections 2.8 and 3.8.

Byte

A contiguous set of bits within a memory location operated upon as a unit. I4DDT's byte commands enable the user to specify a byte format containing from 1 to 36 bits. See Section 3.2.

Delimiter

A non-blank printing character that must immediately precede and follow two or more ASCII or SIXBIT text characters typed in by the user, e.g., /TEXT/, where the slashes are the delimiters. See Section 2.4.3.

Effective address

An address as modified by indexing or indirect addressing.

File

A named, ordered collection of data associated with a particular user id, and that is uniquely identified within a user's set of files by its file name.

Fork

See process.

Global symbol

A symbol that can be referenced by a program other than the one in which it is defined. See Section 2.6.1.

Job

A set of one or more related processes, each of which has its own address space, execute independently, and can communicate with each other.

Loader

A program that loads and links relocatable binary programs preparatory to execution, and generates a symbol table in core for execution under I4DDT.

Local symbol

A symbol which can be referenced only within the program in which it is defined (i.e., a non-global symbol). A local symbol is not accessible to other programs even if the programs are loaded together. See Section 2.6.1.

Location pointer

A memory location containing the actual (effective) address of the data or instruction currently being referenced, or, the register containing the pointer address.

MACRO-10

The name of an assembler for the PDP-10 computer.

Mask

A string of characters containing all 1-bits in positions where data of another string are to be preserved, and containing 0-bits in all positions where data of the second string are to be ignored. The mask is usually combined with its target data in a logical AND operation.

Open location

An open location is a location in a user's memory space whose contents are available for change. A memory location is opened by requesting I4DDT to type out the contents for examination.

Process

An entity that receives scheduling and resource allocation attention from the system and that has a separate flow of control and address space.

Radix-50

A condensed 32-bit representation of a six character symbol (see PDP-10 MACRO-10 Manual, Appendix F).

Symbol

A name consisting of a string of up to six letters and numbers including the special characters period (.), percent sign (%), and dollar sign ($). See Section 2.6.

Unimplemented User Operation (UUO)

A mnemonic code that is not a specific instruction, but must be interpreted by a routine supplied by the programmer.

APPENDIX D

I4DDT COMMAND IMPLEMENTATION STATUS

ADDRESS SPACE SELECTION:	PDP-10	ARRAY	DISK	TMU	PDP-11
;P	YES				
;;P	YES				
;I		YES			
;;I		YES			
;D			YES		
;;D			YES		
;T				YES	
;;T				YES	
;E					
;;E					

INPUT/OUTPUT MODE SELECTION:	PDP-10	ARRAY	DISK	TMU	PDP-11
;$P	YES	YES	YES	YES	
;;$P	YES	YES	YES	YES	
;$S	YES	YES	YES	YES	
;;$S	YES	YES	YES	YES	
;$L	YES	YES	YES	YES	
;;$L	YES	YES	YES	YES	
;$E					
;;$E					
;$T	YES	YES	YES	YES	
;;$T	YES	YES	YES	YES	
;$D	YES	YES	YES	YES	
;;$D	YES	YES	YES	YES	
;$I	YES	YES	YES	YES	
;;$I	YES	YES	YES	YES	

FILE OPERATIONS:	PDP-10	ARRAY	DISK	TMU	PDP-11
;G	YES	YES	YES	YES	
;;G	YES				
;S	YES	YES	YES	YES	
;;S	YES				
;M	YES				
;;M	YES				
;L	YES				
;;L	YES				
;X	YES	YES	YES	YES	

MISCELLANEOUS FUNCTIONS:	PDP-10	ARRAY	DISK	TMU	PDP-11
A<B;Z					
A<B;;Z	YES	YES	YES	YES	
A<B;NU	YES	YES	YES	YES	
A;NU	YES	YES	YES	YES	
;NU	YES	YES	YES	YES	
;O		YES			
;;O		YES			
;C		YES			
;;C		YES			
;A	YES	YES	YES	YES	
;;A	YES	YES	YES	YES	

CHANGE TYPEOUT FORMAT:		PDP-10	ARRAY	DISK	TMU	PDP-11
$R	$$R	YES	YES	YES	YES	
$NR	$$NR	YES	YES	YES	YES	
$A	$$A	YES	YES	YES	YES	
$NS	$$NS	YES	YES	YES	YES	
$S	$$S	YES	YES	YES	YES	
$NC	$$NC	YES	YES	YES	YES	
$C	$$C	YES	YES	YES	YES	
$NF	$$NF	YES	YES	YES	YES	
$F	$$F	YES	YES	YES	YES	
$NT	$$NT	YES	YES	YES	YES	
$T	$$T	YES	YES	YES	YES	
$NO	$$NO	YES	YES	YES	YES	
$O	$$O	YES	YES	YES	YES	
$H	$$H	YES	YES	YES	YES	
$V	$$V	YES	YES	YES	YES	
$NV	$$NV	YES	YES	YES	YES	

START, CONTINUE, AND EXIT:	PDP-10	ARRAY	DISK	TMU	PDP-11
$G	YES	YES		YES	
ADDR$G	YES	YES			
$P	YES	YES			
N$P	YES	YES			
$$P	YES	YES			
$I	YES	YES			
N$I	YES	YES			
$$I	YES	YES			
N$$I	YES	YES			
INSTR$X	YES	YES		YES	
INSTR$$X	YES				
$$Q	YES	YES	YES	YES	

SYMBOL MANIPULATION:	PDP-10	ARRAY	DISK	TMU	PDP-11
NAMES:	YES	YES	YES	YES	
VALUE<SYMBOL:	YES	YES	YES	YES	
SYMBOL:	YES	YES	YES	YES	
SYMBOL$$K	YES	YES	YES	YES	
SYMBOL$K	YES	YES	YES	YES	
$D	YES				
SYMBOL#	YES				
?	YES				
$?	YES				
SYMBOL$?	YES				
STRING$$?	YES				
$N?					

SEARCH FOR A SPECIFIED VALUE:	PDP-10	ARRAY	DISK	TMU	PDP-11
A<B>C$W	YES	YES	YES	YES	
C$W	YES	YES	YES	YES	
A<B>C$N	YES	YES	YES	YES	
C$N	YES	YES	YES	YES	
A<B>C$E	YES				
C$E	YES				
N$M	YES	YES	YES	YES	

MISCELLANEOUS COMMANDS:	PDP-10	ARRAY	DISK	TMU	PDP-11
$Y	YES				
$$Y	YES				
$$Q	YES	YES	YES	YES	

TYPE OUT CONTENTS OF LOCATION:	PDP-10	ARRAY	DISK	TMU	PDP-11
ADDR/	YES	YES	YES	YES	
./	YES	YES	YES	YES	
.+N/	YES	YES	YES	YES	
.-N/	YES	YES	YES	YES	
/	YES				
ADDR[	YES	YES	YES	YES	
ADDR]	YES	YES	YES	YES	
ADDR \	YES	YES	YES	YES	

RETYPE CONTENTS OF A LOCATION:	PDP-10	ARRAY	DISK	TMU	PDP-11
=	YES	YES	YES	YES	
←	YES	YES	YES	YES	
;/	YES	YES	YES	YES	

CHANGE CONTENTS OF A LOCATION:	PDP-10	ARRAY	DISK	TMU	PDP-11
CHANGE (CR)	YES	YES	YES	YES	
CHANGE $ (CR)	YES	YES	YES	YES	
CHANGE (LF)	YES	YES	YES	YES	
CHANGE $ (LF)	YES	YES	YES	YES	
CHANGE ↑	YES	YES	YES	YES	
CHANGE $ ↑	YES	YES	YES	YES	
CHANGE (TAB)	YES				
CHANGE \	YES				

SET AND REMOVE BREAKPOINTS:	PDP-10	ARRAY	DISK	TMU	PDP-11
ADDR$B	YES	YES			
ADDR$NB	YES	YES			
ADDR$$B	YES	YES			
ADDR$$NB	YES	YES			
ADDR1<ADDR2$B	YES	YES			
ADDR1<ADDR2$NB	YES	YES			
ADDR1<ADDR2$$B	YES	YES			
ADDR1<ADDR2$$NB	YES	YES			
Ø$NB	YES	YES			
$B	YES	YES			
$$B	YES	YES			
&NB+M/	YES	YES			
$NBXSSSSSSSSSSSX					

APPENDIX E

USE OF I4DDT IN TMU ADDRESS SPACE

E.1 TMU OPERATION

E.1.1 PROGRAM EXECUTION

The TMU address space consists of up to 512 TENEX pages containing 64-bit TMU instructions, right justified within pairs of PDP-10 words (see Section E.3).

To operate I4DDT in TMU mode, the user first selects the TMU address space by typing ;T or ;;T. After loading the TMU program, the user starts the program by typing $G or addr$G. The address of the starting instruction is stored in register &PC; &PC is incremented by 2 after each instruction is fetched, but before the instruction is executed.

Program execution continues until one of the following is encountered:

- The pseudo TMU instruction HALT
- An error condition
- A user-generated interrupt

E.1.2 PROGRAM INTERRUPTION

The user can interrupt a TMU program during execution by typing any character on his terminal. I4DDT then informs the user where the program was stopped. Execution can be resumed by typing $G.

The user can determine where his program is running by examining register &PC. He can also examine any of the four general registers by using the pseudo TMU instruction TYPE (see Section E.3). The Jump if Interrupt (JINT) pseudo instruction is used to field interrupts. If more than three unprocessed interrupts are queued, the queued interrupts are ignored.

E.2 INTERNAL REGISTERS

The following internal registers are used in TMU operation:

&PC	An 18-bit program counter.
&TCI	Storage for the TMU Condition Indicator Register (TCI).
&TRO	Storage for the TMU Output Register (TRO).
0	A 64-bit general register.
1	A 64-bit general register.
2	A 64-bit general register.
3	A 64-bit general register.

E.3 PSEUDO TMU INSTRUCTIONS

In addition to the actual TMU instructions described elsewhere,* I4DDT also accepts the following 48-bit pseudo TMU instructions. The upper 16 bits of these instructions are stored in the upper 16 bits of the 64-bit TMU address space word, and the lower 32 bits are stored in the lower 32 bits of the address space word.

In the following instruction descriptions, (ireg1) and (ireg2) are integers which reference one of the four TMU general registers.

E.3.1 ACQUIRE ILLIAC

Syntax: ACQ

Description: Acquire ILLIAC if it is not already acquired.

E.3.2 ADD

Syntax: ADD ireg1, ireg2

Description: (ireg1) + (ireg2) → (ireg1)

E.3.3 ADD IMMEDIATE

Syntax: ADDI ireg1, data

Description: (ireg1) + data → (ireg1)

*ILLIAC IV Systems Characteristics and Programming Manual, Section 5.

E.3.4 AND

Syntax: AND iregl, ireg2

Description: (iregl) .AND. (ireg2) → (iregl)

E.3.5 AND LEFT IMMEDIATE

Syntax: ANDLI iregl, data

Description: LH(iregl) .AND. data → LH(iregl)

E.3.6 AND RIGHT IMMEDIATE

Syntax: ANDRI iregl, data

Description: RH(iregl) .AND. data → RH(iregl)

E.3.7 TYPE OUT ASCII TEXT

Syntax: ASCII <delimiter> text <delimiter>

Description: Type out on the user's terminal the ASCII text enclosed by <delimiter>. The text is stored five characters per TMU word as successive ASCII commands.

E.3.8 CALL SUBROUTINE

Syntax: CALL address

Description: (&PC) → RH(address), address + 2 → (&PC)

E.3.9 HALT TMU PROCESSING

Syntax: HALT

Description: Stop processing TMU commands and return to I4DDT's command scanner.

E.3.10 JUMP UNOONDITIONAL

Syntax: J address

Description: address → (&PC)

E.3.11 JUMP IF BIT SET

Syntax: JB iregl, bit no., address

Description: If iregl has bit no. set, then address → (&PC).

E.3.12 JUMP IF EQUAL TO ZERO

Syntax: JE iregl, address

Description: If iregl = 0, then address → (&PC).

E.3.13 JUMP IF GREATER THAN ZERO

Syntax: JG iregl, address

Description: If iregl>0, then address → (&PC).

E.3.14 JUMP GREATER/DECREMENT

Syntax: JGD iregl, address

Description: (iregl) - 1 → (iregl); if (iregl) ≥ 0, then address → (&PC).

E.3.15 JUMP IF INTERRUPT

Syntax: JINT address

Description: If an interrupt occurred, (TCI) → (&TCI), (TRO) → (&TRO), and address → (&PC).

E.3.16 JUMP IF LESS THAN ZERO

Syntax: JL address

Description: If (iregl<0, then address → (&PC).

E.3.17 JUMP IF BIT NOT SET

Syntax: JNB iregl, bit no., address

Description: If iregl has bit no. reset, then address → (&PC).

E.3.18 JUMP IF NOT EQUAL TO ZERO

Syntax: JNE iregl, address

Description: If iregl ≠ 0, then address → (&PC).

E.3.19 JUMP INDEXED

Syntax: JX iregl, address

Description: (iregl) + address → (&PC)

E.3.20 LOAD

Syntax: L iregl, address

Description: address → (iregl)

E.3.21 LOAD RIGHT HALF

Syntax: LRH iregl, address

Description: RH(address) → RH(iregl)

E.3.22 LOGICAL SHIFT LEFT

Syntax: LSHL iregl, count

Description: Logically shift iregl left by the number of positions specified by count.

E.3.23 LOGICAL SHIFT RIGHT

Syntax: LSHR iregl, count

Description: Logically shift iregl right by the number of positions specified by count.

E.3.24 MOVE

Syntax: MOVE iregl, ireg2

Description: (ireg2) → (iregl)

E.3.25 MOVE IMMEDIATE

Syntax: MOVI iregl, data

Description: data → (iregl)

E.3.26 MOVE LEFT IMMEDIATE

Syntax: MOVLI iregl, data

Description: data → LH(iregl)

E.3.27 MOVE RIGHT IMMEDIATE

Syntax: MOVRI iregl, data

Description: data → RH(iregl)

E.3.28 OR

Syntax: OR iregl, ireg2

Description: (iregl) .OR. (ireg2) → (iregl)

E.3.29 OR LEFT IMMEDIATE

Syntax: ORLI iregl, data

Description: LH(iregl) .OR. data → LH(iregl)

E.3.30 OR RIGHT IMMEDIATE

Syntax: ORRI iregl, data

Description: RH(iregl) .OR. data → RH(iregl)

E.3.31 READ TRO

Syntax: RDTRO

Description: Transfer TCI and TRO into &TCI and &TRO, respectively. An implicit RDTRO will be performed after each SOD (Scan Out Data) TMU instruction.

E.3.32 RELEASE ILLIAC

Syntax: REL

Description: Release ILLIAC IV if it was acquired.

E.3.33 ROTATE LEFT

Syntax: ROTL iregl, count

Description: Rotate iregl left by the number of positions specified by count.

E.3.34 ROTATE RIGHT

Syntax: ROTR iregl, count

Description: Rotate iregl right by the number of positions specified by count.

E.3.35 STORE

Syntax: S iregl, address

Description: (iregl) → (address)

E.3.36 STOP CU

Syntax: SCU

Description: Stop the CU and clear the TMU.

E.3.37 STORE RIGHT

Syntax: SRH iregl, address

Description: RH(iregl) → RH(address)

E.3.38 STORE TCI

Syntax: STCI iregl

Description: Store &TCI into the upper 16 bits of iregl and clear all other bits in iregl.

E.3.39 STORE TRO

Syntax: STRO iregl

Description: Store &TRO into iregl.

E.3.40 SUBTRACT

Syntax: SUB iregl, ireg2

Description: (iregl) - (ireg2) → (iregl)

E.3.41 SUBTRACT IMMEDIATE

Syntax: SUBI iregl, data

Description: (iregl) - data → (iregl)

E.3.42 TYPE OUT

Syntax: TYPE iregl, radix

Description: Type out the contents of iregl on the user's terminal using the specified radix. If no radix is specified, use the current I4DDT data radix (&C).

E.3.43 WAIT

Syntax: WAIT count

Description: Wait for the number of milliseconds specified by count.

E.3.44 EXCLUSIVE OR

Syntax: XOR iregl, ireg2

Description: (iregl) .XOR. (ireg2) → (iregl)

E.3.45 EXCLUSIVE OR LEFT IMMEDIATE

Syntax: XORLI iregl, data

Description: LH(iregl) .XOR. data → LH(iregl)

E.3.46 EXCLUSIVE OR RIGHT IMMEDIATE

Syntax: XORRI iregl, data

Description: RH(iregl) .XOR. data → RH(iregl)

E.4 PROGRAMMING EXAMPLE

The following TMU program types out the current contents of the ADVAST data buffer (ADB) on the user's terminal.

```
0/            ACQ         ;acquire ILLIAC
2/            SCU         ;clear the TMU
4/            MOVI 0,77   ;initialize counter
6/            MOVI 1,0    ;initialize ADB index
10/           CALL 100    ;read ADB register into 2
12/           ASCII "D"   ;type "D"
14/           TYPE 1.8.   ;type ADB index(octal)
16/           ASCII "  "  ;type a couple spaces
20/           TYPE 2      ;type contents of 2
22/           ASCII "
"                         ;type a carriage return
24/           ADDI 1,1    ;bump ADB index
26/           JGD 0,10    ;decrement count, jump
                          ; if more to do
30/           REL         ;release ILLIAC
32/           HALT        ;stop program
100/          J 0         ;return to caller
102/          L 3,200     ;get SOD instruction
104/          ROTL 3,16.  ;position opcode
106/          ADD 3,1     ;add ADB index
110/          ROTR 3,16.  ;reposition opcode
112/          S 3,114     ;store generated SOD
                          ; instruction
114/          SOD DOO     ;execute an SOD
116/          STRO 2      ;store result in 2
120/          J 100       ;return
200/          SOD DOO
```

APPENDIX F

USE OF I4DDT IN ILLIAC ARRAY ADDRESS SPACE

F.1 I4DDT ILLIAC INSTRUCTION SYNTAX

The mnemonic operation codes for I4DDT's ILLIAC instructions are the same as the ASK operation codes. However, the following differences exist with regard to the syntax used in writing the instruction operands:

	ASK	I4DDT
Accumulator Registers (ACAR's)		
ACAR0	(0)	(AC0)
ACAR1	(1)	(AC1)
ACAR2	(2)	(AC2)
ACAR3	(3)	(AC3)
PE Index Register X(RGX):	*location	@@location
PE Register S (RGS):	#location	@#location
	$S	@S
PE Register A´(RGA):	$A	@A
PE Register B (RGB):	$B	@B
PE Register R (RGR):	$R	@R
Literals:	=literal	@=literal
SKIP Operand:	,location	location
Logical Operations:	I.OR.E	@I,@!,@E
	J.AND.-E	@J,@&,@-E

F.2 ARRAY IMAGE RESIDENCE

From I4DDT's viewpoint, an ILLIAC program consists of an array image which is composed of a set of control registers and a collection of array memory words. At any given time, the program image is either loaded in ILLIAC or residing in the address space structure of I4DDT. Generally, the sequence which is followed when running an ILLIAC program with I4DDT is as follows:

1. Get the program image into I4DDT's address space.
2. Make any desired changes to the image.
3. Load the image into the array.
4. Start the program.
5. Wait for program completion or error.
6. Dump the image back into I4DDT's address space.
7. Examine the dumped image to find bugs or look at results.
8. Optionally, dump the image from I4DDT's address space into a file (for later examination).

It should be noted that most I4DDT commands executed while the ILLIAC array is acquired (i.e., image loaded) behave exactly in the same manner as if the commands were executed when the image was in the address space of ILLIAC.

F.3 CONTROL REGISTERS AND ARRAY MEMORY SUBSPACES

The ILLIAC space actually consists of two subspaces, control registers and array memory, although for the most part this fact is not evident. I4DDT switches between the two subspaces as is necessary. The control registers address space is entered only when examining or changing control registers. The only way in which control registers may be examined and modified is by typing the name of a control register followed by the appropriate command character(s), making any desired changes, and then typing a control character (e.g., carriage return, line feed, up-arrow). When a carriage return is typed, I4DDT closes out the currently open control register and switches back to the array memory subspace. If, however, the user types an up-arrow or a line feed, I4DDT remains in the control registers subspace and automatically examines the next lower or higher control register.

F.4 ILLIAC ADDRESSES

All addresses opened for examination or modification by I4DDT are assumed to be syllable addresses. However, in ILLIAC instructions, the references to array memory are either syllable addresses or row addresses, depending upon the instructions. For example:

```
10000/        LDA 2
```

indicates that syllable 10000 contains a PE instruction, LDA, which references row 2, but:

```
10001/        SKIP 2000
```

indicates a skip to syllable 2000, and finally:

```
2001/         JUMP 3000
```

indicates a jump to syllable 3000 (must be even) or word 1400 (assuming octal typeout radix).

F.5 CONTROL REGISTER REPRESENTATION

Any control registers which are less than 64 bits in length are stored right-justified within the standard 64-bit control register storage word. Registers such as ICR and IIA , which have a syllable bit in bit 0, are automatically converted into a right-justified syllable address by I4DDT. CU registers are referenced by using their names as given in the reference manual. PE registers are referenced by giving a register type followed by a 2-digit (octal) PE number. For example:

RGA14 is the name of the A register in PE #14.

RGD33 is the name of the mode register in PE #33.

F.6 CLEAN AND DIRTY ARRAY IMAGES

An array image is dirty if at least one of the following statements is true:

1. The image was created by dumping ILLIAC.
2. At least one control register had a quantity deposited into it.
3. The image was the result of getting a previously saved dirty array image.

If none of the above are true, the array image is clean. The cleanness of an image affects the way in which I4DDT initializes and loads the control registers (see paragraph F.7).

F.7 ILLIAC INITIALIZATION AND LOADING

ILLIAC array memory is zeroed before loading.

If the image to be loaded is dirty, then all control registers are loaded with their image values.

If the image is clean, then all writeable CU registers are zeroed, all PE registers are zeroed, all PE's are enabled, and the following CU registers are initialized as indicated:

ICR	initialized with its image value
ISR	initialized to 177777777
AMR	set to 177756
ACR	set to 4220
AIN	is zeroed

F.8 REGISTER PRESERVATION OVER EXAMINES/DEPOSITS

When I4DDT examines/deposits while ILLIAC is acquired, it saves and then restores all registers which it uses to perform the indicated operations. Some registers such as TRO and TRI are used so often that they are initially saved when an examine or deposit is done, and are then finally restored when ILLIAC is restarted instead of before and after every examine or deposit.

F.9 STATE OF ILLIAC AFTER INTERRUPT

When ILLIAC is started and eventually stops as a result of a HALT instruction or some masked error condition, I4DDT fields the interrupt and informs the user about the reason for the interrupt. I4DDT then returns to its command processing mode. The user may examine CU registers at this point without affecting the state of ILLIAC. However, if the user attempts to do anything else at this point, I4DDT is forced to clear ILLIAC controls and TCC before performing the requested operation. Therefore, if the user is interested in looking at some control registers (e.g., FLP, FRP, ALR), he should do so before anything else.

F.10 STOPPING A RUNNING ILLIAC PROGRAM

Sometimes it is desirable to stop an ILLIAC program to find out what it is currently doing. At the present time I4DDT allows a user to do this by typing any character on his terminal while the program is running. I4DDT then stops the program, and informs the user where the program was stopped. The user may then type commands to I4DDT.

F.11 DUMPING THE ARRAY

When ILLIAC is dumped back into I4DDT's address space as an array image, all PE registers are dumped, all readable CU registers are dumped, and all non-zero data pages are dumped.