PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

PDP-23-4

INVISIBLE DEBUGGER (ID)

Aug. 18, 1970

ID- Invisible Debugger

INVISIBLE DEBUGGER, commonly referred to as ID, is a utility program in the PDP-1 time-sharing system written to aid in the debugging of programs.

General Essential Preparation

When a time-sharing user logs in (by pressing the "console" switch on) he is given an ID. The ID can not be examined or modified by commands to ID or by execution of a program.

The first command the user types is usually E to start Expensive Typewriter, (see memo. PDP-22-3), or nF ($0 \leq n \leq 3$) to start the file system on tape unit n. (See memo. PDP-42-2, Preliminary Microtape File System).

When the user runs any program (either his own program or a system program such as Expensive Typewriter), his ID is inactive. It will be restarted if the program executes any illegal instuction or the console's "call" button is pressed (see the section on breakpoints).

After a normal assembly, the assembler returns control to ID, and ID loads the program and its symbols (see description of mta 6). Typing P will begin execution of the program at the address specified to the assembler in the "start" pseudo-instruction.

For the sake of accuracy, descriptions of commands include their effect, if any, on programs with several processes. The user unfamiliar with multiprocessing should assume that only one process exists.

Fields

ID allows operations to be carried out on the core fields of the user, or of spheres owned by the user, or on drum fields owned by the user, or absolute drum fields, by making the field involved the "current field". Initially ID is set up so that the user's core 0 is the current field.

The current field is normally specified by the underbar command. Typing:

```
        x_
```

causes field x to become the current field.

```
        ?
```

is typed by ID if the field exists, i.e. the user does not own the corresponding core or drum field.

Typing underbar (_) alone will cause the current field to be printed out.

The "field" assignment is as follows:

0-5	core fields
100-177	drum fields assigned to user
200-226	absolute drum fields (read only)
1MNN	core field M of sphere owned by user at index NN

This numbering is used for commands H, O, S, U, V, and Z. (explained later). For example, typing:

```
        224V
```

verifies the current field against absolute drum field 24.

In this memo, "field" always refers to this numbering unless "drum field" or "core field" is explicitly stated.

The "current location" is the 12-bit address of the last register opened by those commands which set the current location. The character point (.) , when not part of a symbol or number, has the value of the current location.

The character Q has the value of the last quantity typed by ID or the user.

Register Examination Commands

Command	Action
/	When preceded by an argument, slash opens the register in the current field whose address is specified by the low 12 bits of the argument, sets the current location to that address, and (unless in type-in mode) types the contents of the register in the current mode. When not preceded by an argument, slash (/) behaves just like Q/, except that it does not change the current location.
|	Similar to slash (/), but uses a 15-bit address. In particular, when preceded by an argument, it changes the current field to the core field specified in bits 3-5 of the argument, then does what slash does. Without an argument, it behaves like slash (/) except that the field specified in bits 3-5 of Q will be used, rather than the current field. The current field is not changed.
[	Similar to slash (/), but types the contents of the register as a positive number, regardless of the current type-out mode. Does not change the type-out mode.
]	Similar to [, but types the contents of the register as an instruction.
(	Similar to slash (/),but causes ID to enter type-in mode. In this mode, contents of registers are not typed out at all. Type-in mode is left when a carriage return is typed in or out, except that commands ↑, backspace and tab keep ID in type-in mode.

Register Modification Commands

Fields not assigned to the user can not be modified.

Command	Action
carriage return	If an argument was given and a register is open, deposits the argument in the register. The register becomes closed. Note: whenever carriage return is typed by ID (after certain commands), any open register becomes closed; however the register is not modified.
backspace	Same action as carriage return, and in addition opens the next register the way .+1/ does. This command types a carriage return, the current field (unless zero), the address (in the address type-out mode), a slash (/), tab, and (unless in the type-in mode) the contents of the register (in the current type-out mode) and another tab. Note that if . was 7777, location 0 of the current field will be opened.
↑	Like backspace except that it opens .-1
tab	Same action as carriage return, and in addition opens a register the way slash (/) does. That is, the low 12-bits of the argument to the tab (or, if no argument, Q) will be used as an address in the current field. The address is typed as for backspace.
>	Without argument, same as slash (/). With an argument, deposits the argument in the currently open register (if any), then does what slash (/) does except that the current location (.) is not changed.

Type-out Mode

extra, D

Command	Action
S	Without argument, sets the type-out mode to symbolic mode. This is its initial state. The contents of registers will be typed as symbolic instructions unless the mode is overridden with [,], =, ~, or →. Numbers in the range 777700 to 777777 are typed as negative numbers.
C	Without argument, sets the type-out mode to constants mode. The contents of registers will be typed as unsigned numbers, unless the mode is overridden.
T	Without argument, sets the type-out mode to text mode. The contents of the registers will be typed out as flexo code characters unless the mode is overridden.
→	Types the argument (if none, Q) as a symbolic instruction, regardless of the type-out mode. Neither opens, closes, nor modifies any register.
=	Types the argument (if none, Q) as an unsigned number. Neither opens, closes, nor modifies any register.
~	Types the argument (if none, Q) as flexo code characters. Neither opens, closes, nor modifies any register.
R	Without argument, sets the address type-out mode to symbolic (Relative) mode. This is its initial state. Addresses (printed by commands backspace, tab, ↑, etc.,) will be typed symbolically.
O	Without argument, sets the address type-out mode to octal mode. Addresses will be typed out as numbers.
numR	Sets the radix to num. Num is interpreted as a decimal number, regardless of the previous radix. The radix is initially 8.

H	(Hoctal) Without argument, same as 8R
U	(Unoctal) Without argument, same as 10R

Symbols

A symbol is a string containing only lower case letters, numerals and point (.), containing at least one letter, and having a value associated with it.

Initially the symbol table contains the PDP-1 instruction mnemonics. The most common way of entering symbols in the ID symbol table is by executing a mta 6 (see the section on breakpoints).

Commands	Action
1T	Reads a binary symbol table (prepared by the assembler) and merges it with ID's symbol table.
2T	Reads a symbol table in the user's core (left by the assembler) and merges it with ID's symbol table. The format is:

00100	first address of symbol table
00101	last address of symbol table
00102	number of symbols

fH	(Hoard) Saves ID's symbol table (except initial symbols) on field f. ID prints the lowest address occupied by the symbol table, which extends from there to 7777 on field f.
fO	(Obtain) Kills all symbols like K, then reads the symbol table stored on field f by an "H" command.
x<fH	Like fH, except the symbol table will extend from the address printed by ID to x-1, inclusive.
x<fO	Like fO, except reads symbol table stored by an "x<fH" command.

val sym)	Defines symbol sym with value val.
sym,	Defines symbol sym with the current value of the current location counter (.).
symD	Defines the symbol sym with the value equal to the low 12 bits of Q.
symK	(Kill) Deletes symbol sym from the symbol table.
K	With no argument, deletes all symbols except the PDP-1 instruction mnemonics. If these were redefined, the original value is restored.
symC	(Calm) Half kills the symbol sym. ID will recognize it when typed in, but will not type it out.
symL	(Loud) Undoes the effect of symC, so ID will type out the symbol in symbolic instructions.

Type-in

Numbers are interpreted in the current radix, unless immediately followed by a point (.), in which case they are taken as decimal.

Microprogram instructions may be typed in, but to distinguish them from commands, they must be preceded by a single quote ('), for example:

'SXXP

The special character overbar (¯) has a value of -0.

If an undefined symbol is typed, ID types "U" and ignores everything typed since the last tab or carriage return.

Command	Action
?	Typing a question mark (?) causes ID to ignore everything typed since the last tab or carriage return.
"	Causes the previous 1, 2, or 3 characters to be taken as their flexo code value.
'	Causes the previous 1, 2, or 3 characters to be taken as their squoze code value.

The following connectives are performed in left-to-right order.

space	Same as +
+	18 bit 1's complement addition
-	18 bit 1's complement subtraction. Unary - is complementation, so -0=777777
∧	Logical and
∨	Logical or

Special Registers

The following special registers may be referred to from any field, and opened and modified in the same manner as other registers, except that [and] do not change the current location. For example, to examine the accumulator, type:

```
A/
```

The following registers appear to be consecutive; i.e., backspace, ↑, and expressions like .+3 will work for them the way they do for ordinary registers.

These special registers refer to the current user process (see centerdot (•) command).

A	Accumulator
I	In-Out Register
X	Index Register
F	Flag Register, which contains

```
 0   1   2   3   4   5 . . . . . . . . 12  13  14  15  16  17
                                           Program Flags
AMD AEF AAL SBM SBH PRL                 1   2   3   4   5   6
```

bits 0-1 are decoded as the address mode

00	nam	normal mode
01	bam	base mode
10	iam	index mode
11	dam	defer mode

G	Program Counter, bit 0 contains overflow, bit 1 contains extend mode.
W	W Register, (used for communication with the supervisor)

The following special registers are used to control certain functions within ID.

x	The location for executes (see x command), normally 7775
M	The mask for word searches, normally 777777
M+1	The lower limit for word searches, save, unsave, etc., normally 0
M+2	The upper limit for word searches, save, unsave, etc., normally 7777

Note: only the low 12 bits of M+1 and M+2 are used.

B	Breakpoint locations
B+1	
B+2	
B+3	

When preceded by an argument, the commands A, I, X, and M deposit the argument in the corresponding register. The command x<yM puts x in M+1 and y in M+2.

Typing:

M^{-}

Resets M, M+1, and M+2 to their initial contents.

B^{-}

Resets all breakpoint registers.

Searches

Command	Action
W	(Word) With an argument, searches the current field between the limits in M+1 and M+2 (inclusive) and prints all registers which are equal to the argument when both are and'ed with the mask (M). In other words, only those bit positions which are 1 in the mask are compared.
N	(Non-word) With an argument, searches the current field between the limits in M+1 and M+2 (inclusive) and prints all registers which are not equal to the argument when both are and'ed with the mask. The most frequent use of this command is ON, a search for non-zero locations.
E	(Effective address search) With an argument, searches the current field between the limits in M+1 and M+2 (inclusive) and prints all registers which effectively adddress the argument; ID assumes that instructions are executed in normal mode, not under aam's. Bit 1 of the G register (extend mode) controls the way ID interprets indirect address chains. An E search never prints skp, sft, law, iot, ivk, or opr instuctions. This command is of limited value if the program being debugged does any indexed addressing.

Miscellaneous

Command	Action
Z	With no argument, sets all of the current field to zero.
fZ	Zeros field f between the limits in M+1 and M+2 (inclusive).
x<yZ	Zeros the current field between x and y (inclusive).
fU	(Unsave) Copies the contents of field f between the limits in M+1 and M+2 into the current field between the same limits. For example, if M+1 and M+2 are reset to their initial contents (which they usually are), 101U will copy all of the drum field at user's capability index 1 into the current field.
x<fU	Copies the contents of field f between M+1 plus x and M+2 plus x into the current field between M+1 and M+2 . For example, O<fU is the same as fU. Note that M+1 and M+2 refer to the current field, and addresses in field f are offset by x.
fS	(Save) Like fU, except data moves from the current field to field f.
x<fS	Similiar to x<fU. Copies the contents of the current field between the limits in M+1 and M+2 onto field f between M+1 plus x and M+2 plus x.
fV	Same as O<fV
x<fV	For each address j, within the limits in M+1 and M+2, compares location j in the current field with location j+x in field f. If they differ after both are and'ed with the mask (M), ID prints;

```
j/    contents of current field,    contents of field f,
      location j                    location j+x
```

Commands for Paper Tape

Command	Action
Y	(Yank) Without argument, reads a tape in standard binary format into the current field. Words outside the limits in M+1 and M+2 are ignored. The core modules specified in the data block origins are ignored. The address in the start block is put in the G register (Program Counter).
fY	Like Y, except reads into field f.
x<yY	Like Y, except uses x and y instead of M+1 and M+2 as limits.

Note: if a checksum error occurs while reading, ID types "cksm". By moving the tape one block back and typing c, ID will read the block again. Any other character will cause the block to be accepted as read.

L	(Label) Letters typed after this command are punched in readable form on tape.

There are three characters which terminate the label, having the following effect:

carriage return	Punches the standard input routine and sets ID to punch standard binary blocks.
backspace	Punches a "jmp 7752" instead of the input routine and sets ID to punch standard binary blocks.
tab	Sets ID to punch readin mode tapes, i.e., alternate dio instructions and data words.
D	(Dump or Data) Without argument, same as 0<7777D
x<yD	Punches the current field from x to y (inclusive). in the format set by the L command. If the current field is a core field, the data block origins will be in the current field. Otherwise they will be in core 0.
xD	Like D, except punches between the limits in M+1 and M+2. The data block origins will be in core x.

xJ (Jump) Punches a start block to address x

Breakpoints

Breakpoints are the primary tools for debugging programs. They provide a convenient means for interrupting a program to examine intermediate results, trace the flow of control, etc. The four breakpoint registers B, B+1, B+2, and B+3 contain the addresses of breakpoints in the user's program. No breakpoint is indicated by an overbar (¯). The command xB puts the low 12 bits of x in the low 12 bits of special register B, and puts the current field number, if less than 6 (a user core), and bits 3-5 of x otherwise, in bits 3-5 (the core module) of register B.

Whenever a process attempts to execute an instruction (either directly, or through execute (xct) instructions) which has a breakpoint in it, (except during a multiple proceed- see below), ID resumes control. The instruction at the breakpoint will not yet have been executed. All processes are stopped, and ID types the program counter of the process that reached the breakpoint (which is the same as the location of the breakpoint unless the breakpoint was reached through xct instuctions), a close paren, tab, and the instuction that was being executed. The current location is set to the address in the program counter, that register is opened, bits 3-5 of the program counter (the core module) become the current field, and the process that reached the breakpoint becomes the current process. ID resumes listening for commands from the typewriter. A breakpoint remains in a location until either moved by changing the contents of the breakpoint registers, or removed by depositing overbar (¯or -0) in the breakpoint register, typing B¯(which clears all breakpoints), or by commands E or nF (see below).

CAUTION: Breakpoints should not be placed at locations which are examined or modified by the program, since ID saves their contents and substitutes the trap (bpt) instruction.

The command P (Proceed) is used to resume execution after a breakpoint has been reached. The instruction at the breakpoint will be executed once, and the user's process will continue running. If the breakpoint is reached again it will trap again. xP will proceed through the breakpoint x times before trapping to ID. If a different breakpoint is reached, during a multiple proceed, it will trap immediately regardless of the proceed count.

The P command may be used even when no breakpoint has been reached (e.g. when ID is entered by hitting the call button). In this case, ID simply starts all processes running, giving control of the typewriter to the user.

The command -nP starts the user running so that after n instructions have been executed by the current process the program traps to ID as though a breakpoint had been reached. If a breakpoint had been reached when the command -nP is typed, it is removed. This command is used to closely follow the progress of a program.

The command x× may be used to execute single instructions or subroutine calls. ID places the instruction x in the current field (which must be a user core field) at the address specified by the low 12 bits of the × register, puts that address in the program counter of the current process, and starts all processes (usually this is the only process). When the user's program counter becomes equal to the original program counter incremented by one or two, ID resumes control. The program counter is restored to the contents that it had before the × command. The effect is to execute the instruction x, but if x is a call to a subroutine, the entire subroutine is executed. If the program counter was incremented by one, ID types two carriage returns after the ×; if the program counter was incremented by two, (i.e., the instruction skipped), three carriage returns are typed.

y<x× behaves like x× except the address for the instruction is specified by y, not the × register.

The call button is used to return control to ID. ID prints and opens the register being executed as though a breakpoint had been reached. If the user has no process, ID types "no proc". The instruction printed will not yet have been executed (except for certain types of ivk instructions). If the call button is pressed while ID is already in control, ID stops whatever it was doing and listens for commands.

When a user process executes a mta 7 (dsm), ID is brought into control, all processes are stopped, and ID types a carriage return and listens for commands.

When mta 6 is executed, the above action occurs and in addition, ID puts the process's AC into its PC, and behaves as though the sequence K, 2T, and 101U were typed. (The assembler does this to load a program, define its symbols, and set up its starting address.)

Miscellaneous

Command	Action
x<symF	Execute an arq instruction with x in the IO and the concise code for sym in the AC, on behalf of the user. No registers of any process are changed. Two carriage returns are typed if the arq did not skip, three if it did. x may be omitted when not applicable.
x<y<zF	Executes a mta z with y in the AC and x in the IO on behalf of the user (i.e., 206F reads the user's, not ID's, memory bound). No registers of any process are changed. Two carriage returns are typed if the argument did not skip, three if it did. If x or x and y are omitted, zero is used. If the mta returns information in the AC, it is printed.
nF	If $0 \leq n \leq 3$, starts the file system on tape unit n.
E	Without argument, starts Expensive Typewriter. Note: These two commands reset all breakpoints, reset the mask registers, and set the number of processes to 1.

Error Indications

busy

An attempt to assign the reader, punch, or microtape unit failed. The user must wait until the device becomes available.

sym ovf

ID's symbol table has overflowed. If this occurs during a 1T or 2T, ID will continue reading symbols, but will only redefine symbols already in the table. No new symbols will be added.

?

General purpose error message. ID can't do or doesn't understand the request typed in.

de

A parity error occured while reading the drum, or an attempt was made to write on an absolute drum field.

U, cksm, and no proc are discussed elsewhere.

When a user process executes an illegal instruction, ID is brought into control and types the address of the illegal instruction and the instruction as indicated below.

Instruction	Printout
hlt	adr↑↑ inst
illegal memory reference	adr<< inst
IO function busy	adr?? inst
lock fault	adr⊃⊃ inst
mta 4	adr∨∧ inst
mta 5	adr∧∨ inst
any other illegal inst.	adr>> inst