PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
M.I.T.
CAMBRIDGE, MASSACHUSETTS 02139

PDP-23-3

ID

September 5, 1967

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 other programs. An advanced ID has been written (April, 1966) to allow all operations to be carried out either directly on drum fields or on running cores. It uses the drum to allow the user full use of core (s) and drum field (s) for his program and to provide extra features. ID and the program being debugged each have a drum field to themselves.

For clarity when typing examples are given herein, the typing done by the user of ID is underlined. Also, when needed the following symbols are assigned to the invisible flexo characters:

carriage return 

tabulation →|

space 

backspace ←

upper case 

lower case 

A. General Essential Preparation

1. When a time-sharing user requests ID he is automatically assigned one drum field to be used for his ID program. The user's running field, which was assigned when the console was turned on, and the console's pseudo drum fields will be used as the console's drum and core fields whose contents may be examined and modified by the use of ID. The drum field assigned upon requesting ID is the console's ID field and may not be examined or modified by instructions to ID or by execution of a program.

2. When entering ID, the user has either a binary tape containing the program and its symbols or a binary version of his program existing on his pseudo field 1 with its symbols still in the POSSIBLE SYMBOL TABLE located in the console's running field (i.e., in core 0.)

   a. Program on Field 1 and Symbols in POSSIBLE SYMBOL TABLE: To inform ID of the meaning of the symbols used in the program type:

   2↑TW

   ID will then take a copy of POSSIBLE SYMBOL TABLE and put it into its own ID SYMBOL TABLE. To get a copy of the binary program from pseudo field 1 and place it into the console's current field so that it can be executed, type:

   1↑UW

   (NOTE: If changed, the limits M+1 and M+2 should be initialized before the above command by typing "M⁻".)

Program and Symbols on Tape: To clear all available registers of the current field memory, type:

⇑Z ⇓)

This command will zero all the registers of the current field. Then to kill the previous symbol table, leaving only the initial PDP-1 instruction mnemonics, type:

⇑K ⇓)

To read in the binary tape containing the program or data, place the tape in the reader and type:

⇑Y ⇓)

This causes ID to yank a standard binary black format tape into the current field memory. To inform ID of the meaning of the symbols used in the program, place the symbol tape, which was prepared by POSSIBLE or MIDAS SYMBOL PUNCH, in the reader. If the tape is a binary tape from MIDAS type:

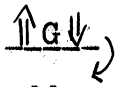
1⇑T ⇓)

If it is an alphanumeric tape from POSSIBLE type:

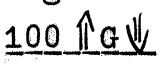
⇑T ⇓)

ID will then read in the symbol tape and will merge the contents of this tape with ID's own symbol table. After this, ID is ready for use and will be able to interpret constants and instructions typed either symbolically or numerically or both.

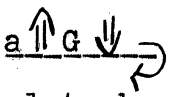
3. Typing



preceded by the address where the user wants his program to begin, will cause the program to start running. For example, typing:



or



will cause control to be transferred to absolute location 100 or symbolic location "a" respectively.

4. To return control to ID after using a "G" command, press the console's CALL BUTTON.

B. The Current Field

This new version of ID (April, 1966) allows operations to be carried out directly on drum fields or on the user's running cores by making the field involved the "current field." Initially ID is set up so that the user's running 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". If $x \leq 177$, x itself is used; otherwise, bits 2÷5 of x are used. Fields 0 to 7 refer to the user's normal running core. For time-sharing users, only 0 and 1 are legal, and 1 is legal only if core 1 is assigned to the user. Fields 10-100 are illegal. Fields greater than (>) 100 refer to drum fields. For example, typing

103_

makes the user's pseudo field 3 the current field. Absolute field are indicated by bit 12. For example,

161_

causes absolute field 21 to become "current". Typing underbar (_) alone will cause the current field to be printed out.

C. Examination and Modification of Stored Information

1. Opening a register in the current field - In using ID, a fundamental idea is that of opening a register so that its contents may be examined and/or changed. This may be accomplished by typing the twelve-bit address of the register in the current field to be opened, either symbolically or as an absolute constant, followed by a slash. For example:

reg+2/

or

2467/

When the above is typed ID will immediately print a tabulation, then the contents of that register in the current field, followed by another tabulation. Continuing the example above:

reg+2/ →| add loc+3 →|

(NOTE: Current drum fields not assigned to the user cannot be examined.)

2. Examination of a register not in the current field - It is frequently desirable to open a register not in the current field so that is contents may be examined and/or changed. This is accomplished by typing a 16 bit extended address of the register to be opened, either symbolically or as an absolute constant, followed by a vertical bar. The corresponding core module will become the "current field". For example:

12345|

will cause core 1 to become current, and open register 2345. Typing

i reg|

will open register reg in core 1 and make core 1 current. (Note: i=10000 in POSSIBLE and ID symbol tables.) Like slash, when the above is typed, ID will immediately print a lower case, tabulation, then the contents of the register, followed by another tabulation.

(NOTE: For time-sharing users, reference to core 1 is legal only if core 1 is assigned to that user.)

3. Modifying and closing a register - Once a register has been opened in either of the above manners its contents may be modified, if desired, by typing the change either symbolically or as a constant. For example:

reg+2/ →| add loc+3 →| add loc+5

(NOTE: System fields or drum fields not assigned to the user cannot be modified.)

A command character which may be helpful in modifying registers is Q. This has the value of the last quantity typed by ID or you. For example, to change the contents of register 50 from 155 to 157 type:

50/ →| 155 →| Q+2

However, the modification is not placed in memory until the user types one of the three terminating characters - up arrow, backspace, or carriage return. The effect of each of these characters is given in the following table:

Terminating Character	Action
	Returns carriage and modifies the contents of the open register if a modification has been typed. The register becomes closed. If a vertical bar was used to open the register, bits 2 through 5 indicate the core module that becomes "current".
←	Same action as carriage return except in addition the next sequential register in the current field is opened automatically (i.e., current field plus the address is typed followed by a slash tab, and the contents of the register). If no register is open when the backspace is typed, the next sequential register in the current field is still opened. NOTE: If the current location is 7777, register 0 of the current field will be opened next.
↑	Same action as backspace except this character opens the preceding register of the current field instead of the following one. NOTE: If the current location is 0, register 7777 of the current field will be opened next.

Once a particular register has been closed by use of either the carriage return, backspace, or up arrow, further modifications of that register is impossible until it is opened again.

4. Additional Interpretation of Register Contents - If, while a register is open, any one of the following characters is typed, the contents of that register will be reprinted in the indicated manner.

Character	Interpretation
=	types out quantity as a constant
→	types out quantity as an instruction
~	types out as if quantity is a concise code.

To illustrate the use of these interpretation characters, consider the following examples:

reg+100/ →| lac abc →| = →| 202147 ≥ →| lac abc

reg+101/ →| dac 6251 ~ →| ubr

where abc has the value 2147.

5. Examination and modification of a deferred register - Once an in truction has been typed out by ID, it is frequently-desired to know the contents of the register addressed by the instruction. The control characters tab (→|), greated than (>), and special uses of slahs (/) and vertical bar (|) provide this facility.

a. After opening a register, the character tab (→|) may be typed to close that register and open the register in the current field addressed by its instruction. This causes the location counter, a register internal to ID which contains the address of the last register opened, to be changed. An example follows:

200/ →| lac abc →| ≥|

abc/ →| 30 →| ⇐

abc+1/→| 0 →|

Modifications may be made to a register while it is opened during this procedure. For example:

200/ →| lac abc →| lac abc+1 →|

abc+1/→| 0 →| 5

b. Like tab, the character > can be used to find out the contents of the register in the current field addressed by its instruction. Unlike tab, it just prints a tab and the contents (not the address followed by a slash.) It opens, modifies, and closes registers in the same manner as tab. The current location counter is not changed. For example:

200/ →| lac abc →| ≥ 30 →| ⇐
201/ →| dac bop

c. The character / when used while a register is open closes the register without making any modifications to it and types out the contents of the register in the current field which was last typed by you or ID. The location counter is changed to the new register opened. For example:

200/ →| -ac abc →| / →| 30 →| ⇐
abc+1/ →| 0→|

or

200/ →| lac abc →| 100/ →| dac 1 →|

d. Like /, the character | when used while a register is open closes the register without making any modifications to it and types out the contents of the 16-bit addresses register which was last typed by you or ID. The location counter and the current field are both changed according to this new register opened. For example:

1200/ →| i+def →| ⊥ →| 2150 →| ⇐
i def+1/ →| 1567 →| (i=10000)

Notice that core 1 was made the current field and the location counter was changed to core 1 location def.

D. The Current Location Counter

The current location counter is a register internal to ID which contains the address of the last register opened in the current field. To re-open a register that has accidently been closed or to refer to registers near the one presently opened, the current location character, point (.), is used. Typing an address followed by a register opening characters such as slash or vertical bar sets the current location counter to that address. Backspace, up arrow (↑), and tab automatically set this register to the appropriate address; carriage return does not affect it. Since point (.) has the value of the current location, expressions such as dap .+1 may be typed into ID (although they will not be typed out in this format).

E. Symbols and the Symbol Table

1. A symbol is a string of not more than six letters and numerals, containing at least one letter, and having a value associated with it. ID maintains a table of symbols and their values, and uses it to interpret symbolic words.

2. Initially, ID's symbol table contains 110 symbols, corresponding to PDP-1 instruction mnemonics, such as the operation mnemonics like lac, tyo, etc., the indirect bit i, the shift mnemonics 1s, 2s, etc.

3. There are five different ways of adding six character symbols to ID's symbol table.
   a. A binary symbol tape may be prepared by an assembler and entered into ID by typing 1T. This causes the tape to be read and merges the symbols with ID's symbol table.
   b. An alphanumeric or numeric tape may be prepared by an assembler and entered into ID by typing T. This causes the tape to be read and merges the symbols with ID's symbol table.
   c. Symbols may be left in core by an assembler and entered into ID by typing 2T.
   d. Symbols may be defined directly by means of a close parenthesis as in the following example:

      2475 sym) →|

      The value 2475 is then associated with sym. Symbols may be redefined in this manner. (Even the initial PDP-1 mnemonics may be redefined, but there is rarely any reason to do so.) The redefinitions can be in terms of their old value:
      If

      abc=50

      the command

      abc+5 abc)

will make

abc=55

a symbol may be defined while a register is open by also using the close parenthesis. This would define the symbol to be the contents of the open register. for example,

1/ →| 720307 →| dpy)

defines dpy to be 720307.

e. Symbols may be defined to be equal to the current location by typing the symbol followed by a comma. This does not affect the contents of the current lo ation. For example, if the register last opened was 50:

50/ →| lac 774 →|

by typing

sym,

sym is defined as 50 but the register still contains lac 774.

sym/ →| lac 774 →|

f. Symbols may also be defined to be equal to the 12-bit address part of the last expression typed by the user or ID by typing the symbol followed by an imply sign (⊃). Thus:

500/ →| add 256 →| con⊃

./ →| add con

Thus, con was defined to be 256.

4. Symbols may be destroyed by using the commands K or symK where sym is a symbol. The command K kills all the symbols in ID's table except the 110 PDP-1 instruction mnemonics. (If any of these were redefined, however, the original value is not restored.)

5. If a symbol which has not been defined is typed, ID types a capital U (undefined) and idnores the entire line.

F. Typing Instructions, Constants, and Location

1. Instructions, constants, and locations, which collectively may be referred to as words, may be typed by the user at any time using any combination of numbers an/or defined symbols separated by appropriate connectives such as plus and minus signs. In ID, a symbol is any combination of letters and numbers not longer than six characters, which contain at least one letter. (In most other versions of DDT, symbols can not be longer than three characters.)

2. The connectives used in forming words are listed in the following table along with their meanings.

Connectives	Meanings
[illegible]	adds value of next symbol or number to word.
+	adds value of next symbol or number to word.
-	subtracts value of next symbol or number to word.
∧	ands value of next symbol or number onto word.
∨	ors value of next symbol or number into word.

Thus,

Typing	Yields
add 10	400010
lac 2147	202147
lac i adr (where adr has previously been defined as 200)	
dpy-i	720007
claVcliVclf7	764207
law 144	700144

G. Evaluation of Words

1. Often it is desirable to be able to evaluate a word that is to be used in a program without actually affecting memory. This may be done at any time without opening a register by simply typing the word to be evaluated followed by the appropriate interpretation characters (see section c-4). When this is done, ID will automatically type out the appropriate interpretation of the word followed by a carriage return.

H. Notes on Symbolic Type-Out

A given register, containing only an octal number, can be interpreted symbolically in more than one way. Thus, ID may sometimes type out instructions you may not expect.

1. If several symbols are defined as having the same value, ID chooses to print out the last one defined.

2. ID will not print a symbol which has been suppressed by putting it into calm mode, although such a symbol may be typed in. The command symC changes a symbol to calm mode. The command symL changes it to the normal (loud) mode.

3. Expressions with negative terms will not type out as they were typed in; for example, if ret=adr+5, then ret-1 typed in will be typed out as adr+4. Similarly, ID recognizes the current location symbol (.) but never prints it out.

4. The symbols 1s, 2s, 3s, ... 9s are defined, but have been placed in a special part of the symbol table so as to be printed out only on shift and rotate instructions.

5. Operate group instructions and skip group instructions type out with inclusive or signs when necessary; for example, 762407 types out as claVcliVclf 7. Thus, if a register contains data which happens to be in this range, the resulting type-out may be in terms of these instructions.

6. Numbers beginning in 77_ _ _ _type out as negative.

I. Control of Modes

1. Although it has been assumed so far that ID normally prints out the contents of registers as instructions with symbolic addresses and normally interprets constants as unsigned octal numbers, a provision has been made to alter this state of affairs with a considerable degree of flexibility.

2. There are several different register opening characters from which to choose according to the type-out mode desired.

Register Opening Characters	Meaning
/	Types out the contents of the preceding 12 bit address number as symbols or constants, according to the mode.
|	Types out the contents of the preceding 16 bit address number as symbols or constants, according to the mode.
[	Types out the contents of the preceding 12 bit address number as constants, but does not change the mode.

Register Opening Characters	Meaning
]	Types out the contents of the preceding 12 bit address number as symbols, but does not change the mode
(	Does not type out the contents of the preceding 12 bit address but puts ID into the type-in mode starting at that address.

3. Type-Out Mode For Instructions - By typing one of two commands to ID, the normal mode of printout of register contents may be controlled.
   a. Symbolic type-out mode is the most often used and the one in which ID is initially. This mode is obtained by typing a capital S. The contents of registers will be printed out as symbols.
   b. Constants type-out mode is obtained by typing a captial C. In this mode, the contents of the registers are typed out a s numbers.

4. Type-In Mode is obtained by opening a register with an open parenthesis (). In this mode, ID does not print out the contents of the register at all; it is a convenient mode for typing short programs or parts of programs. This mode is left by typing a carriage return; however, backspace, up arrow, and tab keep ID in type-in mode and open the appropriate register.

5. Type-out Mode for Address of Registers - By typing one of two commands to ID, the mode of printout of register addresses (as a result of tab, backspace, up arrow, etc.) may be set.
   a. Relative mode is the one in which ID is initially. By typing capital R the mode can be obtained again so that addresses will be typed out symbolically.
      ex. adr+10/ →| lac abc →|←
          adr+11/ →| dac x42 →|
   b. Octal mode causes the register addresses to be: typed out as numbers. It is obtained by typing a capital O.
6. Constant print control - By typing one of the following two commands, the normal mode for the printout and input constants may be controlled.

Command	Resulting Action
H	all constants will be printed out as octal numbers - hoctal mode.
U	all constants will be printed out as decimal numbers - unhoctal mode.

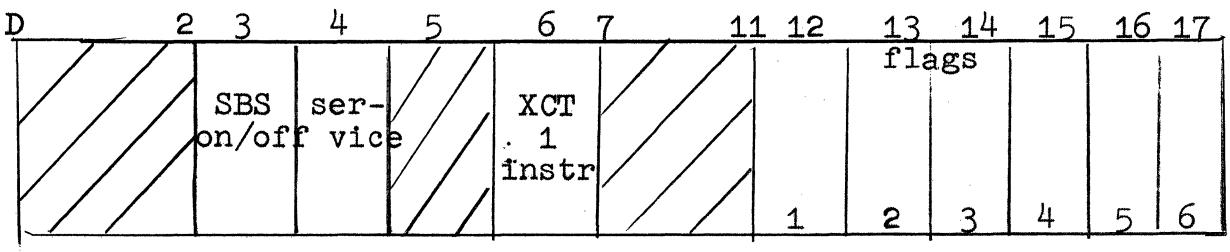
J. Input Radix Control

The current radix, used for both input and output, may be set by xR, where x is a decimal number. H is equivalent to 8R, and U is equivalent to 10R. The character period (.) is used to force interpretation of input constant as decimal regardless of the current radix. If the input constant is not immediately followed by a period, it is interpreted according to the current radix. The character single quote (') causes the last three characters typed in to be taken as their squoze code value. This applies only to the letters or numerals. The character double quote (") causes the first three characters typed in to be taken as their concise code value. This applies only to letters or numerals.

K. Special Registers

The capital letters in the following table indicate special consecutive registers, which are internal to ID. These registers control some of the main functions of ID; they may be referred from any field and are opened and modified in the same manner as a register in the current field.

Capital Letter in their Location Order	Register Contents
A	the stored accumulator of the program
I	the stored IO register of the program
X	the location of ID's execute register
G	the stored program counter of the program; the overflow flip-flop is stored in bit 0, the extend mode in bit 1.
W	dia word - the word which is stored from the IO when a dia is executed, and supplies for the following dcc. Hence, if a breakpoint is placed between a dia and a dcc, the result of the dia may be examined and modified. Also if an illegal instruction trap occurs on a dcc, the write field may be determined.
F	the stored flags of the program and its sequence break information. The format for this information is:



Capital Letter in their Location Order	Register Contents
M	the mask for word searches
M+1	the lower limit for word searches, save and unsave fields, and special uses of yank, tape, verify tape, punch data blocks, and zero memory.
M+2	the upper limit for word searches, save and unsave fields, and special uses of yank tape, verify tape, punch data blocks, and zero memory.
B B+1 B+2 B+3	breakpoint locations

The characters A, I, M, and B when preceded by a single
1 argument deposit the argument in the corresponding register. For example, typing

17777A

deposits 17777 into ID's internal register A, containing the stored accumulator for the program.

The usage of the above control characters will be more fully explained in the sections to follow.

L. Assignment and Deassignment of IO Devices and Drum Fields
ID can assign or deassign IO devices and drum fields independent of the user's program. The capital letter F when preceded by one or two arguments causes ID to execute an arq instruction. The mnenonic or concise code indicating the accumulator contents is the argument immediately preceding the F command. If it is a symbol its concise code will be used. ≺F is equivalent to executing the following three instructions

law flexo≺ or law ≺
cli
arq

(Note: The arq is executed without reference to the special internal registers A and I of ID.)
In certain cases the IO must contain additional information about the device; thus the F command must have two arguments. Typing

x≪F

will put x into the IO and the concise code for the mnemonic of the device requested into the AC and then execute an arq. If the arq skips, then two carriage returns will occur. If the arq returns information in the AC, ID prints out the information in the right 6 bits. The operation of the arq instruction is explained in meno PDP 31.

M. One of the most powerful features of ID is the ability to insert breakpoints in programs. In testing a large program, it is frequently convenient to use breakpoints

* The capital letter F when not preceded by an argument refers to ID's special internal register, F, containing the stored flags of the user's program.

to interrupt the computation so that partial results may be examined or the state of the program determined. Breakpoints may be set up at a location in the user's program by two methods:

1. Typing

   adrB

   causes ID to set up a breakpoint in the current field at location adr. Only one breakpoint can be inserted at a time by this method; the address preceding the B will be deposited into the special register B.

2. Four special registers, B, B+1, B+2, and B+3, canl be used to contain the addresses of breakpoints. No break location is indicated by an overbar($\overline{\ }$); initially all four registers contain overbars. For example:

   B+1/ →| $\overline{\ }$ →| adr

   This puts a breakpoint at location adr in the user's program. If the user transfers control to his program, and the instruction in register adr is reached, computation will cease and control will be returned to ID, which will type out the register location, a close parenthesis, tab, and the original contents of the register. At this point, the user may examine the accumulator, IO, and/or any other register and make modifications as he pleases. A breakpoint remains in the location specified until it is removed by clearing the breakpoint register containing the address. All breakpoints may be cleared by typing B$\overline{\ }$. If the user wants to clear only one breakpoint, he puts an overbar or a minus zero in the breakpoint register containing the break address to be cleared.

   CAUTION: The location selected as breakpoints must not be registers whose contents are modified by the program under test, since ID transplants their contents and substitutes specific transfer commands.

N. Go (G), Proceed (P), and Execute (X)

1. The instruction adrG, where adr is an address in the user's program, is used to start the user's program running at location adr.

2. If a breakpoint trap occurs, control is transferred to ID. To continue operation of the user's program from the point at which the break occurred, the command P is used. Even if the last breakpoint encountered has been deleted or moved, P still proceeds from the point where the break actually occurred.

3. nP, where n is a positive numeral, will cause ID to proceed from a breakpoint trap, and go past the breakpoint n times before breaking again. This multiple proceed commands applys to the last breakpoint that broke.

4. Single instructions may be executed directly by ID; control need not be returned to the user's program: There are two possible ways to execute single instructions in ID:

   a. Typing

      bX

      causes the instruction b to be placed in the address specified by the contents of the execute internal register X and then to be executed.

   b. Typing

      a<bX

      causes the instruction b to placed in address a and then to be executed. The internal register X does not change.

   Normally there are two carriage returns after X; if the PC is incremented by two (that is, the instruction skips), X will return the carriage a third time. If the return PC is not the same as the original PC incremented by one.

D. Word Searches

A valuable feature of ID is its search facility. Three kinds of searches can be made; these types are controlled by the commands N, W, and E, and they all use the special internal registers M, M+1, and M+2.

1. The three types of searches and their respective commands are:
   a. wordW - The word search causes ID to search the current field for and print out all the registers, between the limits in M+1 and M+2 inclusively, containing the given word.
   b. wordN - The non-word search causes ID to search the current field for and print out all the registers, between the limits in M+1 and M+2 inclusively, not containing the given word. This is most frequently used in ON, the search for non-zero memory.
   c. adrE - The effective word search causes ID to search the current field for and print out all the registers, between the limits in M+1 and M+2 inclusively, effectively addressing adr. If the user is in extend mode, (bit 1 of the PC on), indirect addressing chains for effective address searches will be carried to a depth of 1; otherwise they will be carried to a depth of 10, at which point ID will give up.* An E search will never print out skp, sft, law, iot, 74, or opr instructions. This type of word search is valuable for locating incorrect instructions which are modifying the program. If a jda instruction is suspected, try jda adrN

* An E-search with greater depth than 10 octal might take a long time and an E-search with no restriction on depth might get caught in an infinite chain like:

```
adr,    lac i abc
abc,    jmp i adr
```

2. The special internal registers for word searches are m, M+1, and M+2; the use of these registers is explained in the following table.

register	Contents
M	The mask register contains the value of the mask used in word searches. During word searches, only the bits masked 1 in register M are compared. Initially M contains -0; thus all bits are compared unless the register is modified.
M+1	The lower limit for the word search is stored in the M+1 register. Initially, M+1 contains 0; thus the search will begin at 0 unless modified.
M+2	The upper limit for the word search is stored in the M+2 register. Initially, M+2 contains 7777; thus the search will end at 7777 unless modified.

3. Special commands may be used to modify the contents of the special internal registers M, M+1, and M+2. Typing

M⁻

initializes the contents to -0 in M, 0 in M+1, and 7777 in M+2.

fa<laM

puts fa and la in M+1 and M+2 respectively. M remains unchanged. To change M, type

aM

where a is the mask desired for M.

4. There are two ways to print a block of registers:
   a. Set the mask to zero and set up M+1 and M+2 to enclose the area to be printed. Then search for any word.
   b. If irrelevant parts of memory happen to contain zero, merely do a N-search for zero.

P. Zero

Often it is valuable to zero all or parts of a field so that irrelevant parts of the field will contain zero. The following commands may be used:

Command	Meaning
Z	zero all of the current field
fa<laZ	where fa and la are 12-bit addresses limits for the zero command. The registers of the current field between fa and la inclusively are zeroed by this command.
xZ	where x is the field number for the zero command. The field specified is zeroed by between location in M+1 and M+2 inclusively. The current field is not changed.

Q. Yank

In the Preparation Section of this memo (part A), the user was instructed to use the command "Y" to read into the current field a binary tape. For convenience, other variation of this command may be used. They are:

Command	Meaning
Y	Read a tape in binary block format into the current field between the locations specified by M+1 and M+2 inclusively. Words outside of these limits are ignored. The core modules specified in the data block or origins will be ignored.
xY	x is the field number into which a tape in binary block format is read. Otherwise, the command is the same as Y alone. The limits of the yank are in M+1 and M+2 as above. The core modules specified in the data block origins will be ignored.
fa<laY	where fa and la are 16-bit address limits for the yank command. The data block will be checked against core field specified in the block origin. Only words with extended addresses from fa to la inclusively will be stored.

R. Verify

Another feature of ID is the ability to verify the program currently in core or on a drum field with the original binary tape. The capital letter V is used as the command in the following ways:

Command	Meaning
V	Read a binary tape in binary block format; the core modules specified in the data block origins will be ignored. The words read in are compared against the current fields words between locations specified by M+1 and M+2 inclusively. No change is made to memory, any discrepancies are typed out as: location/ memory tape
xV	x is the field number whose contents is to be compared against the tape. The field may be a core field or drum field: Otherwise, the command is exactly the same as V alone. The limits of the verify are in M+1 and M+2 as above. No change is made to memory and any discrepancies are typed out as: location/ memory tape
fa<laV	where fa and la are 16-bit address limits for the verify command. The data blocks will be checked against core field specified in the block origin. Only words with extended addresses from fa to la

inclusively will be checked. No change is made to memory and any discrepancies are typed out as:

extended location memory tape

S. Save and Unsave Drum Fields

Another valuable feature of ID is the ability to save an image of a program on another drum dield, so that it may be stored at some future time. The capital letters S and U, when preceded by additional information are command to save and unsave drum fields. * The special internal registers M+1 and M+2 indicate the limits of the transfer for the current field. The two basic commands and their meaning are:

Command	Meaning
fS	Save on field "f" - an image of the current field between the limits in M+1 and M+2 is written onto drum field f between the limits also M+1 and M+2: This operation does not affect the contents of the current field. Field "f" must be assigned to your console; it must be a number from 1 to 20 when referencing a pseudo field, or from 41_8 to 66_8 when referencing an absolute field.
fU	Unsave field "f" - the contents of the current field between the limits in M+1 and M+2 are replaced by the contents of drum "f" between the limits in M+1 and M+2. The

contents of drum field f are not affected by this operation. Field "f" must either be an absolute system field or a field assigned to your console, thus it must either be a number from 1 to 20_8 when referencing a pseudo field assigned to your console, a number from 41_8 to 66_8 when referencing an absolute field.

* The capital letters S and U when not preceded by a character mean symbolic and unhoctal. (See section I-3 and 6.)

Command	Meaning
x<fS	Add "x" to the origin of the area on field f - an image of the current field between the limits in M+1 and M+2 is written onto drum field f between the limits "x" plus the contents of M+2. Thus the limits in M+1 and M+2 apply only to the current field, not field "f". Field "f" must be assigned to your console, it must be a number from 1 to 20_8 when referencing a pseudo field or from 41_8 to 66_8 when referencing an absolute field.
x<fU	Add "x" to the origin of the area unsaved from field "f" the contents

of the current field between the limits in M+1 and M+2 are replaced by the contents of drum field f between the limits "x" plus the contents of M+1 and "x" plus the contents of M+2.
Thus, the limits in M+1 and M+2 apply only to the current field, not field "f". Field "f" must either be a number from 1 to 20_8 when referencing a pseudo field assigned to your console, a number from 41_8 to 66_8 when referencing an absolute field.

An example of using the latter commands appears below:

100<200M
20<5S

move locations 100 - 200 inclusive from the current field to locations 120 - 220 of field 5. To restore this program material at a later time, the user would type:

100<200M
20<5U

and thus move locations 120 - 220 of field 5 to 100 - 200 of the current field.

T. Hoarding and Reading Symbols

Another feature of ID is the ability to hoard and obtain symbols, so that the symbols, may be stored and restored with the associated program. The capital letters H and O, when preceded by additional information, are commands to hoard and read symbols, * The two basic commands and their meanings are:

Command	Meaning
fH	Hoard ID's symbol table on field f - saves all of the user's symbols (except initial symbols, even if redefined) on the part between n and 7777 inclusive. The number n is printed out and becomes the new memory bound for field y. (N is also in location 7777.) This feature is intended to be used in association with "S" to save a program on lower portion of the same field. The symbols are not changed or killed in any way by "H". Any argument acceptable to "S" as a field number is acceptable to "H".
fO	Obtain the symbol table stored on field f by the command "H" and bobily appends it to ID's initial symbol table. Previous symbols in ID's symbol table are killed (except initial symbol). If what it finds on that field is not a symbol table, it responds with "?",

and ID's symbol table is killed. This feature is intended to be used with "U" to unsave a program and its associated symbols for further reference. Note that the "O" process is different from "T" in that in case of "O", current symbols are fiest killed, where as in the case of "T" new symbols read are merged with current ones. Any argument previously used by "H" as a field number can be used for "O".

Two other commands to hoard and read symbols are available for swapping the symbols to and from a specified location. These are:

Command	Meaning
x<fH	Hoard symbols on field f below location x. The number n is printed out; the table of user's symbols is between n and x-1 inclusive. (N is also in location s-1.) x may be any symbolic or numeric location and any argument acceptable to "S" as a field number may be used for f in this command. The symbols are not changed or killed any way be this command.
x<fO	Obtain symbols from field f below location x previously stored by x<fH and appends then to ID's initial symbol

table. Previously symbols in ID's symbol table are killed (except initial symbols). If what it finds on that field is not a symbol table, it responds with a "?", and ID's symbol table is killed. Any arguments previously used in the "x<fH" command can be used for "x<fO".

* The capital letters H and O when not preceded by a character mean hoctal and octal. (See section I:5 and 6.)

Punching Programs

When final corrections have been made in the user's program, the user may punch it out in its modified form. The four punching commands are L, D, center dot, and J.

1. L causes ID to listen for title. Letter typed after this command will be punched in readable from on tape. The title punch is terminated by carriage return, tab, or backspace. The result of these terminating characters is given in the following table:

Terminating Character	Result
	Punches the standard input routine and sets ID to punch the usual checksummed data blocks.
→|	Sets ID to punch the usual checksummed data blocks, but no input routine. A "jmp 7751" is punched instead.

← Sets ID to punch read-in mode tapes.

2. The capital letter D is used to punch data blocks from the current field. A variety of formats are available to the user for his convenience.
   a. fa<laD, where fa and la are any symbolic or numeric expressions, punches the current field from fa to la inclusive. If the current field is a drum field, the origins of the data blocks will be in core 0. If the current field is a core field, the origins will be in the current field.
   b. D alone is equivalent to 0<7777D. It punches the entire current field. If the current field is a drum field, the origins of the data blocks will be in core 0. If the current field is a core field, the origins will be in the current core.
   c. xD, where x is a core number 0 to 17, punches the current field between the limits in M+1 and M+2. The data block origins will be in core x.

3. aJ, where a is any symbolic or numeric expression, causes ID to punch a start (jump) block to the address specified to denote end of binary tape. The address is typed immediately preceding the J.

4. If a register is open, center dot (·) will close the register and punch its contents as a one-word data block. This is convenient if the tape needs only a few modifications, known in advance.

V. Error Indications and Corrections:

1. ID has several error alarms associated with its use. these are typed out by ID and have the following general meanings:

cksm	A sum check error occurred in reading a binary program or symbol tape. By moving the tape back one block and typing "c", ID will read the block again. If the reader is left on and "d" is typed, the block will be accepted as read.
de	Drum swap was not successful. Error may be caused by trying to write on locked field, or a timing error in drum.
busy	This indicated that the reader or punch is busy and the user must wait until available.
U	This indicated that the immediately preceding word contains an undefined symbol. ID will act as if nothing had been typed. Thus, for example, typing an undefined symbol in a word into an open register will result in "U", but typing a carriage return will close the register with its previous contents rather than zero.
?	Error has been made in the command to ID. ID can't do or doesn't understand the request typed in.
<sym>	The symbol table has overflowed. Sym is the last symbol successfully

entered. If this occurs during a T, 1T, or 2T, ID will continue reading symbols, but will only redefine symbols already in the table. No new symbols will be entered.

2. When a user's program executes an illegal instruction, ID is brought back into control and the address of the illegal instruction is typed and followed by >> and a tab. Then, the contents of that register are typed out. Below is a list of various types of illegal instructions:
   a. hlt instruction
   b. instruction with an illegal operation code.
   c. instruction which indirectly addresses a location above the memory bound.
   d. a reader or punch instruction when no assignment has been obtained for the program.
   e. arq instruction with invalid code
   f. a dcc drum instruction addressing an unassigned field or locations in core above the memory bound.
   g. a bpt instruction at a location to which a breakpoint was not assigned by the user through ID.

3. When the user of ID realizes that he has made a typing error, he may delete all that he has typed since the last carriage return or tabulation by typing a multiplication sygn (x). For example:

   loc/ →| add a →| abcx →| add abc
   ./ | add abc.

APPENDIX I
SUMMARY OF CONTROL CHARACTERS

A.	accumulator storage (19)*
B,B+1,B+2,B+3	registers containing breakpoint location(22)
C	without argument: set word print mode to constants (17)
	without argument: set symbol to calm mode (15)
D	punch data blocks (36)
E	effective address search (25)
G	without argument: storage for program counter (19)
	with one argument: start program running, go to (24)
F	without argument: storage for program flags (19)
	with one of two arguments: execute an arq (21)
H	without argument: set constant printout mode to (18)
	hoctal (octal) with one or two arguments: hoard symbols onto field (34)
I	i-o storage (19)
J	punch start (jump) block (37)
K	kill defined symbols (3)
L	without argument: listen for title punch (36)
	with argument: set symbol to loud mode (15)
M	mask register (26)
M+1	lower limit for word search (26)
M+2	upper limit for word search (26)
N	not-word search (25)
O	without argument: set location print mode to octal (18)
	with argument: obtain symbol table from field (34)

P	proceed (24)
Q	last quantity (7)
R	without argument: set location print mode to (18)
	relative with one argument: set radix (18)
S	without argument: set word print mode to symbolic (17)
	with one or two arguments: save memory on field (31)
T	read symbol table (T, 1T, 2T) (12)
U	without argument: set constant printout mode to unhoctal (decimal) (18)
	with one or two arguments: unsave field into current field (31)
V	verify tape (29)
W	word search (25)
X	execute as instruction (24)
Y	read binary tape (28)
Z	zero memory (27)
0-7	octal numerals and/or symbol constituents (14)
8,9,a-z	symbol constituents (12)
"	take as concise code (18)
~	print as concise code (9)
⊃	define symbol as address typed (18)
∨	inclusive or (14)
∧	and (14)
↑	modify and open previous register (8)
→	print as instruction (9)
(	open register in type-in mode (17)
)	define symbol (12)
[	examine register as octal constant (16)
]	examine register as instruction (17)
-	minus (14)
+	plus (14)
␣ (space	plus (14)

,	define as (13)
=	print as octal (9)
.	current location; if preceded by number take constant as decimal integer (18)
×	delete type input (39)
/	examine 12-bit address register (10)
tab	modify and open addressed register; also alters sequence of location (9)
bk sp	modify and open next register (8)
car ret	modify and close register (8)
uc, lc	set case
|	examine 16-bit address register (10)
>	modify and open addressed register (10)
!	use squoze code of preceding symbol (18)
· (center dot)	punch opened register as one word block (37)

* The numbers in parentheses indicate the page number where the character can be found.

APPENDIX II

ID SYMBOL TABLE

BASIC INSTRUCTIONS

add	400000
adm	360000
and	020000
cal	160000
dac	240000
dap	260000
dio	320000
dip	300000
→div	560000
dzm	340000
idx	440000
ior	040000
iot	720000
isp	460000
jda	170000
jdp	140000
jmp	600000
jsp	620000
lac	200000
law	700000
lio	220000
→mul	540000
opr	760000
sad	500000
sas	520000
→sft	660000
skp	640000
sub	420000
xct	100000
xor	060000

OPERATE GROUP

cla	760200
clc	761200
clf	760000
cli	764000
cma	761000
hlt	760400
→lai	760040
cmi	760100
lat	762200
→lia	760000
nop	760000
opr	760000
stf	760010
→swp	760060
xx	760400

SKIP GROUP

→clo	651600
skp	640000
sma	640400
sni	644000
spa	640200
spi	642000
→spq	650500
sza	640100
szf	640000
→szm	640500
szo	641000
szs	640000

IN-OUT TRANSFER GROUP

cbs	720056
cks	720033
→dba	720061
→dcc	720062
→dia	720060
dpy	730007
→dra	720063
eem	724074
esm	720055
ioh	730000
iot	720000
lem	720074
lsm	720054
ppa	730005
ppb	730006
rpa	730001
rpb	730002
rrb	720030
tyi	720004
tyo	730003

SHIFT/ROTATE GROUP

ral	661000
rar	671000
rcl	663000
rcr	673000
ril	662000
rir	672000
sal	665000
sar	675000
scl	667000
scr	677000
→sft	660000
sil	666000
sir	676000

MISCELLANEOUS

→clo	651600
1	10000
1s	1
2s	3
3s	7
4s	17
5s	37
6s	77
7s	177
8s	377
9s	777

TIME SHARING INSTRUCTIONS

sdl	723477
isb	720052
wat	722477
arq	722277
bpt	722177
dsm	722377
ckn	720027
rbt	720237
cac	720053
asc	720051
dsc	720050
lea	724677
lei	724577
rer	724777