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

PDP-23-1

ID

May 20, 1966

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. <u>General Essential Preparation</u>

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

      <u>2⇑T⇓</u>↵

      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:

      <u>1⇑U⇓</u>↵

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

b. <u>Program and Symbols on Tape</u>: To clear all available registers of the current field memory, type:

⇑Z⇓ ↵

This command will <u>z</u>ero 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 <u>y</u>ank a standard binary block 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

   <u>⇑G⇓</u>

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

   <u>100⇑G⇓</u>

   or

   <u>a⇑G⇓</u>

   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≤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 fields 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

    1 reg|

   will open register reg in core 1 and make core 1 current. (NOTE: 1=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 instruction 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 (→|), greater than (>), and special uses of slash (/) 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/ →| lac 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/ →| 1+def →| ⊥ →| 2150 →| ←
    1 def+1/ →| 1567 →| (1=10000)

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

D. <u>The Current Location Counter</u>

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 character 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 114 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 POSSIBLE or MIDAS SYMBOL PUNCH 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 POSSIBLE 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 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.

d. 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 location. 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 →|

e. 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 114 PDP-1 instruction mnemonics. (If any of these were redefined, however, the original value is not restored.) The command symK kills only the particular symbol sym.

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

F. <u>Typing Instructions, Constants, and Location</u>

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 and/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
■	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)	210200
dpy-i	720007
claVcliVclf (7 space	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. If clf 6 is typed into ID, it will be printed back as opr 6.

2. Symbols of four characters or more will only be printed out as the first three characters. Thus, if two symbols abc and abcd are defined as different octal values, ID will print both of them symbolically as abc. One may type the interpretive character to find which is used.

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 <u>inclusive or</u> signs when necessary; for example, 762407 types out as <u>claVcliVclf 7</u>. 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. <u>Control of Modes</u>

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 capital C. In this mode, the contents of the registers are typed out as 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 of 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 commands H and U discussed in the preceding section control the radix used by ID to interpret all numbers typed out by the users. 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 as octal regardless of the current constant radix. The character single quote (') causes the last three characters typed in to be taken as their sqoze code value. This applies only to letters or numerals. The character double quote (") causes the first three characters typed in to be taken as their sqoze 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.
F	the stored flags of the program
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 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. <u>Assignment and Deassignment of IO Devices and Drum Fields</u>

ID can assign or deassign IO devices and drum fields independent of the user's program. In this way, the user can be assured of obtaining essential equipment before starting his program running. The capital letter F when preceded by one or two arguments provides a convenient way to assign or deassign equipment directly from ID.* The table of possible request if found in the <u>Assignment and Deassignment of In-Out Equipment and Drum Fields Memo</u> (PDP-31). The mnenonic or concise code indicating the device requested is the argument immediately preceding the F command. Thus, typing <u>αF</u> where α is a mnemonic or concise code indicating the device requested, will request assignment of that device to the user's console. αF is equivalent to executing the following three instructions:

```
law flexoα   or   lawα
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

<u>x<αF</u>

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 assignment or deassignment of

* 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.

fields is successful, then two carriage returns will occur. On the successful field assignments that return information in the AC, ID prints out the information in the right 6 bits and restores the AC to its contents before the request was made. For assignments and deassignments of in-out devices (not fields) two carriage returns will be returned only on successful assignments. [On unsuccessful assignments, only one carriage return is given.] An assignment will be successful if the field(s) or device requested is not already assigned or if the assignment is already in effect.

M. Breakpoints

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 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, can be used to contain the addresses of breakpoints. No break location is indicated by an overbar (‾); initially all four registers contain overbars. For example:

    B+1/ →| ‾ →| 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^-. 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 command works only for the breakpoint whose address is in register B.

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 be 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 or two, control is transferred to the location specified by the PC. Otherwise control is returned to ID.

O. <u>Word Searches</u>

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. <u>wordW</u> - The <u>w</u>ord 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. <u>wordN</u> - The <u>n</u>on-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 <u>0N</u>, the search for non-zero memory.

   c. <u>adrE</u> - The <u>e</u>ffective 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 <u>adr</u>. 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 <u>jda</u> instruction is suspected, try <u>jda adrN</u> in addition to <u>adrE</u>.

* 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 between locations 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 prepared by POSSIBLE or MIDAS. For convenience, other variation of this command may be used. They are:

Command	Meaning
Y	Read a tape in POSSIBLE or MIDAS binary block format into the current field between the locations specified by M+1 and M+2 inclusively. The core modules specified in the data block origins will be ignored. If a checksum is encountered, the process will stop. It is then possible to move the tape back one block, restart the reader, and type "c" to continue reading, if desired.
xY	x is the field number into which a tape in POSSIBLE or MIDAS 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. If a checksum is encountered, the process will stop. It is then possible to move the tape back one block, restart the reader, and type "c" to continue yanking, if desired.
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 yank in. The process will stop on encountering a checksum. To continue, move the tape back one block, restart the reader, and type "c" to continue yanking.

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 POSSIBLE or MIDAS 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
	If a checksum error is encountered, the process will stop. It is then possible to move the tape back one block, restart the reader, and type "c" to continue reading, if desired.
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
	The program will stop on encountering a checksum. To continue, move the tape back one block, restart the reader, and type "c" to continue reading and verifying.

Command	Meaning
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

The process will stop on encountering a checksum. To continue, move the tape back one block, restart the reader and type "c" to continue verifying.

S. Save and Unsave Drum Fields

Another valuable feature of ID is the ability to save an image of a program on another drum field, so that it may be restored 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 meanings 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 in M+1 and M+2. This operation does not affect the contents of the current field. Field "f" must be assigned to your consoles; it must be a number from 1 to 17_8 when referencing a pseudo field, or from 41_8 to 57_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 field 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 17_8 when referencing a pseudo field assigned to your console, a number from 41_8 to 57_8 when referencing an absolute field assigned to your console, or a number from 61_8 to 66_8 when referencing an absolute system field.

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

Two other commands to unsave and save drum fields are available for swapping information to a different location on the drum field and the current field. These commands are:

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+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 be assigned to your console; it must be a number from 1 to 17_8 when referencing a pseudo field or from 41_8 to 57_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 an absolute system field or a field assigned to your console; thus, it must either be a number from 1 to 17_8 when referencing a pseudo field assigned to your console, a number from 41_8 to 57_8 when referencing an absolute field assigned to your console, or a number from 61_8 to 66_8 when referencing an absolute system field.

An example of using the latter commands appears below:

100<200M

20<5S

moves 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 read symbols, so that the symbols may be stored and restored with the associated program. The capital letters H and R, 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 a field and its associated symbols on the upper 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".
fR	Read the symbols stored on field f into ID's symbol table - reads all of the user's symbols previously stored on field f by the command "H" and bodily appends it to ID's initial symbol table. Previous 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 "?", and ID's symbol table is not changed. This feature is intended to be used with "U" to unsave a program and its associated symbols for further reference. Note that the "R" process is different from "T" in that in case of "R", current symbols are first killed, whereas 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 "R".

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 inclusively. (N is also in location x-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 in any way by this command.
x<fR	Reads symbols from field f below location x previously stored by x<fH and appends them 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 not changed. Any arguments previously used in the "x<fH" command can be used for "x<fR".

* The capital letters H and R when not preceded by a character mean hoctal and relative. (See section I-5 and 6.)

U. 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. Letters typed after this command will be punched in readable form 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.
←	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 core.
   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 7, 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" the tape will continue to be read.
d.e.	Drum swap was not successful. Error may be caused by trying to write on locked field, or a timing error in drum.
no L?	This indicates that user tried to punch out before obtaining a title on the tape.
busy	This indicates that the reader or punch is busy and the user must wait until available.
U	This indicates 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.

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 directly or 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 or parameters.

   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 sign (×). For example:

   loc/ →| add a →| <u>abcx</u> →| <u>add abc</u>
   <u>./</u> | add abc.

APPENDIX I

SUMMARY OF CONTROL CHARACTERS

A.	accumulator storage (19)*
B, B+1, B+2, B+3	registers containing breakpoint locations (22)
C	set word print mode to constants (17)
D	punch data blocks (36)
E	effective address search (25)
F	without argument: storage for program flags (19) with one or two arguments: execute an arq (21)
G	without argument: storage for program counter (19) with one argument: start program running, go to (24)
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	listen for title punch (36)
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	set location print mode to octal (18)
P	proceed (24)
Q	last quantity (7)
R	without argument: set location print mode to (18) relative with one or two arguments: read symbol table (34) from field.
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)

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

U	without argument: set constant printout mode to unhoctal (decimal) (18) with one or two arguments: unsave field into (31) current field
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 (13)
∨	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 typed 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)
'	take preceding constant as decimal integer (18)
• (center dot)	punch opened register as one word block (37)

APPENDIX II

ID SYMBOL TABLE

BASIC INSTRUCTIONS

add	400000
and	020000
cal	160000
dac	240000
dap	260000
dio	320000
dip	300000
dis	560000
→div	560000
dzm	340000
idx	440000
ior	040000
iot	720000
isp	460000
jda	170000
jmp	600000
jsp	620000
lac	200000
law	700000
lio	220000
→mul	540000
mus	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
lap	760300
lat	762200
→lia	760020
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
szp	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
i	010000
1s	1
2s	3
3s	7
4s	17
5s	37
6s	77
7s	177
8s	377
9s	777

TIME SHARING INSTRUCTIONS

sps	723077
sdl	723477
lsb	720052
wat	722477
arq	722277
bpt	722177
dsm	722377
ckn	720027
rbt	720217
cac	720053
asc	720051
dsc	720050
nmn	725377
nmf	725477
sbr	722577
lea	724677
lei	724577
rer	724777

```
ID 16 aug 65 part 0

2200/


bpt=iot 2177
arq=iot 2277
dsm=iot 2377
jdp=140000
adm=360000

/low end initial symbol table
low=.-2400+2033
/top end of symbol table
tst=.-2
/end of symbol table, pointer t low end
est=.-1
/symbol limit, max number.  est-sl-sl must be >2
/lowest location used by symbol table is est-sl-sl-2.
sl=1000
/number of break points
nbp=4
/number of internal registers
nir=nbp+6
/number of drum fields
ndf=26



/fixed loc in exec routine
xlc=16



2200,     jmp ent   /begin at the beginning, the king said....
```

```
define feed a
law i a
jda fee
           terminate
define dispatch low,upp
[upp-uc 44]x1000 low-uc 44
termin
define letter a,b
disp [[a+uc-44]],b
           terminate
define     bprint                   /good for up to nbp=10
b=1
repeat nbp-1,746254                 7200+b      b=b+1
           terminate


dfp,       add                      /drum field of program (real)
dpf,       add                      /used for holding dfp

aa,        746100     0                        /A
           747100     0                        /I
           746600     0                        /F
           744400     0                        /M
           744454     007201                   /M+1
           744454     007202                   /M+2
           746200     0                        /B
           bprint                              /B+1, etc.
l=0
r=1

start
```

ID part 1 1/26/65.

/this is the beginning of the real program

```
ent,        lac ids    /were we in ID?
            sza
            jmp en1    /yes
            idx ids    /we are now
            jsp ssw    /set state word pointers
            eem
            lac i acp
            dac ac
            idx acp
            lac i acp
            dac io
            idx acp
            lac i acp
            dac pc
            idx acp
            lac i acp
            dac fg

/now get the breakpoint package

            law bp
            dap . 2
            lac i bpp
            dac bp
            idx bpp
            idx .-2
sas (dac bp nbp
            jmp .-5

/get kdm pointer

            law xlc
            dap xmw
            lac i xmw
            add xc0    /kdf-aw1 = 4
            dap xmw
            lac i xmw
            lem
            dap kdm    /pointer into exec core

/get why

            jsp gy
            jmp en2
```

```
en1,        jsp gy
            sad (4      /call button
            jmp zrt
            lac dpf
            dac dfp     /restore drum field
            jmp err


/dispatch

en2,        and (77
            sas why     /check validity
            jmp .       /Leo, you goofed.
            add red
            dac bkf     /setup flag
            dap . 1
            jmp .       /go, man, go

/dispatch table

red,        . 1         /constant
            jmp zrt     /'ddt' typed from mystic
            jmp .       /arq--leo goofed again
            jmp cl      /dsm issued
            jmp ii      /illegal op
            jmp cll     /call button
            jmp lbp     /bpt given
            jmp rx      /return from X
            jmp .       /leo screwed up

/subroutine to get why, the reason ID was entered
/leaves reason in why and ac  - io preserved

gy,         dap gyx
            eem
            law xlc
            dap xmw
            lac i xmw
            add xc1
            add xc2
            sub xc3
            dap xmw
            lac i xmw
            dap xmw
            lac i xmw
            lem
            dac why
gyx,        jmp .
```

```
/random constants

acp,        050000      /pointer to ac in exec core
bpp,        050000      /pointer to bpt package in exec core
xmw,        050000      /gets into exec corè.
xc0,        000004      /04=kfd-aw1   (executive symbols)
xc1,        000017      /15=sp2-aw1   (executive symbols)
xc2,        000015      /13=brk-sp2        "
xc3,        000003      /brk-why           "
why,        000000      /reason ddt was activated
bp,         000000      /break-
            000000      /point
cn1,        000000      /pack-
            000000      /age.
xe2,        000000      /temp storage for pc
nms,        000000      /switch for G P or X
ids,        000000      /ID switch(non 0→ID running)
opc,        000000      /storage for pc during X
kdm,        050000      /pointer to kdm word in exec
bpa,        000000      /temp for pointer tracing
xe3,        000000      /temp for instruction in bpt routine
p11,        000000      /storage for proceed routine
mpc,        lac bk1 nbp         /random number for bpi-bpo
mb,         006770      /memory bound


/subroutine to set up pointers to state word and bpt pack
/sets pointers acp and bpp
/jsp ssw  to enter        ac clobbered io preserved

ssw,        dap ssx
            law x1c
            dap xmw
            eem
            lac i xmw
            add xc1     /sp2-aw1   = 15
            dap xmw
            add xc2     /brk-sp2 = 13
            dac ssx 1
            lac i xmw
            dap xmw
            lac i xmw
            dap acp     /pointer to ac
            lac ssx 1
            dap xmw
            lac i xmw
            dap xmw
            lac i xmw
            dap bpp     /pointer to bpt package
            lem
ssx,        jmp .
            0           /temporary storage
```

```
cl,     clc                       /clear the proceed indicator, dsm entry
        dac bk0
        jmp xe 3

cll,    law 2       /call button entry
        sas nms
        jmp cl1
        clc         /call while X in progress
        dac bk0     /cant proceed
        law 7427
        jda tys
        law 7255
        jmp ixe 3                 /pc=xe2 so mp3 works right


cl1,    law 7777    /normal call
        and pc
        jmp 3bp



ixe,    law 7427                  /illegal execute
        jda tys
        law 1010
ixe+3,  jda tys
        jsp lct
        jsp mp3     /setup
        jsp fet     /and get the offending word
        jda lwt
        jmp xe+1

rx,     lac pc      /X return
        sub xe2     /did X
        sas (1      /skip?
xe,     jsp lcc     /yes
        lio opc     /no skip
        dio bk0     /restore old pc for proceed after X
        jsp bpo
        jmp zrt


/drum read error routine

dre,    lac (723564
        jda tys
        lac (6534
        jda tys
        jmp lis
```

```
1bp,        dzm bkf    /breakpoint
            law 2
            sad nms
            jmp ixe    /bpt while X in progress.
            law 7777
            and bp 3   /true loc of bpt
            dac t
            law bk1    /setup
            dap 2bp
2bp,        lac .                /check for assigned breakpoint
            sad t
            jmp 3bp
            idx 2bp
            sas (lac bk1 nbp
            jmp 2bp

ii,         law 2      /illegal instruction
            sad nms    /were we X ing
            jmp ixe    /yes-illegal execute
            lio (741010
            law 7777   /check
            and pc
            dap t2     /for
            law bk1
            dap . 1    /illegal
            lac bk1
            sad t2     /instruction
            jmp 3bp 1
            idx .-3    /under
            sas (lac bk1 nbp
            jmp .-5    /breakpoint.
            dio bkf
            lac t2     /to set up bk0 after bp3
            jmp 3bp 1


3bp,        lio (55
            dio t3
            dac bk0    /set up for proceed from bpt or illegal
            jsp bpo
            lac pc
            dzm mod
            dzm lcf
            jda pad
            lac t3
            jda tys
            lac pad
            jmp ta5
```

```
/G go

bg0,        jda chk
            dzm nms     /new mode switch
            spi
            jmp err
            dap xe2     /setup address
            jsp mp3     /setup fetch and counter for bpt
            jsp bpi     /put in breakpoints
            jsp xxx     /for sequence break(same xxx as exec)
p1,         jsp lcc
            jmp p12

/X execute from location n

xe0,        spi
            jmp err
            dap xe2
            law 2
            dac nms
            lac bk0
            dac opc     /save old pc til after X
            jsp mp3
            jsp fet
            dac xe3
            jmp p1

/subroutine mp3

mp3,        dap . 7
            dzm tsf     /fetch from drum
            lac xe2
            add c4
            dac tas     /at loc=c(xe2)
            lac c4
            dac cn1     /proceed counter= 0
            jmp .

/P  proceed and multiple proceed logic

 pr0,       spi
            law 0       /single proceed
            cma
            add c4
            dac cn1
            lac bk0     /loc to proceed from
            sad (-0
            jmp err     /cant proceed
            dap xe2
            dzm nms
            law bk1
            dap p8
            dap . 1
p2,         lac .
            sad bk0     /is this proceed through bpt?
```

```
jmp p3
idx p2
sas (lac bk1 nbp

jmp p2
```

```
        lac bkf   /did we stop by xct ing a bpt?
        sza i
        jmp p1-2  /no
p8,     lac bk1   /yes
        sad bp 3  /is there still one there?
        jmp p9    /yes
        idx p8
        sas (lac bk1 nbp
        jmp p8
        jmp p1-2  /no longer there

p9,     lac p8
        jmp p3 1

p3,     lac p2
        dap p11
        jsp bpi
        jsp lcc
        lac p11
        dap . 1
        lac .
        add c4    /after putting in bpt s
        dap bp 3
        dac tas   /put back old instruction
        dzm tsf
        lac p11   /so that it will be executed
        add (nbp
        dap . 1
        lac .     /this is the instruction from under bpt
        jda dep   /put it back
        law 1
        dac nms   /setup proceed.
        jmp p1-1


/xxx routine and associated trash

xxx,    dap xx1
        eem
        lac fg
        and (23000
        sad (21000
        law 1
        lio i kdm
        ril 1
        rcr 1
        dio i kdm
        lem
xx1,    jmp .
```

```
/set the state of the user and go

p12,       jsp ssw
           laaaaaaaccccc   bk1      lac bk1
           dac bp
           lac bk1 nbp
           dac bp 1  /set up bpt package
           eem
           lac ac
           dac i acp
           idx acp
           lac io
           dac i acp
           idx acp
           lac xe2
           dac i acp
           idx acp
           lac fg
           dac i acp

/now the bpt stuf

           law bp
           dap . 1
           lac bp
           dac i bpp
           idx bpp
           idx .-3
           sas (lac bp 4
           jmp .-5
           dzm ids    /note that we are leaving ID
           lac nms
           dsm        /pass the buck to exec.
```

```
bpi,      lio (-0              /breakpoints insert into user's field
          jmp .+2
bpo,      cli                  /break points take out
          dio dff
          dap bp6
          lac bp6+2
          dac bp2+1
          law bp5
          dap dpx
          spi i
          idx dpx
bp1,      law bk1
          dap bp3
bp2,      add (nbp
          dap bp4              /gets changed
bp3,      lac .                /used by cbp
          spa
          jmp bp5+1
          dap tas
bp4,      lac .
          dac dep
          cla
          jmp dp0
bp5,      dac i bp4
          idx bp3
          sas mpc
          jmp bp2
          cla
          sad dff
bp6,      jmp .
          law (bpt
bp6+2,    dap bp4              /used as a constant above
          dzm dff
          idx dpx
          lac .
          dac bp2+1
          jmp bp1

cbp,      dap cbx              /clear all breakpoints
          law bk1
          dap bp3
          clc
          dac i bp3
          idx bp3
          sas mpc
          jmp .-4
cbx,      jmp .
```

```
fet,        lio (-0              /fetch
            dio dff
            jmp dep+2

dep,        .                    /deposit
            dzm dff
            dap dpx
            lac tas
            sma
            jmp dpx-1
dep+6,      lac tsf              /re-entry from word search
            sza
            jmp dp2
dp0,        lio dff              /re-entry from bpi, bpo
            spi i
            lac dfp
            dip t1
dp1,        law 7777
            and tas
            jda chk
            dap t1
            lio t1
            dia
            law 1
            lio dff
            spi
            add dfp
            swp
            law dep
            dcc
            jmp dre
            lac dep
dpx,        jmp .

chk,        0                    /check against F
            dap chx
            law 7777
            and chk
            sas chk
            jmp ta0
            sub mb
            sma
            jmp ta0
            add mb
chx,        jmp .
```

```
dp2,        lio dff             /fetch, deposit internal registers
            law 7777
            and tas
            sub (ac
            spa
            jmp ta0
            sub (4
            sma
            jmp dp4
dp3,        lac i tas
            spi i
            lac dep
            dac i tas
            jmp i dpx
dp4,        sub (2
            sma
            jmp dp6
            spi
            jmp dp3
            add one
            sma
            jmp dp5
            lac dep             /M+1
            jda chk
            jmp dp3
dp5,        lac dep             /M+2
            jda chk
            sub 11
            spa
            jmp ta0
            jmp dp3
dp6,        sub (nbp
            sma
            jmp ta0
            spi                 /B thru B+nbp-1
            jmp dp9
            lac dep
            sad (-0
            jmp dp3
            jda chk
            dac chk
            law bk1
            dap dp7
dp7,        lac .               /check whether already assigned
            sad chk
            jmp dp8
            idx dp7
            sas mpc
            jmp dp7
            jmp dp3
dp8,        clc
            dac i dp7
            jmp dp7+3
```

```
dp9,        lac i tas
            sma
            jmp i dpx
            law 56
            jda tys
            clc
            dac lwt
            jmp pn2

pv,         dap pvx             /punch, verify swap routines
            clc
            dac dff
            lac pvf
            sza i
            jmp pv1
            spa
            jmp pv2             /from pwd
            law 7777
            and fa
            sub lo
            spa
            jmp pv1
            dap .+5
            sub wc
            add one
            szm
            jmp pv1
hi,         lac .               /used by searches
pvx,        jmp .
pv1,        lac fa
            dac pvf
            dap lo
            jsp zd
            lac 0
            jmp pvx
pv2,        lac dep
            jmp pvx
```

```
zd,     dap zdx          /zero drum, used for searches also
        law 1
        add wrd
        sub lo
        sub est
        sma
        cla
        add est
        dap esp
        dap wc
        lac dff
        sza
        jmp zd2
        dap .+1
        dzm .
        idx .-1
        sas esp
        jmp .-3

zd1,    lac lo           /re-entry point
        add dfp
        swp+cli-opr
        dia
        lio wc
zda,    dcc
        jmp dre
zdx,    jmp .
zd3,    law i 1          /re-entry point
        add lo
        add wc
        sub wrd
        sza i
        jmp zrt
        lac wc
        adm lo
        law 1
        add wrd
        sub lo
        sub wc
        sma
        cla
        adm wc
        lio dff
        spi i
        jmp zd1

zd2,    lio lo
        dia
        lac wc
        add dfp
        swp+cli-opr
        jmp zda
```

```
sav,      nop         /or sas mst+3 for field 23 for radio astronomy.
          jda ckf
          dac t1
          lia
          lac dfp
          dac t
          jmp su1
uns,      jda ckf     /unsave
          dzm t
          dip t
          lio t
          lac dfp
          dac t1
su1,      dzm lo
          law i 1
          add mb
          dac wrd
          clc
          dac dff
          lac t
          dac dfp
          jsp zd
su2,      lac t1
          dac dfp
          law su3
          dap zdx
          jmp zd1
su3,      lac t
          dac dfp
          law su2
          dap zdx
          jmp zd3

spf,      add (20
          cli+swp-opr
          rcr 6s
          add c4
          jmp su1-5

ckf,      0
          dap cfx
          lac ckf
          and (37
          sub (ndf
          szm
          jmp err     /field number too big
          lac ckf
          rar 6s
cfx,      jmp .
```

```
ws,     spi             /W,N,E
        jmp err
        dzm tsf
        dap ws2
        lac ll
        sub ul
        szm
        jmp err
        jsp lcc
        clc
        dac dff
        lac wrd
        dac chi
        lac ll

        dac lo
        lac ul
        dac wrd
        jsp zd
```

```
ws0,        lac wc
            dap hi
            cla
            dap ws4+1

ws4,        dzm sym
            lac .
            dac t2
ws2,        jmp .                 /ea1 or ws1

ea1,        and (770000
            sad (jda
            jmp .+4
            and ci
            sza
            jmp ea2
            law 7777
            and t2

ws1,        xor chi
can,        and msk               /used as and
wea,        xx                    /sza or sza i
            jmp ws3

            law 7777
            and ws4+1
            add lo
            dap loc
            dzm lcf
            jda pad
            law 2136
            jda tys
            lac i ws4+1
            jda lwt
            jsp lcc

ws3,        idx ws4+1
            sas hi
            jmp ws4
            jmp zd3

ea2,        idx sym
            sad c10
            jmp ws3
            law 7777
            and t2
            sub mb
            sma
            jmp ws3
            add mb
            dac tas
            sub lo
            spa
            jmp dep+6
            dap .+5
            sub wc
            add one
            szm
            jmp dep+6
            lac .
            jmp ws4+2
```

```
ver,        jsp ar              /verify
            jsp lcc

vf1,        lac t
            dap fa
            lac fa
            sub chk             /lo
            sub (dio
            spa
            jmp vf2
            add chk             /lo
            sub wrd
            szm
            jmp vf2
            jsp pv
            dac chi
            lac i la
            sad chi
            jmp vf2

vf3,        lac fa
            dap loc
            dzm lcf
            jda pad
            law 2136
            jda tys
            lac chi
            jda lwt
            jsp lct
            lac i la
            jda lwt
            jsp lcc

vf2,        idx fa
            idx la
            sas rb1
            jmp vf1+2
            jsp rbk
            jmp vf1

ar,         dap arx             /assign reader
            law 51
            arq
            jmp bus
            jsp lct
            law 4642
            jda tys
            jsp soi
            jsp lct
arx,        jmp .

bus,        jsp lct             /busy
            lac (356224
            jda tys
            lac (223034
            jda tys
            jmp pn2
```

start

ID part 2 1/26/65.

```
lis,        law i 47                /initialize assignments.
            arq                     /used by ff1
lis+2,      law i 51
            arq

zrt,        lac dpf                 /zero drum routine return
            dac dfp

lse,        jsp lcc
            dzm mod
            dzm tas
            dzm iif
lss,        clc
            dac chi

lsp,        dzm wrd
            lac cun
ssn,        dip sgn
            dzm dnm
            dzm syl
n2,         dzm sym l
            dzm sym r
            clc
            dac let
            dac chc

lsr,        dzm bbf
            lio sk1
            dio wea
            init bax, lwt
            tyi
ps1,        dio ch
            law dtb
            add ch
            dap .+1
            lac .
cas,        xx                      /rar 9s or dio ch
            and (777
            dac t2
            sub (44
            spa
            jmp ln
cad,        add (jmp uc             /used as add
            dap lsx
            sub ar1                 /last no-eval routine
            spq
            jmp i lsx
            lac sym l
            ior sym r
            lio lcf
            sad (45
            dio iif
            law syl
            lio let
            spi i
            jsp evl
            jmp ev4
```

```
law 77
sad ch
dzm tas
lac (flex U
jda tys
jmp lss
```

```
evl,      dap evx   /symbol lookup
          law mst+2
          dap esk
evc,      lac est
          dap es4

evg,      dap ev2
ev2,      lac .
          spa
          jmp esn
          cla
es3,      sas sym l
          jmp esi
es4,      lac .
          sad sym r
          jmp ev3
esi,      idx es4
          idx es4
esk,      sub mst+2              /or (lac low
          sma
          jmp .+3
          add i esk
          jmp evg
          idx evx
ev3,      idx es4
evx,      jmp .
esn,      idx es4
          lac i ev2
          xor c4
          jmp es3
```

```
ev4,    dap sgn
        lac wrd
sgn,    xx              /operator and syllable addr.
        dac wrd
        lio chi
        spi
        lac lwt
lsx,    jmp .

n,      rir 5s          /number routine
        lac syl
        ral 3s
        spi i
cun,    ior t2
        dac syl
        lac dnm
        ral 2s
        add dnm
        ral 1s
        spi i
        add t2
        dac dnm
        jmp l1
ln,     add (44-12
        spa
        jmp n

        dzm let         /letter routine
l1,     dzm chi
        idx t2
        idx chc
        sas (4
        jmp ln3
        lio sym r
        dio sym l
        dzm sym r
ln3,    sub (6
        szm
        jmp lsr
        lac sym r
        mul spd+1
        div spd+2
        jmp .
        add t2
        dac sym r
        jmp lsr
```

```
uc,n+44,    lio (rar 9s         /upper case
            jmp .+2
lc,         lio ps1             /lower case
            dio cas
            jmp lsr+1

sqo,        lac dnm             /' means take decimal number
            jmp n1+1

quo,        lac sym 1                     /" means take as flexo codes
            sza i
            lac sym r
            jda spv
            lac tys
            jmp n1

q,          lac lwt             /Q means last quantity
            jmp n1

f,          lio chi
            spi i
            jmp ff
            law fg
            jmp n1-1

a,          law ac              /A means accumulator
            jmp n1-1

ir,         law io              /I means i-o
            jmp n1-1

m,          law msk             /M means mask register

            dac iif

n1,         dzm chi
            dac syl

            jmp n2

ta0,
err,        lac (743521         /?
er1,        jda tys
            law 7234            /lc, blk
            jda tys
            law i 51
            arq
            jmp lse

ser,        law 7435            /symbol table overflow error
            jda tys
            law 0772
            jda tys
            lac est
            dap sr1
```

```
sr1,        lac
            spa
            idx sr1
            idx sr1
            lac i sr1
            jda pi
            lac (741034
            jda tys
            jmp lis+2

daq,        law 7777                /⊃ defines sym as address of Q
            and lwt
            jmp .+2

com,        lac loc                 /comma defines sym as loc
            dac df1
```

```
def,    lac let         /define symbol
sk1,    sza
        jmp err
        law pn2

de,     dap dex
        lio df1
        jsp evl
        jmp df2
        law est-sl-sl
        sub est
        sma
        jmp ser         /symbol table full
        law i 1
        adm est
        dio i est
        sub one
        dap est
        lio sym r
        dio i est
        sub one
        swp
        lac sym l
        sza i
        jmp dex
        dio est
        xor c4
        dac i est
        jmp dex

df2,    dio i es4
dex,    jmp .
dot,    lio chc
        lac loc
        spi i
        lac dnm
        dac syl
        law 44
        dac t2
        jmp l1
del,    dzm iif         /end of no-eval routines, delete
        jmp pn2
```

```
val,        dac df1              /open paren, sets up value for define
            jmp lss

kil,        spi                  /K
            jmp ki5
            lac let
            sza
            jmp err
lal,        lac low              /used below
            sad i est
            jmp err
            law kil              /delete one symbol
            dap evx
            law lal
            jmp evl+2
```

```
eql,    jda eap         /print octal integer (=)
        jda opt
pn2,    jsp lct
        jmp lss

arw,    jda eap         /print as instruction (→)
        jda pi
ar1,    jmp del         /used by cad+2

                        /octal-decimal switch setup
oct,    spi i                   /H
        jmp err
        law 10
        jmp .+4
dec,    spi i           /U
        jmp uns
        law 12
        dap ops
        jmp lse

                        /symbolic-constant switch setup
smb,    spi i           /S
        jmp sav
        law pi
        jmp .+2
cns,    law opt         /C
        dap pns
        jmp lse

                        /octal-relative switch setup
oad,    law pvl         /O
        jmp tls-1
rad,    spi i                   /R
        jmp err
        law pev
        dap pa1
tls,    jmp lse

pls,    lac cad         /plus, space
        jmp ssn

4in,    spi             /minus
        dio wrd
        lac csu
        jmp ssn

uni,    jmp ssn-1               /union, ∨

isc,    lac can         /intersection, ∧
        jmp ssn
```

```
tab,        spi i               /tab
            jda dep
            dzm lcf

ta3,        dac lwt
            jsp lcc
            lio lcf
            sni
            jmp ta4
            sub (ac
            spa
            jmp ta0
            sub (nir            /internal symbol print
            sma
            jmp ta0
            add (nir
            sal 1s
            add (aa
            dap .+1   lac .
            jda tys
            idx .-2
            xct .-3
            jda tys
            jmp ta4+1
bs,         spi i               /backspace
            jda dep
            idx loc
            jmp ta3

fs,         spi i               /arrow up (forward space)
            jda dep
            law i 1
            adm loc
            jmp ta3
```

```
bac,      law opt          /open bracket (bar-constant)
          jmp .+2
bas,      law pi           /closed bracket (bar-symbolic)
          dap bax
          jmp bar
vb,       clc              /vertical bar
          dac mod
bar,      lac lcf          /slash
          dac tas          /tas used for temporary storage
          lac iif
          dzm iif
          dac lcf
          sza
          dac tsf
          lac wrd
          spi i
          jmp ta5
          lac tas
          dac lcf
          lac lwt
          dzm tsf
          jmp ta6

uc8,      spi i            /> means make corr. and open register
          jda dep
          dzm tsf
          jmp ta6

cr,       spi i            /carriage return
          jda dep
          dac lwt
          law 72
          jda tys
          jmp lse+1

eas,      law ws4+2                  /effective address search
          dap dpx
          law ea1
          jmp ws

nws,      lac sk2          /not word search
          dac wea

wds,      law ws1          /word search
          jmp ws


pbx,      jda eap          /print as bcd (~)
          jda tys
          jmp pn2
```

```
rd,         spi i                     /read binary tape (Y)
            jmp err
            jsp ar
            jmp .+2
nb,         jsp rbk
            law i 1
            add ch
            and (7777
            jda chk                   /check last address of block
            lac ch
            sub t
            dap wc
            law 7777
            and t
            add dfp
            swp
            dia
            lio wc
            law buf
            dcc
            jmp dre
            jmp nb


bgn,        jmp bg0
xec,        jmp xe0
pra,        jmp pr0

bk,         spi i                     /B
            jmp 1bk
            clc
            dac bbf
            dac let
            law bk1
            dac syl
            dac iif
            dzm chi
            jmp lsr+1
1bk,        dac dep
            law bk1
            add c4
            dac tas
            dac tsf
            jsp dep+1
            jmp lse

ovb,        clc                       /overbar
            spi
            jmp n1
            sas bbf
            jmp err
            jsp cbp
            jmp lse

start
```

ID part 3 1/26/65.

```
ttl,        spi i                 /title punch and punch format setup
            jmp err
            law 47
            arq
            jmp bus
            feed 30
            jsp lcc
            jmp tp1

jbk,        spi                   /jump block
            jmp err
            law 47
            arq       /do we have the punch
            jmp err
            lac wrd
            ior cj
            dac lwt
            feed 40
            lio lwt
            jsp pbw
            feed 520
            jmp lis
```

```
pul,        jda chk                 /lower limit setup
            dap fa
            jmp lss

pwd,        dac dep                 /punch word
            lac tas
            sma
            jmp ta0
            lac tsf
            spi
            ior mod
            sza
            jmp ta0
            law 47
            arq          /do we have the punch?
            jmp bus
            spi
            jmp .+3
            jsp dep+1
            dac lwt
            clc
            dac pvf
            lac tas
            dap fa
            dap la
            jmp pb5

pun,        spi                               /punch any length block
            jmp err
            law 47
            arq          /do we have the punch?
            jmp err
            jsp zro+1

pb5,        lac fa
            ior c77
            dac t
            sub la
            sma
            jmp pb6                 /next hundred too high
            idx t

pb4,        jsp pbb                 /pbb or pur
            lac t
            dap fa
            jmp pb5

pb6,        lac la
            dac t
            idx t
            xct pb4
            jmp pn2
```

```
vfy,    jsp zro+1               /for verify
        jmp ver

zro,    law zd                  /for zero registers
        dap zvp                 /used by block operations
        lac wrd
        spi
        jmp .+3
        jda chk
        jmp .+5
        cla
        dap fa
        law i 1
        add mb
        dac wrd
        dap la
        law 7777
        and fa
        jda chk
        dac chk
        dac lo
        sub wrd
        szm
        jmp err
        dzm dff
        dzm pvf
        law zd3
zvp,    jmp .

tbl,    spi i                   /symbol table reader
        jmp tb2
        jsp ar
        lac t4
        sad (jmp 7750
        jmp tb5
        sad (jmp 6151
        jmp tb1-1
        sas (jmp 7751
        jmp err
        dzm sym 1

tb1,    jsp gwd                 /reader-macro
        jda pz
        lac la
        sad rb1
        jmp tbn
        jsp gwd
        dac df1
        jsp de
        jmp tb1
```

```
tbn,        law est
            sub est
            sar 1s
            jda opt

tbm,        jsp rbk                 /skips rest of tape
            jmp tbm

gwd,        dap gwx
            lac la
            sas rb1
            jmp gw1
            jsp rbk
            jmp gwd+1
gw1,        dap gwa
            idx la
gwa,        lac .
gwx,        jmp .

tb5,        jsp gwd                 /read midas table
            dac sym 1
            jsp gwd
            and (177777
            dac sym r
            ior sym 1
            sza i
            jmp tbn
            jsp gwd
            dac df1
            lac sym 1
            lia
            ril 1s
            sma+spi-skp
            jmp tb5
            and (177777
            dac sym 1
            jsp de
            jmp tb5
```

```
tb2,    dzm sym 1          /table read from ts macro or possible
        sas one
        jmp pot            /see if table read from possible
        lio mst
        dia
        law low
        dac est
        add dfp
        cli+swp-opr
        dcc
        jmp dre
        lac low-1
        sub (lac 6150
        sma
        jmp err
        add mst
        spa
        jmp err
        law i 6151-low
        add low-1
        dac t3
        law low-2
tb3,    dap tb4
tb4,    lac .
        dac df1
        law i 1
        add tb4
        dap tb4
        lac i tb4
        jda pz
        jsp de
        lac tb4
        sad t3
        jmp lse
        sub (1
        jmp tb3
```

```
pot,    sas (2
        jmp err              /not possible
        lac (400006
        dac tas
        jsp fet
        sal 1
        cma
        dac opt              /twice no of symbols in possible
        idx tas
        jsp fet
        dac t1               /origin of table
gfd,    lio t1
        dia
        law 100
        add opt
        sad (-100
        jmp lse              /finished
        spa
        cla                  /full block
        add (-100
        dac t2               /word count(negative)
        cmaVlia
        adm opt              /number of remaining words
        lai
        adm t1               /initial drum address next
        lai
        ior dfp
        lia
        law buf              /initial core address - reader buffer
        dap gsb
        dcc
        jmp dre

gsb,    lac .
        and (177777
        lia
        idx gsb
        dio sym r
        lac i gsb
        sni i
        jda df1
        idx gsb
        idx t2
        isp t2
        jmp gsb
        jmp gfd
```

```
pz,t6,    0                        /permute zones. (temp storage here)
          dap pzx
          lac pz
          and (202020
          ral 1s
          xor pz
          xor c4
          dzm sym r
          lia
pz1,      cla
          rcl 6s
          add ps1+1
          dap .+2
          law 77
          and .
          dac t2
          idx t2
          lac sym r
          ral 2s
          add sym r
          ral 3s
          add t2
          dac sym r
          sni i
          jmp pz1
pzx,      jmp .

ki1,      jmp .+2
          jmp err
          sub ev2
          dac t3
          idx t3
          dac t4
          idx t4
          idx es4
          and (7777
ki2,      sub t4
          dap ki3
          add t3
          dap ki4
ki3,      lio .
ki4,      dio .
          sas est
          jmp ki2
          add t3
          jmp .+2
ki5,      law low                  /delete all symbols
          dac est
          jmp lse

fee,t2,   0                        /feed subroutine and temp storage.
          dap fex
          cli
          ppa
          isp fee
          jmp .-2
fex,      jmp .
```

```
lwt,        0                    /Q, last word typed
            dap pnx
            lac lwt
pns,        jda pi               /pi or opt
pnx,        jmp .

eap,        0                    /eql,arw,pbx common
            dap .+7
            lac eap
            dac lwt
            lac iif
            sza
            jmp ta0
            jsp lct
            jmp .

ff,         lac sym l
            sza i
            lac sym r
            jda spv
            law ff1
            dap lsx
            lio let
            law syl
            spi i
            law tys
            jmp ev4    /returns with ac proper for arq
ff1,        lio io
            arq        /do the arq with phony ac but real io
            jmp ff2
            dio io     /F interacts only with io
            dac fft    /temp storage
            jsp lcc
ff3,        jsp lcc
            lac fft
            jda opt    /type out ac since it may contain info
            dac lwt
            jmp lse    /return to listen

/since lcc not transparent, save and restore ac etc

ff2,        dio io
            dac fft
            jmp ff3
fft,        0          /temp for ac
```

```
pi,t4,      xx                       /print instruction
            dap px
            jsp pev
            lac pi
            sub ci
            spa
            jmp ppk
            dac pi
            law 72
            jda tys
            tyo
            law 71
            jda tys

ppk,        cli
            tyo
            szf 2
            jmp pvl
            law 72
            jda tys
            and (760000
            sad (sft
            jmp i66
            sad pr0+1                          /law 0
            jmp plo
            rar 1s
            sza
csu,        sub (320000              /used as sub
            spa
            jmp plo
pvl,        lac pi
            sza i
            szf 1 i
pv3,        jda opt
px,         jmp .

i66,        law 1                    /1s-9s
            add pi
            and pi
            sza
            jmp pvl
            law pa1+1
            dap pex
            lac ea
            jmp eak+2

pad,        0                        /print address
            dap px
            law 7777
            and pad
            dac pi
            clf 1
pa1,        jsp pev                  /pev or pvl
            lac (flexo +
            jda tys
            jmp pvl
```

```
ta4,        jda pad
            lio mod
            law 7221
            spi
            law 7456              /for type-in mode
            jda tys

ta5,        dzm loc
            dap loc
            lio lcf
            dio tsf

ta6,        dap tas
            jsp lct
            lac cad
            dip tas
            lac mod
            sza
            jmp lss
            jsp fet
            dac lwt
bax,        jda lwt               /pi, opr or lwt
            jmp pn2
```

```
pev,    dap pex             /symbol lookup subr
        law i 7777
        and pi
        sad (opr            /detect operates
        jmp sev
        and (760000
        sad (skp            /detect skips
        jmp sev
        clf 2
eak,    lac est
        dap ea
        clf 1

eal,    lio i ea
        spi
        idx ea
        dap psw
        spi i
        cli
        dio t6
        idx ea
ea,     lac .
        szf 2               /test for "skip or operate" or other
        jmp sko
        xor pi
        spa
        jmp eix
        lac pi
        sub i ea
        spa
        jmp eix
        szf i 1
        jmp psw
        lac i ea
        sub i ei1
        szm
        jmp psw

eix,    idx ea
        lia
        sub mst+2
        swp
        spi i
        jmp .+3
        sas mst+1
        jmp eal
        szf 2
        jmp pex
        szf i 1
        jmp pvl
        lac pi
ei1,    sub
        lia
        sza         /detect neg nums
        jmp i77
        dio pi
eiy,    jsp spt
        lac pi
sk2,    sza i
        jmp px
        szf i 2
```

```
pex,    jmp .
        cma
        dac t2    /mask
        jmp eix

sev,    dac t1              /save instruction
        lac pi
        cma+stf 2-opr
        dac t2              /mask
        jmp eak

sko,    ior t1
        sas i ea
        jmp eix
        szf 1
        xor t1
        sza i
        jmp eix
        xor pi
        lia
        and t2
        sza
        jmp eix
        dio pi
        szf i 1
        jmp psw
        lac (flexo V
        jda tys

psw,    lac .               /best symbol thus far
        dac sym r
        lio t6
        dio sym 1
        lac ea
        dap ei1
        stf 1
        szf 2
        jmp eiy
        jmp eix

i77,    law i 7777                    /numbers 77xxxx
        and pi
        sas (770000
        jmp eiy-1
        law 7254
        jda tys
        lac pi
        cma
        jmp pv3

tys,    0                   /type symbol
        dap tyx
        setup opt,3
```

```
tyl,    lac tys
        ral 6s
        dac tys
        and c77
        sza i
        jmp tyc

        sad (72
        jmp dns
        sad (74
        jmp ups
        swp
tyb,    tyo
tyc,    count opt, tyl
        lac lwt
        cli
tyx,    jmp .
```

```
dns,    lac ps1             /redundant case shift filter
        lio (72
dn1,    sad cas
        jmp tyc
        dac cas
        jmp tyb

ups,    lac (rar 9s
        lio (74
        jmp dn1

lcc,    dap lcx             /lower case, carriage return
        law 7277
        jmp lc1

lct,    dap lcx             /lower case, tab
        law 7236
lc1,    jda tys
lcx,    jmp .

so1,    rpb                 /skip over input routine
soi,    rpb                 /enter here
        spi i
        jmp so1
        dio t4

rbk,    dap rbx             /read a block into buffer
        init rb1, buf
        dap la
        dzm chi
        rpb
        dio t2
        dio t
        spi
        jmp lis+2           /start block read.
        rpb
        dio ch

        law i 1             /check for block format
        add ch
        sub t2
        and (777700
        sza
        jmp err

rb0,    rpb
rb1,    dio .
        lac i rb1
        adm chi
        idx rb1
        index t2, ch, rb0
        add chi
        add t
        rpb
        dio chi
        sad chi
rbx,    jmp .
```

```
        lac (356342              /checksum error
        jda tys
        lac (224434
        jda tys
        jsp lct
        tyi
        lai
        sas (char rc             /type "c" for continue
        jmp lis+2
        law 51
        arq
        jmp bus
        jsp lct
        jmp rbk+1

pur,    dap pb2             /punch read-in mode blocks

pu2,    lio fa
        jsp pbw
        jsp pv
        swp
        jsp pbw
        index fa, t, pu2
        jmp pux

pbb,    dap pb2             /punch binary block format
        dzm t2
        lio fa
        jsp pbw
        lio t
        jsp pbw

pb1,    jsp pv
        swp
        jsp pbw
        index fa, t, pb1
        lio t2
        jsp pbw
pux,    feed 5
pb2,    jmp .

plo,    jsp pev
        jmp pa1+1

pbw,    dap pby             /punch binary word
        repeat 3, ppb       rcl 6s
        adm t2
pby,    jmp .
```

```
/combined octal-decimal print subroutine

opt,        0
            dap opx
            dzm op1
opa,        lac opt
opb,        dac op2
            cli+swp-opr
            rcl 1s
            div opsops,          10
            sas op1
            jmp opb
            sni

            lio (20
            tyo
            lac op2
            dac op1
            sas opt
            jmp opa
opx,        jmp .
op1,        0

ci,         10000
c10,        10
```

```
/symbol print subroutine

spt,    dap spy
        lac sym l
        and (177777
        jda spv
        jsp tys+1
        lac sym r
        jda spv
        jsp tys+1
spy,    jmp .

spv,    0
        dap spx
        init spj, spd
        dzm tys
spr,    lio spv
        cla
        rcl 1s
spj,    div
        .
        dio spv
        rar 1s
        add (spl-add
        dap .+1
        lio
        spa
        ril 6s
        lac tys
        rcl 6s
        dac tys
        idx spj
        sas (div spd+3
        jmp spr
spx,    jmp .

spd,    3100
        50
one,    1

spl,    flex 01
        flex 23
        flex 45
        flex 67
        flex 89
        flex ab
        flex cd
        flex ef
        flex gh
        flex ij
        flex kl
        flex mn
        flex op
        flex qr
        flex st
        flex uv
        flex wx
        flex yz
        737200
```

```
/dispatch table, LC, UC

dtb,    disp pls, pls           /space
        letter 1, quo           /1,"
        letter 2, sqo           /2,'
        letter 3, pbx           /3,~
        letter 4, daq           /4,⊃
        letter 5, uni           /5,∨
        letter 6, isc           /6,∧
        letter 7, pul                   /7,<

        letter 10, uc8          /8,>
        letter 11, fs           /9,↑
        disp err, err
        disp err, err
        disp err, err
        disp err, err
        disp err, err
        disp err, err

        letter 0, arw           /0,→
        disp bar, err           //,?
        letter 34, smb          /s
        letter 35, tbl          /t
        letter 36, dec          /u
        letter 37, vfy          /v
        letter 40, wds          /w
        letter 41, xec          /x

        letter 42, rd           /y
        letter 43, zro          /z
        disp err, err
        disp com, eql           /,,=
        disp err, err
        disp err, err
        disp tab, tab           /tab
        disp err, err
```

```
        disp pwd, err                   /•,_
        letter 23, jbk                  /j
        letter 24, kil                  /k
        letter 25, ttl                  /l
        letter 26, m                    /m
        letter 27, nws                  /n
        letter 30, oad                  /o
        letter 11, pra                  /p

        letter 32, q                    /q
        letter 33, rad                  /r
        disp err, err
        disp err, err
        disp min, pls                   /-,+
        disp def, bas                   /),]
        disp ovb, vb                    /‾,|
        disp val, bac                   /(,[

        disp err, err
        letter 12, a                    /a
        letter 13, bk                   /b
        letter 14, cns                  /c
        letter 15, pun                  /d
        letter 16, eas                  /e
        letter 17, f                    /f
        letter 20, bgn                  /g

        letter 21, oct                  /h
        letter 22, ir                   /i
        disp lc, lc                     /lower case
        disp dot, del                   /•,x
        disp uc, uc                     /upper case
        disp bs, bs                     /backspace
        disp err, err
        disp cr, cr                     /carriage return


start
```

ID part 4 4-22-65
/private variables.

```
wrd,        0                       /quantity being assembled
sym,        0          0            /alpha symbol being assembled
chc,        0                       /character count
chi,        0                       /+0 when letter or number has been
                                    /typed since last typeout or c.r. input
let,        0                       /+0 when letter in syllable, otherwise -0
ch,         0                       /character
syl,        0                       /syllable
t,          0                       /temporary storage
la,         dio                     /last address
fa,         dio                     /first address
mod,        0                       /mode, -0 for "type-in"
opd,dnm,    0                       /decimal number
op2,        0

/constants
c4,         400000
c77,        77
c01,        010100
cj,         600000
c3,         3
c2,         020000

pvf,        .                       /punch, verify flop
                                    /0 - usually, set by zro
                                    /-0 - center dot
                                    /not ±0 - continue in pv subroutine

bkf,        0                       /breakpoint flop
esp,        dzm .                   /used by zero routine

dfi,        0                       /value for defining symbol

loc,        0                       /current location
tas,        0                       /address part for fetch or deposit
                                    /current register.  instruction part
                                    /+ for register closed, tells dep and
                                    /fet subroutine to ignore
tsf,        0                       /current examination flop
                                    /0 - external register
                                    /non-0 - internal register
lcf,        0                       /current location flop
                                    /0 - external
                                    /non-0 - internal
iif,        0                       /initial internal flop
                                    /0 - usually
                                    /non-0 - set when type A,I,M,B
bbf,        0                       /B flop
                                    /0 - usually
                                    /-0 - when B typed, not affected by uc,lc

cn3,        400000                  /special proceed counter

dff,        0                       /deposit-fetch flop, 0 dep, -0 fet
lo,         .
wc,         .
```

```
mst,        6151-low
            lac tst+1-22
            lac tst+1
            630000

buf,        0                   /reader buffer
t3,         0
t5,         0
buf+100/


pf,         0
pc,         0

ac,         0                   /Internal registers.
io,         0
fg,         0
msk,        -0
ll,         0
ul,         6277
bk1,        -0
            repeat nbp-1, -0

.+nbp/
bk0,        -0                  /switch for legal proceed
                                /if legal, has proceed address
awm,        30
t1,         0

c,

constants

start lis
```