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 \[\text{\textbackslash r}\]
- tabulation \[\text{\textbackslash t}\]
- space \[\text{.}\]
- backspace \[\text{\textbackslash b}\]
- upper case \[\text{\textbackslash u}\]
- lower case \[\text{\textbackslash l}\]
A. General Essential Preparation

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

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

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

   \[ 2\uparrow T\downarrow \]

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

   \[ 1\uparrow U\downarrow \]

   (NOTE: If changed, the limits M+1 and M+2 should be initialized before the above command by typing "M^n".)
b. Program and Symbols on Tape: To clear all available registers of the current field memory, type:

\[ \text{\texttt{Z4}} \]

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

\[ \text{\texttt{K4}} \]

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

\[ \text{\texttt{x4}} \]

This causes ID to \texttt{yank} 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:

\[ \text{\texttt{T4}} \]

If it is an alphanumeric tape from POSSIBLE type:

\[ \text{\texttt{AT4}} \]

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

\[ \text{Ctrl-G} \]

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

\[ 100\text{Ctrl-G} \]

or

\[ a\text{Ctrl-G} \]

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, typing

\[ 161_\]

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:

\[ \text{reg+2/} \]

or

\[ \text{2467/} \]

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:

\[ \text{reg+2/} \rightarrow \text{add loc+3} \rightarrow \]

(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 its 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:

\[ \text{12345|} \]

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

\[ \text{1 reg|} \]

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.

<table>
<thead>
<tr>
<th>Character</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>types out quantity as a constant</td>
</tr>
<tr>
<td>→</td>
<td>types out quantity as an instruction</td>
</tr>
<tr>
<td>~</td>
<td>types out as if quantity is a concise code</td>
</tr>
</tbody>
</table>

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/ →| i+def →| ↓ →| 2150 →| ←
   i def+1/ →| 1567 →| (i=10000)
```

Notice that core 1 was made the current field and
the location counter was changed to core 1 location
def.
D. The Current Location Counter
The current location counter is a register internal to ID which contains the address of the last register opened in the current field. To re-open a register that has accidently been closed or to refer to registers near the one presently opened, the current location character, point (.), is used. Typing an address followed by a register opening 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 \texttt{lac}, \texttt{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 \texttt{IT}. 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 \texttt{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:

      \[2475\text{ sym} \rightarrow\]

   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
   \[\text{abc}=50\]
   the command
   \[\text{abc+5 abc}\]
   will make
   \[\text{abc}=55\]
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,

\[ 1/ \rightarrow 720307 \rightarrow \text{dpy} \]
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:

\[ 50/ \rightarrow \text{lac} 774 \rightarrow \]

by typing

\[ \text{sym} \]
sym is defined as 50 but the register still contains \text{lac} 774.

\[ \text{sym}/ \rightarrow \text{lac} 774 \rightarrow \]

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 (\text{⇒}). Thus:

\[ 500/ \rightarrow \text{add} 256 \rightarrow \text{con}\text{⇒} \rightarrow \text{add} \text{con} \]

Thus, con was defined to be 256.

4. Symbols may be destroyed by using the commands \text{K} or \text{symK} where \text{sym} is a symbol. The command \text{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 \text{symK} kills only the particular symbol \text{sym}.

5. If a symbol which has not been defined is typed, ID types a capital \text{U} (undefined) and forgets the symbol.
F. Typing Instructions, Constants, and Location

1. Instructions, constants, and locations, which collectively may be referred to as words, may be typed by the user at any time using any combination of numbers 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.

<table>
<thead>
<tr>
<th>Connectives</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>adds value of next symbol or number to word.</td>
</tr>
<tr>
<td>-</td>
<td>subtracts value of next symbol or number to word.</td>
</tr>
<tr>
<td>∧</td>
<td>ands value of next symbol or number onto word.</td>
</tr>
<tr>
<td>v</td>
<td>ors value of next symbol or number into word.</td>
</tr>
</tbody>
</table>

Thus,

Typing

add 10
lac 2147
lac i adr (where adr has previously been defined as 200)
dpy-1
clavcl1Vclf
law 144

Yields

400010
202147
210200
720007
764207
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 elf 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 \( 2a, 3a, \ldots, 9a \) 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 \textit{inclusive or} signs when necessary; for example, 762407 types out as \texttt{clavcl1Valf7}. 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.

\section*{I. Control of Modes}

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

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

<table>
<thead>
<tr>
<th>Register Opening Characters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Types out the contents of the preceding 12 bit address number as symbols or constants, according to the mode.</td>
</tr>
<tr>
<td>\</td>
<td>Types out the contents of the preceding 16 bit address number as symbols or constants, according to the mode.</td>
</tr>
<tr>
<td>[</td>
<td>Types out the contents of the preceding 12 bit address number as constants, but does not change the mode.</td>
</tr>
</tbody>
</table>
Register Opening Characters

<table>
<thead>
<tr>
<th>Characters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[</td>
<td>Types out the contents of the preceding 12 bit address number as symbols, but does not change the mode.</td>
</tr>
<tr>
<td>(</td>
<td>Does not type out the contents of the preceding 12 bit address but puts ID into the type-in mode starting at that address.</td>
</tr>
</tbody>
</table>

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. \( \text{adr+10} \rightarrow \text{lac abc} \rightarrow \text{adr+11} \rightarrow \text{dac x42} \rightarrow \)
   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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Resulting Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>all constants will be printed out as octal numbers - hoctal mode.</td>
</tr>
<tr>
<td>U</td>
<td>all constants will be printed out as decimal numbers - unhoctal mode.</td>
</tr>
</tbody>
</table>

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 s quoze 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 s quoze 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.

<table>
<thead>
<tr>
<th>Capital Letter</th>
<th>in their Location Order</th>
<th>Register Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>the stored accumulator of the program</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>the stored IO register of the program</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>the location of ID's execute register</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>the stored program counter of the program; the overflow flip-flop is stored in bit 0, the extend mode in bit 1.</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>the stored flags of the program</td>
</tr>
<tr>
<td>M</td>
<td></td>
<td>the mask for word searches</td>
</tr>
<tr>
<td>M+1</td>
<td></td>
<td>the lower limit for word searches, save and unsave fields, and special uses of yank, tape, verify tape, punch data blocks, and zero memory.</td>
</tr>
<tr>
<td>M+2</td>
<td></td>
<td>the upper limit for word searches, save and unsave fields, and special uses of yank tape, verify tape, punch data blocks, and zero memory.</td>
</tr>
<tr>
<td>B, B+1, B+2, B+3</td>
<td></td>
<td>breakpoint locations</td>
</tr>
</tbody>
</table>
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.
Assignment and Deassignment of IO Devices and Drum Fields

ID can assign or deassign IO devices and drum fields independent of the user's program. 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 Assignment and Deassignment of In-Out Equipment and Drum Fields Memo (PDP-31). The mnemonic or concise code indicating the device requested is the argument immediately preceding the F command. Thus, typing \( \text{aF} \) where \( \text{a} \) is a mnemonic or concise code indicating the device requested, will request assignment of that device to the user's console. \( \text{aF} \) is equivalent to executing the following three instructions:

\[
\text{law flexo}\text{o}\text{a} \quad \text{or} \quad \text{law}\text{a}
\]

\[
\text{cli}
\]

\[
\text{arq}
\]

(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

\[
\text{x}\text{aF}
\]

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

```
adr@B
```

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 \texttt{adrG}, where \texttt{adr} is an address in the user's program, is used to start the user's program running at location \texttt{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 \texttt{P} is used. Even if the last breakpoint encountered has been deleted or moved, \texttt{P} still proceeds from the point where the break actually occurred.

3. \texttt{nP}, where \texttt{n} is a positive numeral, will cause ID to proceed from a breakpoint trap, and go past the breakpoint \texttt{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 \texttt{bx}

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

   b. Typing \texttt{a<bx}

   \texttt{a<bx} causes the instruction \texttt{b} to be placed in address \texttt{a} and then to be executed. The internal register \texttt{X} does not change.

   Normally there are two carriage returns after \texttt{X}; if the PC is incremented by two (that is, the instruction skips), \texttt{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.
0. **Word Searches**

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

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

   * 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 1 abc
abc, jmp 1 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.

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>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.</td>
</tr>
<tr>
<td>M+1</td>
<td>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.</td>
</tr>
<tr>
<td>M+2</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

\[ \text{fa} \text{la} \text{M} \]

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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>zero all of the current field</td>
</tr>
<tr>
<td>fa&lt;laZ</td>
<td>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.</td>
</tr>
<tr>
<td>xZ</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>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 &quot;c&quot; to continue reading, if desired.</td>
</tr>
</tbody>
</table>

| 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>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 &quot;c&quot; to continue reading, if desired.</td>
</tr>
<tr>
<td>XV</td>
<td>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 &quot;c&quot; to continue reading and verifying.</td>
</tr>
</tbody>
</table>
Command
fa<laV

Meaning
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 Unsavc 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>Save on field &quot;f&quot; - 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 &quot;f&quot; 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.</td>
</tr>
<tr>
<td>FU</td>
<td>Unsavc field &quot;f&quot; - 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 &quot;f&quot; 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.</td>
</tr>
</tbody>
</table>

\(^*\) 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x&lt;fS</td>
<td>Add &quot;x&quot; 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 &quot;x&quot; plus the contents of M+1 and &quot;x&quot; plus the contents of M+2. Thus the limits in M+1 and M+2 apply only to the current field, not field &quot;f&quot;. Field &quot;f&quot; must be assigned to your console; it must be a number from 1 to 178 when referencing a pseudo field or from 418 to 578 when referencing an absolute field.</td>
</tr>
<tr>
<td>x&lt;fU</td>
<td>Add &quot;x&quot; to the origin of the area unsaved from field &quot;f&quot; - 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 &quot;x&quot; plus the contents of M+1 and &quot;x&quot; plus the contents of M+2. Thus, the limits in M+1 and M+2 apply only to the current field, not field &quot;f&quot;. Field &quot;f&quot; must either be an absolute system field or a field assigned to your console; thus, it must either be a number from 1 to 178 when referencing a pseudo field assigned to your console, a number from 418 to 578 when referencing an absolute field assigned to your console, or a number from 618 to 668 when referencing an absolute system field.</td>
</tr>
</tbody>
</table>
An example of using the latter commands appears below:

```
100<200m
20<58
```

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<54
```

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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fH</td>
<td>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 &quot;S&quot; 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 &quot;H&quot;. Any argument acceptable to &quot;S&quot; as a field number is acceptable to &quot;H&quot;.</td>
</tr>
<tr>
<td>fR</td>
<td>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 &quot;H&quot; 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 &quot;?&quot;, and ID's symbol table is not changed. This feature is intended to be used with &quot;U&quot; to unsave a program and its associated symbols for further reference. Note that the &quot;R&quot; process is different from &quot;T&quot; in that in case of &quot;R&quot;, current symbols are first killed, whereas in the case of &quot;T&quot; new symbols read are merged with current ones. Any argument previously used by &quot;H&quot; as a field number can be used for &quot;R&quot;.</td>
</tr>
</tbody>
</table>
Two other commands to hoard and read symbols are available for swapping the symbols to and from a specified location. These are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x&lt;fh</td>
<td>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 &quot;S&quot; 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.</td>
</tr>
<tr>
<td>x&lt;fr</td>
<td>Reads symbols from field f below location x previously stored by x&lt;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 &quot;?&quot;, and ID's symbol table is not changed. Any arguments previously used in the &quot;x&lt;fh&quot; command can be used for &quot;x&lt;fr&quot;.</td>
</tr>
</tbody>
</table>

* The capital letters H and R when not preceded by a character mean hoctal and relative. *(See section 1-5 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:

<table>
<thead>
<tr>
<th>Terminating Character</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Punches the standard input routine and sets ID to punch the usual checksummed data blocks.</td>
</tr>
<tr>
<td>✓</td>
<td>Sets ID to punch the usual checksummed data blocks, but no input routine.</td>
</tr>
<tr>
<td>✓</td>
<td>Sets ID to punch read-in mode tapes.</td>
</tr>
</tbody>
</table>

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. f<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. \texttt{aJ}, where \texttt{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 (\texttt{*}) 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 (x). For example:

   loc/ ->| add a ->| abx x| add abc
            / | 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 octal (18)
   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 octal
   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 unoctal (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)
V 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)
x delete typed input (39)
/ examine 12-bit address register (10)
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tab</strong></td>
<td>modify and open addressed register; also alters sequence of location (9)</td>
</tr>
<tr>
<td><strong>bk sp</strong></td>
<td>modify and open next register (8)</td>
</tr>
<tr>
<td><strong>car ret</strong></td>
<td>modify and close register (8)</td>
</tr>
<tr>
<td><strong>uc, lc</strong></td>
<td>set case</td>
</tr>
<tr>
<td>**(</td>
<td>**</td>
</tr>
<tr>
<td><strong>(&gt;)</strong></td>
<td>modify and open addressed register (10)</td>
</tr>
<tr>
<td><strong>(\cdot)</strong></td>
<td>take preceding constant as decimal integer (18)</td>
</tr>
<tr>
<td><strong>• (center dot)</strong></td>
<td>punch opened register as one word block (37)</td>
</tr>
</tbody>
</table>
# APPENDIX II

## ID SYMBOL TABLE

<table>
<thead>
<tr>
<th>BASIC INSTRUCTIONS</th>
<th>SKIP GROUP</th>
<th>MISCELLANEOUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>add 400000</td>
<td>→clo 651600</td>
<td>→clo 651600</td>
</tr>
<tr>
<td>and 020000</td>
<td>skp 640000</td>
<td>i 010000</td>
</tr>
<tr>
<td>cal 160000</td>
<td>sma 640400</td>
<td>is 1</td>
</tr>
<tr>
<td>dac 240000</td>
<td>sni 644000</td>
<td>2s 3</td>
</tr>
<tr>
<td>dap 260000</td>
<td>spa 640200</td>
<td>3s 7</td>
</tr>
<tr>
<td>dio 320000</td>
<td>spi 642000</td>
<td>4s 17</td>
</tr>
<tr>
<td>dip 300000</td>
<td>→spq 650500</td>
<td>5s 37</td>
</tr>
<tr>
<td>dis 560000</td>
<td>sza 640100</td>
<td>6s 77</td>
</tr>
<tr>
<td>→div 560000</td>
<td>szf 640000</td>
<td>7s 177</td>
</tr>
<tr>
<td>dzm 340000</td>
<td>→czm 640500</td>
<td>8s 377</td>
</tr>
<tr>
<td>idx 440000</td>
<td>szp 641000</td>
<td>9s 777</td>
</tr>
<tr>
<td>lor 040000</td>
<td>sza 640000</td>
<td></td>
</tr>
<tr>
<td>lot 720000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isp 450000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jda 170000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jmp 600000</td>
<td>cbs 720056</td>
<td></td>
</tr>
<tr>
<td>jsp 620000</td>
<td>cks 720033</td>
<td></td>
</tr>
<tr>
<td>lac 200000</td>
<td>→dba 720061</td>
<td></td>
</tr>
<tr>
<td>lsw 700000</td>
<td>→dc 720052</td>
<td></td>
</tr>
<tr>
<td>lio 220000</td>
<td>→dia 720060</td>
<td></td>
</tr>
<tr>
<td>→mul 540000</td>
<td>dpy 730007</td>
<td></td>
</tr>
<tr>
<td>mus 540000</td>
<td>→dra 720063</td>
<td></td>
</tr>
<tr>
<td>opr 750000</td>
<td>cem 724074</td>
<td></td>
</tr>
<tr>
<td>sad 500000</td>
<td>esm 720055</td>
<td></td>
</tr>
<tr>
<td>sas 520000</td>
<td>ich 730000</td>
<td></td>
</tr>
<tr>
<td>→sft 660000</td>
<td>lot 720000</td>
<td></td>
</tr>
<tr>
<td>skp 640000</td>
<td>lem 720074</td>
<td></td>
</tr>
<tr>
<td>sub 420000</td>
<td>lem 720054</td>
<td></td>
</tr>
<tr>
<td>xct 100000</td>
<td>ppa 730005</td>
<td></td>
</tr>
<tr>
<td>xor 060000</td>
<td>ppb 730006</td>
<td></td>
</tr>
<tr>
<td>cla 760200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clc 761200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clif 760000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cli 764000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cma 761000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hlt 760400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>→lal 760040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lap 760300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lat 762200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>→lia 760020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop 760000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opr 760000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stf 760010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>→swp 760050</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xx 760400</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## IN-OUT TRANSFER GROUP

<table>
<thead>
<tr>
<th>OPERATE GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>cla 760200</td>
</tr>
<tr>
<td>clc 761200</td>
</tr>
<tr>
<td>clif 760000</td>
</tr>
<tr>
<td>cli 764000</td>
</tr>
<tr>
<td>cma 761000</td>
</tr>
<tr>
<td>hlt 760400</td>
</tr>
<tr>
<td>→lal 760040</td>
</tr>
<tr>
<td>lap 760300</td>
</tr>
<tr>
<td>lat 762200</td>
</tr>
<tr>
<td>→lia 760020</td>
</tr>
</tbody>
</table>

## SHIFT/ROTATE GROUP

<table>
<thead>
<tr>
<th>SHIFTS/ROTATE GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ral 661000</td>
</tr>
<tr>
<td>rar 671000</td>
</tr>
<tr>
<td>rcl 663000</td>
</tr>
<tr>
<td>rcr 673000</td>
</tr>
<tr>
<td>ril 662000</td>
</tr>
<tr>
<td>rir 672000</td>
</tr>
<tr>
<td>sal 665000</td>
</tr>
<tr>
<td>sar 675000</td>
</tr>
<tr>
<td>scl 667000</td>
</tr>
<tr>
<td>scr 677000</td>
</tr>
<tr>
<td>→sft 660000</td>
</tr>
<tr>
<td>sil 666000</td>
</tr>
<tr>
<td>air 676000</td>
</tr>
</tbody>
</table>

## TIME SHARING INSTRUCTIONS

<table>
<thead>
<tr>
<th>TIME SHARING INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>aps 723077</td>
</tr>
<tr>
<td>sdi 723477</td>
</tr>
<tr>
<td>lsb 720052</td>
</tr>
<tr>
<td>wat 722477</td>
</tr>
<tr>
<td>arq 722277</td>
</tr>
<tr>
<td>bpt 722177</td>
</tr>
<tr>
<td>dsm 722377</td>
</tr>
<tr>
<td>ckn 720027</td>
</tr>
<tr>
<td>rbt 720217</td>
</tr>
<tr>
<td>cac 720053</td>
</tr>
<tr>
<td>asc 720051</td>
</tr>
<tr>
<td>dsc 720050</td>
</tr>
<tr>
<td>mmn 725377</td>
</tr>
<tr>
<td>nf 725477</td>
</tr>
<tr>
<td>sbr 722577</td>
</tr>
<tr>
<td>lea 724677</td>
</tr>
<tr>
<td>lei 724577</td>
</tr>
<tr>
<td>rer 724777</td>
</tr>
</tbody>
</table>
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 >=
//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
         terminate
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

    746100  0                   /A
    747100  0                   /I
    746600  0                   /F
    744400  0                   /M
    744454  007201              /M+1
    744454  007202              /M+2
    746200  0                   /B
bprint

l=0
r=1
.start
ID part 1 1/26/65.

/* this is the beginning of the real program */

ent, lcs $d /were we in ID? /
sza
  jmp en1 /*yes*/
idx ids /*we are now*/
jsp ssw /*set state word pointers*/
eem
  lcs 1 acp
dac ac
idx acp
  lcs 1 acp
dac io
idx acp
  lcs 1 acp
dac pc
idx acp
  lcs 1 acp
dac fg

/*now get the breakpoint package*/

  law bp
  dap . 2
  lcs 1 bpp
  dac bp
idx bpp
idx .-2
sas (dac bp nbp
  jmp .-5

/*get kdm pointer*/

  law xlc
dap xmw
  lcs i xmw
add xc0 /*kdf-aw1 = 4*/
dap xmw
  lcs 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 bktf /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 cl1   /call button
jmp 1bp  /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
dap i 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
xmmw, 050000 /gets into exec core.
xc0, 000004 /o4=kfd-aw1 (executive symbols)
xc1, 000017 /i5=sp2-aw1 (executive symbols)
xc2, 000015 /i3=brk-sp2 "
xc3, 000003 /brk-why "
why, 000000 /reason ddt was activated
bp, 000000 /break-
  000000 /point
  000000 /pack-
  000000 /age.
xe2, 000000 /temp storage for pc
nms, 000000 /switch for G P or X
id5, 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
plt, 000000 /storage for proceed routine
mpc, xad bk1 nbp /random number for bpl-bp0
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 xlc
dap xmw
eem
  lac i xmw
  add xc1 /sp2-aw1 = i5
dap xmw
  add xc2 /brk-sp2 = i3
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 . /temporary storage
cl,  clc  /clear the proceed indicator, dsm entry
dac bk0
jmp xe 3

c11,  law 2  /call button entry
sas nms
jmp c11
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

c11,  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?
x,  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 lls
1bp,  
dzm bkf /breakpoint
law 2
sad nms
jmp ixex /bpt while X in progress.
law 7777
and bp 3 /true loc of bpt
dac t
law blk1 /setup
dap 2bp

2bp,  
lac . /check for assigned breakpoint
sad t
jmp 3bp
idx 2bp
sas (lac blk1 nbp
jmp 2bp

ill,  
law 2 /illegal instruction
sad nms /were we X ing
jmp ixex /yes-illegal execute
lio (741010
law 7777 /check
and pc
dap t2 /for
law blk1
dap . 1 /illegal
lac blk1
sad t2 /instruction
jmp 3bp 1
idx .-3 /under
sas (lac blk1 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 taz
/G go
bg0,
    jda chk /new mode switch
    dzm nms
    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 /fetch from drum
    dzm tsf
    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 bkd nbp
jmp p2
lac bkf /did we stop by xct ing a bpt?
sza i
jmp p1-2 /no
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 bp1
jsp 1cc
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
    laaaaaaaaacccc  blk1  lac blk1
dac bp
lac blk1 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 stuff

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  /pass the buck to exec.
bp1, lio (-0)
jmp +2
/breakpoints insert into user's field
bpo, cli
/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
/break points take out
bp3, lac .
/spa
jmp bp5+1
dap tas
bp4, lac .
dac dep
cla
jmp dp0
dac 1 bp4
bp5, idx bp3
sas mpc
jmp bp2
cla
sad dff
bp6, jmp .
law (bpt
bp6+2, dap bp4
/dzm dff
idx dpx
lac .
dac bp2+1
jmp bp1
/bp as a constant above
/clear all breakpoints
cbp, dap cbx
law bk1
dap bp3
clc
dac 1 bp3
idx bp3
sas mpc
jmp .-4
cbx, jmp .
fet,   lio (-0)
   dio dff
   jmp dep+2

dep,   .
   dzm dff
   dap dpx
   lac tas
   sma
   jmp dpx-1

dep+6, lac tsf
   sza
   jmp dp2

dp0,   lio dff
   spi i
   lac dfp
   dip tl

dp1,   law 7777
   and tas
   jda chk
   dap tl
   lio tl
   dia
   law 1
   lio dff
   spi
   add dfp
   swp
   law dep
   dcc
   jmp dre
   lac dep

dp2,   jmp .

chk,   0
   dap chx
   law 7777
   and chk
   sas chk
   jmp ta0
   sub mb
   sma
   jmp ta0
   add mb

chx,   jmp .
dp2,   lio dff
law 7777   /fetch, deposit internal registers
and tas
sub (ac
spa
jmp ta0
sub (4
sma
jmp dp4

dp3,   lac i tas
spi 1
lac dep
dac i tas
jmp i dp
sub (2
sma
jmp dp6
spi
jmp dp3
add one
sma
jmp dp5
lac dep
jda chk
jmp dp3

dp4,   /M+1
lac dep
jda chk
jmp dp5

dp5,   /M+2
lac dep
jda chk
sub 11
spa
jmp ta0
jmp dp3

dp6,   /B thru B+nbp-1
sub (nbp
sma
jmp ta0
spi
jmp dp9
lac dep
sad (-0
jmp dp3
jda chk
dac chk
law bk1
dap dp7
dap dp7

dp7,   /check whether already assigned
lac .
sad chk
jmp dp8
idx dp7
sas mpc
jmp dp7
jmp dp3
dap dp7

dp8,   clc
dac i dp7
jmp dp7+3
dp9,  lacs
sma
Jmp 1 dpx
law 56
jda tys
clc
dac lw
Jmp pn2

pv,  dap pvx
clc
dac df
lac pvf
sz  
Jmp pv1
spa
Jmp pv2
law 7777
and fa
sub lo
spa
Jmp pv1
dap .+5
sub wc
add one
szm
Jmp pv1.

h1,  lac .

pvx,  Jmp .

pv1,  lac fa
dac pvf
dap lo
jsp zd
lac 0
Jmp pvx

pv2,  lac dep
Jmp pvx

punch, verify swap routines
/from pwd
/used by searches
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
dcc
zda,    jmp dre
zd,    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 sul
uns,      jda ckf    /unsave
dzm t
dip t
lio t
lac dfp
dac t1
sul,      dzm lo
    law i 1
    add mb
dac wrd
c1c
dac dfp
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
c1i+swp-oopr
rcr 6s
add c4
jmp su1-5
cfk,      c
    dap cfx
    lac ckf
    and (37
    sub (ndf
    szm
    jmp err    /field number too big
    lac ckf
    rar 6s
cfx,      jmp .
ws, spi
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
/W,N,E
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
/we used as and
wet,  xx
/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
vfl,
lac t
dap fa
lac fa
sub chk
sub (dio
spa
jmp vf2
add chk
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
law 51
arq
jmp bus
jsp lct
law 4642
jda tys
jsp sol
jsp lct
arx,
jmp .
bus,
jsp lct
lac (356224
jda tys
lac (223034
jda tys
jmp pn2
start
lis,  law i 47  /initialize assignments.
arq
lis+2, law i 51
arq
zrt,  lac dpf
      dac dpf  /zero drum routine return
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 skd
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
     add (jmp uc
     dap lsx
     sub ar1  /used as add
     spa
      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
      /last no-eval routine
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
sgn, xx dac wrd
     lio chl
     spi
     lac lwt
lsx, jmp .

n, rir 5s
    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 11
ln, add (44-12
    spa
    jmp n

11, dzm let
    dzm chl
    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
Jmp .+2
/lower case
lc, lio ps1
dio cas
Jmp lsr+1

sqo, lac dnm
Jmp n+1
'/ means take decimal number
quo, lac sym l
sza i
lac sym r
jda spv
lac tys
Jmp n1
'/" means take as flexo codes

q, lac lwt
Jmp n1
/Q means last quantity
f, lio chi
spi i
Jmp ff
law fg
Jmp n1-1

a, law ac
Jmp n1-1
/A means accumulator
ir, law io
Jmp n1-1
/I means i-o
m, law msk
/M means mask register
dac iif

n1, dzm chi
dac syl
Jmp n2

ta0, err, lac (743521
Jda tys
/lc, blk
erl, law 7234
Jda tys
law 1 51
arq
Jmp lse

ser, law 7435
Jda tys
/law 0772
Jda tys
lac est
dap srl
/symbol table overflow error
sr1,  lac
spa
idx srl
idx srl
lac i srl
jda pi
lac (741034
jda tys
jmp lis+2

daq,  law 7777
and lwt
jmp .+2

com,  lac loc
dac df1

/defines sym as address of Q

/comma defines sym as loc
.../define symbol

df2,  dio i es4

dex,  jmp .
dot,  lio chc
    lac loc
    spi i
    lac dnm
    dac syl
    law 44
    dac t2
    jmp 11

del,  dzm iif
    jmp pn2

/symbol table full
/end of no-eval routines, delete
val, dac df
j mp l s s

/ open paren, sets up value for define

kil, spi
j mp k i 5
lac le t
s za
j mp e r r
l acl o w
sad i e st
j mp e r r
l aw k i l
dap e v x
l aw l a l
j mp e v l + 2

/K

/ used below

/delete one symbol
eql, jda eap
    jda opt
    jmp lss

pn2, jsp lct
    jmp lss

arw, jda eap
    jda pi

ar1, jmp del

oct, spi i
    jmp err
    law 10
    jmp .+4

dec, spi i
    jmp uns
    law 12
    dap ops
    jmp lse

smb, spi i
    jmp sav
    law pi
    jmp .+2

cns, law opt
    dap pns
    jmp lse

oard, law pvl
    jmp tls-1

rad, spi i
    jmp err
    law pev
    dap pad

tls, jmp lse

pls, lac cad
    jmp ssn

4in, spi
    dio wrd
    lac csu
    jmp ssn

uni, jmp ssn-1

isc, lac can
    jmp ssn

/plus, space

/minus

/union, V

/intersection, ∧
bac, law opt
    jmp .+2
bas, law pi
    dap bax
    jmp bar
vb, clc
dac mod
bar, lac lcf
dac tas
    lac iiif
dzm iiif
dac lcf
    sza
dac tsf
    lac wrd
spi 1
    jmp ta5
    lac tas
dac lcf
    lac lwt
dzm tsf
    jmp ta6
uc8, spi 1
    jda dep
dzm tsf
    jmp ta6

/c means make corr. and open register

/vertical bar
\slash .
/tas used for temporary storage

/carriage return

eas, law ws4.+2
    dap dpx
law ea1
    jmp ws

/word search

/not word search

/pbx, jda eap
    jda tys
    jmp pn2

/effective address search
rd,          spi i                /read binary tape (Y)
            jmp err
            jsp ar
            jmp .+2

nb,          jsp rbk
            law 1 1
            add ch
            and (7777
            jda chk
            jac 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 ch1
            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 1
         jmp err
         law 47
         arq
         jmp bus
         feed 30
         jsp lcc
         jmp tpi

/jump block

jbk,       spi
         jmp err
         law 47
         arq
         jmp err
         lac wrd
         lor cj
         dac lwt
         feed 40
         lio lwt
         jsp pbw
         feed 520
         jmp lls

/title punch and punch format setup
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 l
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 l
tbl,    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  rbl
      jmp  tbm  /skips rest of tape

gwd,  dap  gwx
      lac  la
      sas  rbl
      jmp  gw1
      jsp  rbl
      jmp  gw+1

gw1,  dap  gwa
      idx  la

gwa,  lac  .
gw2,  jmp  .

sb5,  jsp  gw
      dac  sym  l
      jsp  gw
      and  (177777
      dac  sym  r
      lor  sym  l
      sza  i
      jmp  tb5
      jsp  gw
      dac  df1
      lac  sym  l
      lia
      ril  1s
      sma+sp1-skp
      jmp  sb5
      and  (177777
      dac  sym  l
      jsp  de
      jmp  sb5
tb2,  
  
dzm sym l
sas one
jmp pot
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 5151-low
add low-1
dac t3
law low-2

tb3,  
dap 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

/table read from ts macro or possible
/see if table read from possible
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)
cma&lia
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 1 gsb
sni 1
jda df1
idx gsb
idx gsb
idx t2
isp t2
jmp gsb
jmp gfd
pz, t6, 0
  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 1
  jmp pz1
pzx,
  jmp .

k11,
  jmp .+2
  jmp err
  sub ev2
  dac t3
  idx t3
  dac t4
  idx t4
  idx es4
  and (7777
k12,
  sub t4
  dap k13
  add t3
  dap k14
k13,
  lio .
k14,
  dio .
  sas est
  jmp k12
  add t3
  jmp .+2
k15,
  law low
  dac est
  jmp lse
fee, t2, 0
  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
px, jmp .
eap, 0 /eql,arw,pbx common
dap .+/7
lac eap
dac lwt
lac iff
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
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 166
sad pro+1
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 .

166, law 1 /1s-9s
add pi
and pi
sza
jmp pvl
law pax+1
dap pex
lac ea
jmp eax+2

pad, 0 /print address
dap px
law 7777
and pad
dac pi
clf 1

pa1, jsp pev /pev or pv1
lac (flexo +
jda tys
jmp pvl
ta4, jda pad
llo 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 /p1, opr or lwt
jmp pn2
pev,  
dap pex  
law i 7777  
and pi  
sad (opr  
jmp sev  
and (760000  
sad (skp  
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  
jmp sko  
xor pi  
spa  
jmp eix  
lac pi  
sub i ea  
spa  
jmp eix  
szf 1 1  
jmp psw  
lac i ea  
sub i eil  
szm  
jmp psw  
eix,  
idx ea  
lia  
sub mst+2  
swp  
spi 1  
jmp +3  
sas mst+1  
jmp eal  
szf 2  
jmp pex  
szf 1 1  
jmp pvi  
lac pi  
eil,  
sub  
lia  
sza  
jmp 177  
dio pi  
eiy,  
jsp spt  
lac pi  
sk2,  
sza i  
jmp px  
szf 1 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 i
  xor t1
  sza i
  jmp eix
  xor pi
  lia
  and t2
  sza
  jmp eix
dio pi
  szf i l
  jmp psw
  lac (flexo V
  jda tys

psw:
  lac .  /best symbol thus far
dac sym r
  lio to
  dio sym l
  lac ea
dap eii
  stf i
  szf 2
  jmp eiy
  jmp eix

177:
  law i 7777  /numbers 77xxxx
  and pi
  sas (770000
  jmp eiy-1
  law 7254
  jda tys
  lac tys
  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
tyb,  count opt, tyl
lac lwt
cli

Tyx,  jmp .
dns, lac ps1
   lio (72
   sad cas
   jmp tyc
dac cas
   jmp tyb

dnl,

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

lcc, dap lcx
   law 7277
   jmp lc1

lct, dap lcx
   law 7236

lc1, jda tys

lcx, jmp 

so1, rpb
   rpb
   spi i
   jmp so1
dio t4

soi, rpb

rbk, dap rbx
   init rb1, buf
dap 1a
dzm chi
   rpb
dio t2
dio t
   spi
   jmp lis+2
   rpb
dio ch

   law 1 1
   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 

/redundant case shift filter

/lower case, carriage return

/lower case, tab

/skip over input routine

/enter here

/read a block into buffer

/start block read.

/check for block format
lac (356342)
jda tys
lac (224434)
jda tys
jsp lct
ty1
la1
sas (char rc
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,    read 5
pb2,    jmp .

plo,    jsp pev
jmp pa1+1

pbw,    dap pby  /punch binary word
repeat 3, ppb
adm t2
rcl 6s
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 1
  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,^n
letter 2, sqo /2,!
letter 3, pbx /3,^n
letter 4, daq /4,^C
letter 5, uni /5,V
letter 6, isc /6,V
letter 7, pul /7,<
letter 10, uc8 /8,>
letter 11, fs /9,^r
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 disp err, err
tletter 35, tbl /t disp err, err
tletter 36, dec /u disp err, err
tletter 37, vfy /v disp err, err
tletter 40, wds /w disp err, err
tletter 41, xec /x disp tab, tab /tab
letter 42, rd /y disp err, err
tletter 43, zro /z disp err, err
tdisp com, eql /,,= disp err, err
disp pwd, err
letter 23, jbk
letter 24, kil
letter 25, ttl
letter 26, m
letter 27, nws
letter 30, oad
letter 11, pra
letter 32, q
letter 33, rad
disp err, err
disp err, err
disp min, pls
disp def, bas
disp ovb, vb
disp val, bac
disp err, err
letter 12, a
letter 13, bk
letter 14, cns
letter 15, pun
letter 16, eas
letter 17, f
letter 20, bgn
letter 21, oct
letter 22, ir
disp lc, lc
disp dot, del
disp uc, uc
disp bs, bs
disp err, err
disp cr, cr

start
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrd, 0</td>
<td>quantity being assembled</td>
</tr>
<tr>
<td>sym, 0</td>
<td>alpha symbol being assembled</td>
</tr>
<tr>
<td>chc, 0</td>
<td>character count</td>
</tr>
<tr>
<td>chi, 0</td>
<td>+0 when letter or number has been typed since last typeout or c.r. input</td>
</tr>
<tr>
<td>let, 0</td>
<td>+0 when letter in syllable, otherwise -0</td>
</tr>
<tr>
<td>ch, 0</td>
<td>character</td>
</tr>
<tr>
<td>syl, 0</td>
<td>syllable</td>
</tr>
<tr>
<td>t, 0</td>
<td>temporary storage</td>
</tr>
<tr>
<td>la, dio</td>
<td>last address</td>
</tr>
<tr>
<td>fa, dio</td>
<td>first address</td>
</tr>
<tr>
<td>mod, 0</td>
<td>mode, -0 for &quot;type-in&quot;</td>
</tr>
<tr>
<td>opd, dnm, 0</td>
<td>decimal number</td>
</tr>
</tbody>
</table>

**Constants**

- \( c_4 \), 400000
- \( c_{77} \), 77
- \( c_{01} \), 010100
- \( c_j \), 600000
- \( c_3 \), 3
- \( c_2 \), 020000

- \( p_{vf} \), .
  - punch, verify flop
  - 0 - usually, set by zro
  - -0 - center dot
  - not +0 - continue in pv subroutine

- \( b_{kf} \), 0
  - breakpoint flop
- \( e_{sp} \), dzm .
  - used by zero routine

- \( d_{fl} \), 0
  - value for defining symbol

- \( l_{oc} \), 0
  - current location
- \( t_{as} \), 0
  - address part for fetch or deposit
  - current register. instruction part
  - + for register closed, tells dep and fet subroutine to ignore

- \( t_{sf} \), 0
  - current examination flop
  - 0 - external register
  - non-0 - internal register

- \( l_{cf} \), 0
  - current location flop
  - 0 - external
  - non-0 - internal

- \( i_{lf} \), 0
  - initial internal flop
  - 0 - usually
  - non-0 - set when type A,I,M,B

- \( b_{bf} \), 0
  - B flop
  - 0 - usually
  - -0 - when B typed, not affected by uc,lc

- \( c_{n3} \), 400000
  - special proceed counter

- \( d_{ff} \), 0
  - deposit-fetch flop, 0 dep, -0 fet

- \( l_{o} \), .

- \( w_{c} \), .
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
bkd, -0
repeat nbp-1, -0

+nbp/
bko, -0 /switch for legal proceed

amm, 30 /if legal, has proceed address
ti, 0
c,

constants

start lis