STANFORD TIME-SHARING PROJECT
Memo No. 23

October 2, 1964

USER'S MANUAL TO THE ODIN TIME SHARING SYSTEM

by Gary Feldman and Harold Gilman

Abstract: The following is a description of the operating procedures of ODIN, the Preliminary Time Sharing System for the PDP-1.

The research reported here was supported in part by the National Science Foundation (GP-3207).

TABLE OF CONTENTS

USER'S MANUAL TO THE ODIN TIME SHARING SYSTEM

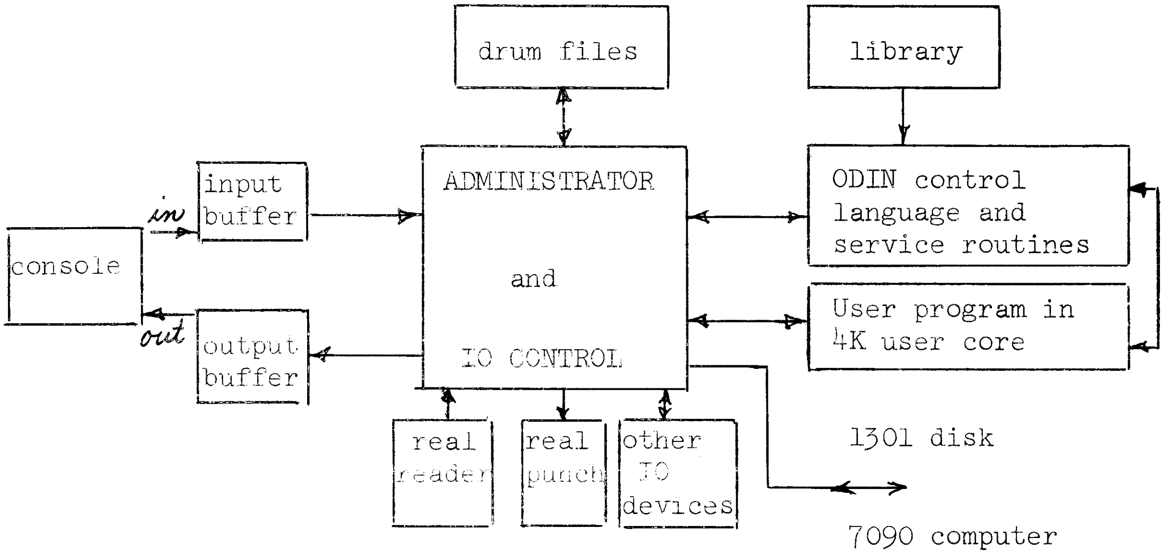
by Gary Feldman and Harold Gilman

Introduction:

The ODIN time sharing system provides each of five users simultaneous computer service. At present there are three local teletype consoles, a typewriter console, and a connection into the TWX' network. Each user has a console, access to the real paper tape reader and punch, and a "computer" similar to the PDP-1 with 4096 words of core memory. The difference between the user's "computer" and the PDP-1 lies mainly in changed and expanded input-output services (such as limited communications with the 1301 disk file and the 7090 computer), the availability of certain service and library routines, and a system command language for controlling the operation of programs and using and maintaining paper tape files stored on the PDP-1 drum. It is this last difference which is fundamental. Since many users will want to perform paper tape operations simultaneously, the paper tape facility must be simulated with a system of private files. Once one has learned how to use the file system, programming under the ODIN time sharing system is essentially the same as programming for the bare PDP-1. With certain restrictions outlined below any program written for the 4K PDP-1 will run as a time shared program under ODIN.

Organization of ODIN:

As far as a user is concerned, ODIN appears to have the following configuration:



ODIN is divided into two major sections. One is the administrator and IO control and the other is the service and file control program. The administrator's function is to parcel out time fairly to each of the users and to sort out properly the stream of input-output generated in various ways. Each user has a 4k core image called the user program stored on a drum field. When the user is activated his user program is brought into core and started where it last left off. The administrator cycles through the active users in a strict round robin giving each user approximately 70 milliseconds of run time before dismissal. For each swap (i.e. writing out the old user program and reading in the new) the system takes 30 milliseconds. The total 100 milliseconds is called a quantum. A user is active provided:

a. He is running a user program or a service program.
b. He has not filled his console output buffer.
c. His program is not waiting for input.

This means that when any program would be waiting for input or for output completion it is immediately dismissed and the next user in the round robin is given service. When there is only one active user no swaps need take place so that user runs at full efficiency.

In addition the administrator routes all input-output, translates teletype code to DEC concise code and vice-versa, and translates some special iot (input or output) commands into system actions.

The file control programs store a directory of drum files which simulate PDP-1 paper tape and provide a language for file manipulation. In addition there is an octal debugging service and cliche decoder which interprets lists of system control language commands.

Programming Under ODIN:

Programming for the PDP-1 is normally done in an assembly language called Macro. This assembler produces standard PDP-1 machine code. All commands listed in Appendix II may be user in programs under ODIN with these exceptions, restrictions,and additions:

1. Do not use program flag one for anything except a type-in listen loop. (A listen loop is code of the form

```
szf i 1
jmp .-1
tyi
```

which makes the computer wait until the depressing of a typewriter key changes flag one to on status.) The administrator automatically feeds flag one on to all programs at the beginning of their quantum and after each console input. This is done so that any program with a listen loop will not wait for input but will immediately execute its "tyi". The "tyi" instruction is seen by the IO control program which checks for characters appearing in the input buffer. If there are none, the user program is instantly dismissed; otherwise, the next character is fed to the user program. Since this is the case, if the user wishes to save a little time he can replace all his listen loops by bare "tyi's". This has the disadvantage, however, of rendering his programs incompatible with the PDP-1 when time-sharing is not running. Because flag one is always on, it is impossible to use it for any kind of logic. In order to allow old programs which have logic tests of flag one, the user may turn off automatic flag one mode by executing the suitable control command.

2. If the user desires that his program not be dismissed while waiting for console input, he may use the special system command iot 117. This command is interpreted as "skip on input buffer empty". (Iot i 117 means "skip on input buffer non-empty). A use for this might be to keep a scope display runnign while listening for input characters. The code would be.

```
begin,      iot 117
            jmp listen
               code
               to
               run
               the
               display
            jmp begin
listen,     tyi
               code
               to
               handle
               the
               character
            jmp begin
```

3. Do not use any of the sense switches. The console may be assumed to be off, and all sense switches will uniformly have the value zero. Old programs which use the switches should be rewritten. All library routines have already been modified, and descriptions of how to use them under ODIN will be found below.

4. Do not use the instruction "lat"; the test word will always have the value zero.

5. Do not attempt to enter extend mode (i.e. execute the command eem) or leave extend mode (lem). These commands will result in the error message "ilg iot". Do not attempt to use any of the sequence break commands: esm, lsm, asc, dsc, isb, cac, or cbs. These will be treated by IO control as no-operation (nop) commands. Do not attempt to read or write the PDP-1 drum except for fields 34-37 which are free (and unprotected) scratch area. All core addresses inside of drum commands will be changed to refer to user memory; and all drum commands which refer to fields other than 34-37 will be treated as illegal iot's, causing the message "ilg iot" to be printed.

6. Quantum synchronization. There are some operations, e.g. a data transfer to the 1301 disk which will fail if interrupted before completion. To guarantee that an operation gets a full quantum of useful runtime (70 milliseconds), place the special instruction iot 17 immediately before the first instruction of the operation. Iot 17 means that the program will be dismissed immediately and the next instruction will be taken at the beginning of the quantum the next time around the round robin.

Console code translation:

The internal code of all programs requiring console input is concise code. Input coming from a teletype console is translated to concise before being fed to any programs. Thus the command "tyi" serves to read any console. Similarly output going to a teletype is translated from concise to teletype code; so a "tyo" serves to write on any console. Because the character set of the teletype is not identical to that of the typewriter, certain characters have to be transliterated from ASCII to concise. The list of transliterations may be found in Appendix III.

Files:

Paper tape allows fast input-output to the bare PDP-1. Since there is only one paper tape reader and one punch (hereafter called real reader and punch respectively) it is necessary to simulate paper tape by means of paper-tape images stored on the PDP-1 drum. Each console has a section of drum assigned to it for storing paper tape images. Three of the consoles have "long files" consisting of five drum fields and two (consoles No. 2 and No. 3) have "short files" containing two drum fields.

A file is simply a paper tape image on the drum. Associated with it is a name supplied by the user and an octal drum address supplied automatically by the file control system or manually by the user. This address marks the beginning of the paper tape image. At the end of each file the system automatically supplies an "end of file" mark. The ODIN control language allows the user to name files, kill files, move paper tape from the real reader to files, move files to the real punch, move from one file to another, read from files with a program that reads paper tape, and punch onto files with a program that punches paper tape.

The user makes use of various files by manipulating the location of the input pointer and the output pointer. This is done by the ODIN control language. The input pointer tells the user or system service program from which file it should read its paper tape input and the output pointer tells onto which file the paper tape output should be punched. It is possible to set the pointers directly to the beginning of files by referring to the files by name, or to the middle of files by manually setting the appropriate octal drum addresses.

The drum is formatted into a continuous string of forty (octal) word blocks. Two frames of paper tape are stored in each word. (Alphabetic information is stored at two characters per word and binary information at two-thirds of a binary word per word. Thus one could expect to store one core load of binary information in 1-1/2 drum fields.) When forty words have been read or written a drum transfer takes place. If a file is ended in the middle of a forty word block the end of file is placed at the end of the block. This means that all drum addresses referred to should be multiples of 40_8.

The ODIN Control Language:

In the description of the ODIN control language the character 'carriage return' will be denoted by '↲', the character 'backspace' will be denoted by 'ω', and the character 'center dot' by ':'. The control language is an interpreter for a specialized set of single character contol statements. Like the debugging program DDT each legal control character specifies a different service routine to perform some action or change the state of the universe in some way. Unlike DDT many of the control characters can be followed by parameters to give additional information about the control action. It is also possible to list several control functions inside of a <u>cliche</u> and have the executed interpretively as a control "program".

In order to talk to ODIN in its control language, it is necessary to call the system. Depending on the type of console this is done in different ways. From the typewriter use ":↲" or ":ω". If the 'center dot' is followed by the 'carriage return', it means that the system will be ready to listen to the control language from the console; if the 'center dot' is followed by the 'backspace', control is passed directly to the next instruction of the current control cliche. (For illustrations of how to use ":ω" effectively see the sections of examples). To indicate that it is ready to accept control information, the control system types "ODIN".

Because the typewriter will not accept input while it is typing out, it may be necessary to call the system while caught in a type out loop. An emergency call can be made from the typewriter by depressing the ribbon switch and typing "↲'s" until "ODIN" is typed out. Sometimes the user may want to include the character 'center dot' within text and not have it call the system. This may be done by typing 'center dot' twice. The 'center dot' will be entered into text once for every two times it is typed.

From the teletypes the procedure is exactly the same except that "#" (sharp) is used in place of 'center dot'. (Note that "#" does not transliterate into 'center dot' but rather into "~". It is "\" (i.e. 'backward slash' made by shift "L") which means 'center dot' on the typewriter.) As in the case of the typewriter, enter "#" into text by typing it twice. Because the teletype will listen while it is typing out, there is no need for an analogue of the ribbon switch.

In the description of the control functions to follow we will use these conventions.

1. An asterisk (*) preceding any control function will mean that the function may <u>not</u> be included in a cliche.

2. Numerical parameters to control functions are always octal numbers and will be represented below by #1, #2, ...

3. Alphanumeric identifiers used as parameters may be 1-6 characters long and must begin with an alphabetic character. They will be represented below by ϵ1, ϵ2, ...

4. A command part is a system control command minus the 'carriage return'. When command parts appear as parameters to cliche definitions they will be represented by %1, %2, ...

5. Some parameters will be directly specified rather than by the above notations, as in the command "l,et↲". (which means load expensive typewriter). The types of parameters and the formats for the parameters are indicated implicitly by the symbols used. E.g. the command "n,ϵ1↲" indicates that the control function "n" takes an alphanumeric identifier for its only parameter.

The control functions command are field free in that spaces are ignored.

b,ϵ1,ϵ2↲

Causes the file named "ϵ1" to be renamed "ϵ2". Example: if one had a file named "eng", "b,eng,song↲" would rename the file "song".

*c,ϵ1,(%1)↲

Causes a cliche named "ϵ1" to be defined. When the cliche is executed it will perform the command "%1↲". Example: To define a cliche to rename the file as above use "c,rename,(b,eng,song)↲".

```
*c,ϵ1,(%1↲
%2↲
%3↲
 .
 .
 .
%n)
```

Causes a cliche named "ϵ1" to be defined. When the cliche is executed it will perform the commands "%1↲,%2↲, ... ,#n↲" in sequence. If any of the commands is an exit to a user program, the exit will occur and the cliche will be suspended. If the system is recalled by typing ":ω" ('center dot' 'backspace') the execution of the cliche will continue where it left off. (See examples below and c.f. the section ODIN Control Language above).

*d↲

Causes all cliches to be deleted.

*d,ϵ1,ϵ2, ...,ϵn↲

Causes the cliches named "ϵ1,ϵ2, ...,ϵn" to be deleted.

e,ε1↲

Causes the cliche named "ε1" to be executed. Example:"e,rename↲" will cause the cliche dfeined above to be executed.

f,1↲

Causes automatic-flag-one mode (see above) to be entered. ("f↲" will have this effect also).

f,0↲

Causes automatic-flag-one mode to be discontinued. This command is used only in very special cases.

h↲

Causes the system to type out a messate telling the location at which the user program was last interrupted and the contents of the accumulator and i-o register at the last interruption.

i↲

Causes the location of the reader pointer to be typed out.

i,ε1↲

Causes the input pointer to be attached at the beginning of the file named 'ε1". Example: to have a program read the file named "song", it would be necessary to type "i,song↲" sometime before entering the program that would do the reading.

i,#1↲

Causes the input pointer to be attached directly to the drum address "#1". The restrictions on "#1" are that it be smaller than the end address of the console's file area and that it be equal to zero modulo 40 (octal). To have a program read from file starting at 270440 type "i,270440↲".

k↲

Causes all the files to be killed and the output pointer to be moved to the beginning of the drum block of files. For example if one were at the typewriter "k↲" would kill all the files and move the output pointer to 230000.

k,ε1,ε2, ... ,εn↲

Causes the files named "ε1,ε2, ...,εn" to be killed. This command does not move the output pointer.

l,et↲

Causes the text editor called Expensive Typewriter to be loaded and begun as the user program. Instructions for using the modified version which lives in the ODIN library will be found below.

l,ddt↵

Causes the debugger called DDT (DEC Debugging Tape) to be loaded and begun as a user program. The version in the ODIN library is the non-extend mode version whose starting address is 6000_8.

l,macro↵

Causes the assembler called Macro to be loaded and begun as a user program. Instructions for using the modified version that lives in the ODIN library will be found below.

l,macsym↵

Causes the symbol package that mates with Macro to be loaded and begun as a user program. Instructions for use to be found below.

m,ε1,ε2↵

Causes the contents of the file named "ε1" to be moved to the file named "ε2". The paper tape image beginning at the drum address associated with "ε1" is copied on the drum beginning at the drum address associated with "ε2". When the move is completed the system will type out "end of file.".

m,rdr,ε1↵

Causes the paper tape which is in the real reader to be copied onto the drum files starting at the drum address associated with the file "ε1". This command waits for the reader to be turned on before commencing; however, it cannot detect the end of the paper tape. When the tape has run out of the real reader wait approximately 20 seconds and then call the system. This wait allows the read-in buffer to empty onto the drum; the time is a function of the number of active users.

m,ε1,pch↵

Causes the contents of the file named "ε1" to be copied onto the paper tape in the real punch. When the transfer is completed the system will type out "end of file.".

n↵

Causes the list of all the files named by the user to be typed out with their associated addresses.

n,ε1↵

Causes a file to be created named "ε1". The drum address associated with this file will be the current location of the output pointer. For example, if the output pointer were located at 270440 the command "n,easy↵" would define the file "easy" to begin at drum location 270440.

o↙

Causes the current location of the output pointer to be typed out.

o,ϵ1↙

Causes the output pointer to be attached to the beginning of the file named "ϵ1". Example: to have a program write out onto the file named "song", it would be necessary to type "o,song↙" sometime before entering the program that would do the punching.

o,#1↙

Causes the output pointer to be attached directly to the drum address "#1". The restrictions on "#1" are that it be smaller than the address of the console's file area, larger than the beginning address of the console's file area, and that it be equal to zero modulo 40_8. To have a program punch onto files starting at 236040 type "o,236040↙".

p↙

Causes all the user's cliche definitions to be punched out starting at the current location of the output pointer. The format is compatible with Expensive Typewriter.

r↙

Causes system to simulate the PDP-1's read-in mode. Reading commences at the current location of the input pointer. If one had a binary program stored in a file named "song", "i,song↙" "r↙" would serve to have it read in and begun as a user program.

t↙

Causes the system to transfer to the user program and begin running it at the location after the last executed instruction. The contents of the accumulator and i-o register as well as the state of all the program flags (except flag one) are restored to their state previous to interruption. For example if the user program was typing out and the system was called, the command "t↙" would cause the typing to continue exactly where it left off.

t,#1↙

Causes the system to transfer to the user program and begin running it at the octal location "#1". The parameter "#1" is always taken modulo 10000_8. For example, if DDT is the current user program the command "t,6000↙" would serve to restart it.

v↲

Causes the control system to take alphabetic information from the drum starting at the current location of the input pointer. These characters are interpreted as if they came from the console and are executed just as if they were control commands typed by the user. If, for example, the cliches were punched onto a file named "song" by the sequence "n,song↲" "o,song↲" "p↲", they could be read into the system to redefine the cliches by the sequence "d↲" "i,song↲" "v↲". As the control information is used by the interpreter, the characters are typed out onto the console as that the user may monitor them.

w↲

Causes the system to wait inactive until the appropriate character is typed. The system types out "to continue type →". When '→' is typed the system becomes active again.

y↲

Causes the same action as the control function "v↲", except that the type out of control information is supressed.

#1↲

Typing an octal number of four or fewer digits followed by a '↲' puts the ODIN control language into octal debugging mode. The contents of the register at location "#1" in user core is typed out and the register is opened similarly to DDT. At this point ODIN acts exactly like DDT in spirit, but, of course, with a different set of conventions. When a register is open the contents may be changed to any octal constant of six or fewer digits by typing that constant. Typing "#1↲" has opened the register #1. Whether one changes its contents or not one may close it and open the next register (#1+1) by typing 'ω' or 'a' or open the preceding register (#1-1) by typing 'u' or open the current contents of register #1 by typing the character 'tab'. One may close the currently open register and exit from the octal debugging mode by typing '↲'. To rectify a mistake in entering an octal constant type 'x' and then begin typing the constant again. As an example of the use of the octal debugging mode, here is a dialogue to change three locations.

(The information typed out by ODIN will be underlined)

```
1243
         600454          600100ω
1244     200234          200012'tab'
12       654033          600130↵
```

'ω'

This causes the current imcompletely typed control command to be forgotten. It is used when a mistake is made while typing in a control command. (While in octal debugging mode it does not have this effect, see above). Example: if one were trying to name a file "song", and typed "n,sin", typing a "ω" would make ODIN forget the command. Once a command is ended with a "↵" (or ")" in the case of cliches) then the "ω" has no effect.

Simple Examples of Some of the System Commands:

The following is a dialogue between a user sitting at a teletype console and the ODIN control system. Then conventions are that "#" is the character that calls the system and that all information that the system types out will be underlined.

```
#↲                             (To call the system)

ODIN

k↲                             (To kill all the files so as to begin fresh).

n↲                             (As confirmation that all files are killed)

name drum address              (The system lists no files)

n,george↲                      (The user names a file)

n↲                             (Checking.)

name  drum address

george 110000                  (The file has been named and its corresponding
                               drum address is 110000 which is located at the
                               beginning of the user's file area.)
                               (The user wishes to run a binary program called
                               spacewar.)

n,rdr,george↲                  (The paper tape containing spacewar is read
                               onto the file called "george". After the
                               tape runs out of the real reader the system
                               is called.)

#↲

ODIN

b,george,spcwar↲               (The user renames the file appropriately.
                               This is unnecessary, of course.)
n↲

name  drum address

spcwar 110000

n,file2↲                       (The user names a file to begin after the
                               spacewar file. "File2" will be placed at
                               the current location of the output pointer.)
```

```
n↲

name   drum address

spcwar  110000

file2  117540

i,spcwar↲                      (In order to run spacewar, the user attaches
                               the input pointer to the beginning of the
i↲                             file "spcwar", so that the read-in mode
reader=  spcwar 110000         simulator can read the image of the binary
                               program into user core and start it as
                               a user program.)

r↲                             (Spacewar is now running.  It will continue
                               to run indefinitely until the system is
#↲                             called.)

ODIN                           (The user is now talking to the system)

h↲                             (He asks where spacewar was interrupted when
                               the system was called.)

now at 44756 ac - 625751 io-622377

t↲                             (This continues spacewar where it left off.)

#↲                             (Calling the system again.)

ODIN

t,4↲                           (Spacewar begins at octal location 4, so
                               that the command will restart spacewar.)

#↲                             (Recalling the system.)

ODIN

m,spcwar,file2↲                (The user is moving the image in file "spcwar"
                               to the file "file2".  When the move is com-
end of file.                   pleted the system types out "end of file.".)

n,file3↲                       (Naming a third file to begin at the end of
                               "file2".)

n↲

name   drum address

spcwar  110000
```

```
file2  117540
file3  127300
k,file2↵                       (Killing "file2".)
n↵
name  drum address
spcwar  110000
file3  127300
k,file3,spcwar↵                (Killing the other two files.  Notice that
o↵                             the location of the output pointer remains
punch= file2  127300           unchanged, at the end of "file2" since
                               that was the last punching done.)

k↵                             (However, "k↵" does move the output pointer
o↵                             back to the beginning of the drum space.)
punch=  110000

o,127300↵                      (The user can move the output pointer
o↵                             directly to an octal drum address.)
punch= octal.  127300

i,134640↵                      (Also the input pointer.)

i↵

reader= octal.  134640

(Now for some exercises using cliches)

d↵                             (Deleting all the cliches.)

c↵                             (Checking that there are no defined cliches.)

name      text

c,useles,(w↵                   (For practice, the user names a useless
w↵                             cliche.  Notice that the last control
w)                             function in the cliche is not terminated
                               by a "↵", but rather by a ")".  When the
                               system sees the ")" it types back a
                               carriage return.)

c↵
name    text                   (This is the format that the system uses
useles  w                      to store cliches.)
        w
        w
        )
```

```
e,useles

to continue type →
→
to continue type →
→
to continue type →
→
```

(To execute this useless cliche. The cliche interprets the first control function which is "w↲". Then it waits until the user types an " →". At this point it interprets the second control function which is also a "w↲". Each control function in a cliche is interpreted in turn. When the last one is finished the system exits the cliche and is ready to accept new console input. During the execution of a cliche, if the user wants to interrupt and talk to the system, he may do so by calling the system with "#↲".)

```
c,silly,(h)↲

c↲
name    text
useles  w
        w
        w
        )
silly   h
        )
```

(Naming another useless cliche. Notice that cliches which contain only one control function take a special format in which the ")" must be followed by a "↲". This is because the interpreter looks at console input a line at a time, and, hence, will not process any command until it sees at least one '↲'.)

```
e,silly↲
```

(Executing silly.)

```
now at 17311
```

(Notice that the printing of the contents of the accumulator and the i-o register are suppressed inside of a cliche.)

```
d,silly↲
```

(Deleting silly.)

```
d↲
```

(Deleting everything.)

```
c,restrt,(t,4↲
h↲
e,restrt)
c↲
name    text
restrt  t,4
        h
        e,restrt
        )
```

(Recall that the user program is still spacewar. The user can restart it any time by typing "t,4↲". Instead he defines a cliche that will start spacewar, and when it is recalled by typing "#ω" it will type out the last location before interruption and then start spacewar over again. It is perfectly legal to have a cliche execute itself in a recursive manner.)

```
e,restrt↲
```

(This executes the restart cliche. The first action of the cliche is to exit to location 4 in user core, starting spacewar. At this point space war is running).

```
#ω
```

(This calls the system and continues the execution of the cliche.)

```
now at 44200
```

(This is the location at which spacewar was interrupted. The cliche continues and executes the command "e,restrt↲" is executed and now spacewar is running.

```
#ω

now at 44134
```

(Calling the system again. continues the cliche.)

```
#↲
ODIN
```

(This allows the user to call the system and exit from the cliche.)

```
k↲
n,cliche↲
o,cliche↲
o↲
punch= cliche 110000
p↲
m,cliche,pch↲
end of file.
```

(The user wants to punch this cliche out onto paper tape in order to permantly save it. He names a file, attaches the output pointer to the file, observes that the pointer is correctly attached, gives the command to punch the cliches onto the file, and finally moves the contents of the file onto the punch. When the system is finished moving it types out "end of file.".)

```
d↲
k↲
n,cliche↲
m,rdr,cliche↲
#↲
ODIN
i,cliche↲
y↲
c↲
name    text
restrt  t,4
        h
        e,restrt
        ↲
```

(To begin afresh. The user reads the cliches from paper tape onto a file and then reads them into the system from the file.)

Editing with Expensive Typewriter under ODIN

Expensive typewriter is used exactly as before, except, of course, reading and punching done with drum files rather than directly with paper tape. However, since sense switches cannot be used under ODIN some method other than flipping sense switch one must be used to return control mode from text mode. It turns out that ET jumps to location 440 when sense switch one is on. This means that the act of starting ET at location 440 puts it into control mode. This method, like the sense switch, is subject to the restriction that the last character typed in text mode must have a 'carriage return'. As an example: load Expensive Typewriter by

```
l,et↲
k↲                                  (These are ET commands not ODIN commands)
a↲
Now we are apending to the bufer
but we are making typpin misteaks so
we will hve to edit the textt.
#↲                                  (In order to transfer to location 440 we
ODIN                                must call the system)

t,440↲                              (Now we are back in ET but in control mode.
                                    We will edit the text, going into control
                                    mode by the above method whenever necessary)
lc↲
Now we are appending to the buffer
#↲
ODIN
t,440↲
2c↲
but we are making typing mistakes
#↲
ODIN
t,440↲
3c↲
so we will have to edit the text.
#↲
ODIN
t,440↲
```

It is certainly a nuisance to have to go through such a complicated ritual everytime it is necessary to change from text mode to control mode. Fortunately cliches can make matters considerably simpler. Consider these two cliches:

```
c,et1,(1,et↲
e,et2)
c,et2,(t,440↲
e,et2)
```

When the cliche "etl" is executed it loads and starts Expensive Typewriter. The first time that we must enter control mode we type "#ω". This calls the system and continues the execution of "etl". The effect is to execute the cliche "et2" which transfers to ET control mode. Thereafter, everytime we want to enter control mode we type "#ω" which causes the cliche "et2" to begin again and transfer to location 440. Example:

```
e,etl↲
k↲
a↲
We will edt teh above
with our new brigh
cliches.
#ω                                (Entering control mode)
lc↲
We will edit the above
#ω                                (Entering control mode)
.ℓ↲
We will edit the above
3i↲
shiny
#ω                                (Entering control mode)
w↲
We will edit the above
with our new bright
shiny
cliches.
```

(If we wish we may exit from the recursive cliche to talk to the system for some reason, say naming an output file; we type:)

```
#↲
ODIN
n,outp↲
o,outp↲
t,100↲                            (This is the regular starting place of
                                  Expensive Typewriter.)

p↲                                (This is the ET command which punches the
                                  buffer.)

s↲                                (This command punches a stop code.)
#↲
ODIN
m,outp,pch↲                       (This transfers the file to the punch)

end of file.
```

If we want to make use of the line numbering feature of ET which is ordinarily activated by turning on sense switch two, we may do so by changing one command in user core. What is done is to make the code that checks the state of sense switch two go automatically to the line numbering section. We replace the "jmp" command after a skip with a "nop" using the octal debugging feature. The ritual is:

```
720↵
   600727  760000↵
```

("720 " opens the register; it will contain 600727 which is "jmp 727". We replace it with 760000 which is the command "nop".)

```
t,100↵
w↵
```

(We start ET and ask it to write out the buffer.)

```
1    We will edit the above
2    with our new bright
3    shiny
4    cliches.
```

Using Macro under ODIN

The assembler Macro is used just as before, except, of course, reading and punching are done with drum files rather than directly with paper tape. The procedure is to load Macro with the command "l,macro ". Macro then halts waiting for the tape to be loaded. Attach the input pointer to the file containing the english to be a assembled; e.g. if the english is in a file named "eng", use the command "i,eng↵". To simulate pressing the continue switch type "t↵". Macro now reads from the file "eng", doing pass one of the assembly process. When pass one is complete Macro halts. Re-attach the input pointer to the beginning "eng" with "i,eng↵", since pass two reads the english again. Punching should be done onto a drum file, say "bin". Attach the output pointer to this file with "o,bin↵". Simulate the continue switch with "t↵". When pass two is completed punch a jump block by typing "t↵".

This process may be made into a cliche as follows:

```
c,assem,(l,macro↵
i,eng↵
t↵
i,eng↵
o,bin↵
t↵
t)
```

This cliche works because each time a user program halts the system is called. If the system is in the middle of executing a cliche, it continues from where it left off.

A two (or more) tape assembly can be handled with a slightly more complicated cliche. Macro begins at location 0, so simulate pressing the start switch with "t,0↵". For example:

```
c,assem,(l,macro↵
i,engl↵
t↵
i,eng2↵
t,0↵
i,engl↵
o,bin↵
t↵
i,eng2↵
t,0↵
t)
```

There is one problem in using these assembly cliches, namely, Macro halts inopportunely whenever it encounters an error condition in the assembly. This throws the whole cliche execution out of synchronization. This problem can be cured by making some octal patches and starting at location 1421 instead of "pressing continue" to start pass one. A cliche ritual for this is:

```
c,assem,(e,fix↲
i,eng↲
t,1421↲
i,eng↲
o,bin↲
t↲
t)
```

```
c,fix,(1,macro↲
0↲
760200↲
1421↲
760000↲
3635↲
600000)
```

This cliche uses the octal debugging feature of ODIN to make the patches that cause Macro to eliminate all error halts. Now the cliche assemblies will proceed in all cases until completion of the entire assembly

Using the Macro Symbol Package under ODIN

The Macro Symbol Package has been modified so that it only punches and does not type out any symbol lists. Therefore, since there are no choices, the sense switches may be left down. The symbol package is loaded immediately after running an assembly with the command "l,macsym↵". It then waits for a title to be typed from the console. When the title is terminated with the "carriage return", the symbols are punched onto the current output file. If desired the symbol punch may be included in the assembly cliche:

```
c,assem,(e,fix↵                 See above for the definition of "fix".
i,eng↵
t,1421↵
i,eng↵
o,bin↵
t↵
t↵
l,macsym)                       After the symbol package is loaded, type
                                a title to initiate the punching onto the
                                file "bin".
```

Examples of Advanced Cliches:

1. The user has a program which does a data analysis on alphanumeric tape punching out the analysis as it reads the data. Assume the program begins at location 100 and then halts waiting for its input tape. After each analysis the program halts. There are several tapes to process so the following cliches would be useful:

```
c,first,(k↲
n,prog↲
w↲
m,rdr,prog↲
i,prog↲
r↲
e,second).
```

The "first" cliche reads the program into the file "prog". The command "w↲" is used so that the user can wait until the reader is free. After the "m,rdr,prog↲" is executed the user will call the system with "#ω" to remain in the cliche. After "r↲" is executed, the analysis program begins at location 100 and then halts. This has the effect of continuing the cliche.

```
c,second,(n,data↲
w↲
m,rdr,data↲
n,out↲
i,data↲
o,out↲
t↲
e,third)
```

The "second" cliche moves the first data tape onto the file "data". Typing "#ω" continues the cliche. It then attaches the input and output pointers and allows the analysis program to continue. When the analysis is finished, control passes to the "third" cliche.

```
c,third,(m,out,pch↲
w↲
m,rdr,data↲
k,out↲
n,out↲
t,100↲
i,data↲
o,out↲
t↲
e,third)
```

This cliche punches out the last analysis. Then it reads in the next data tape. Typing "#ω" continues the cliche. It names an output file and starts the analysis program. The program halt returns control to the cliche, whereupon it sets up the input and output pointers and continues the program. When the analysis is finished the "third" cliche is restarted. It is executed as many times as necessary to process all the data tapes.

2. The user has an english program to edit, assemble, and debug. Before he loads the english tape, he loads an english tape with the following list of system commands and cliches into a file. (The tape must end in a stop code):

```
d,e,d,et,etl,begin↲
d,lm,mac,macl↲
c,d,(t,60000↲               A recursive cliche for entering DDT
e,d)↲
c,e,(t,440↲                 A recursive cliche for entering ET
e,e)                        control mode
c,et,(l,et↲                 A cliche for entering ET, setting up
e,etl↲                      files, starting ET, and initially going
t,100↲                      control mode.
e,e)
c,etl,(i,eng↲               This cliche is called by "et" to set
k,neweng↲                   up file pointers.
n,neweng↲
o,neweng)
c,lm,(l,macro↲              A cliche for the assembly
0↲
760200↲
1421↲
760000↲
3635↲
600000↲
i,eng↲
t,1421↲
i,eng↲
t↲
t)
c,mac,(k,bin↲               The master cliche for setting up files,
m,neweng,eng↲               calling the assembly cliche,
k,neweng↲                   doing a symbol punch, and arranging
n,bin↲                      for the loading of DDT.
o,bin↲
e,lm↲
l,macsym↲
e,macl)
c,macl,(i,bin↲              This cliche sets up the file pointers for
k,outp↲                     DDT input and output, loads DDT, and sets
n,outp↲                     up a call to the recursive cliche "d".
o,outp↲
l,ddt↲
e,d)
c,begin,(k,eng↲             The cliche for reading in the english
n,eng↲                      tape and calling the editing cliche.
w↲
m,rdr,eng↲
e,et)
```

To load the tape use:

```
k↵
n,cliches↵
m,rdr,cliches↵
#↵
ODIN
i,cliches↵
v↵
```

Then begin the editing process with

```
e,begin↵
```

The system will type out:

to continue type →

When the reader is free type:

→

After the english tape has been moved onto the drum recall the system:

#ω

This passes control to the cliche "et". Expensive Typewriter will be loaded; to continue to execute the cliche type:

#ω

This cliche calls "etl" to set up the pointers and restarts ET. Now begin editing as usual. The first time it is necessary to enter control mode type:

#ω

This passes control to the cliche "e". Whenever control mode must be entered type:

#ω

When the editing is finished, it is time to do the assembly. Type:

```
e,mac
```

When this cliche finishes the new english will be in file "eng", and the binary with symbol punch will be in file "bin". (Do not forget that the macro symbol punch waits for a title. Because there are input buffers, the title may be typed anytime after the cliche "mac" has begun.) The cliche finishes by setting up the input and output pointer for DDT and loading it. To read in the binary program to DDT type its control characters:

Z	Zero core.
K	Kill the symbol table.
Y	Read in the binary program.
T	Read in the symbols.

When it is necessary to go from the binary program back to DDT type:

```
#ω
```

This passes control onto the cliche "d" which recalls DDT. Whenever it is necessary to recall DDT thereafter type:

```
#ω
```

When the debugging is finished, punch out the binary from DDT. The file "outp" has been set up for that purpose. If copies of the files are wanted they can be moved to the punch:

```
m,outp,pch↵
m,eng,pch↵
m,bin,pch↵
```

APPENDIX I

Error Messages

bff - Buffer Full. This indicates that more than 36 characters have been typed into the ODIN interpreter without any 'carriage return'.

cbf - Cliche Buffer Full. This indicates that the text of a cliche has exceeded 70 characters. To avoid this trouble use nested cliches, i.e. where one cliche calls another.

cfe - Cliche Format Error. This means that some format mistake has been made in defining a cliche. To correct is just retype the cliche definition.

cnu - Cliche Name Undefined. This message indicates that the user has asked to execute a cliche that has not been defined.

gfe - General Format Error. This means that some format error has been made in entering a system control command.

ion - Illegal Octal Number. This message is typed out when the user attempts to directly attach either the input or output pointers outside the legal limits.

mfe - Move Format Error. This means that a move file command has been incorrectly typed.

nad - Name Already Defined. This means that either the name in a file naming command or the cliche name in a cliche definition has already been defined.

noc - Non Octal Character. This means that a character not in the set 0,1,2,3,4,5,6,7 has been typed when ODIN expected a purely octal number. Such times are in octal debugging mode and in directly attaching an input or output pointer.

ntl - Name Too Long. This indicates that an identifier has exceeded six characters.

udn - Undefined Name. This means that either a file name or cliche name referred to in the control command in undefined.

exceeded drum space - This means that the user has attempted to read or punch beyond his allotted drum space. The only recovery is to move un-needed files to the real punch to make room for additional paper tape images.

halt at #1 ac- #2 io-#3 - This means that a user or service program has halted at location #1 with the contents of the accumulator equal to #2 and the contents of the I - O register equal to #3.

ilg instr at #1 ac-#2 io-#3 - This means that the user or service program attempted to execute an illegal instruction at location #1. The contents of the AC and IO are indicated as above.

ilg iot at #1 ac-#2 io-#3 - This means that the user program attempted to execute an illegal "iot" command at location #1. AC and IO as above.

nesting too deep - This means that cliches have been nested to a depth greater than 8.

too many cliches - This means that more than 13 cliches have been defined.

too many names - This means that more than 18 files have been defined.

APPENDIX II

ADD	40	YYYY	ADD
AND	02	YYYY	LOGICAL AND
ASC	72	CC51	ACTIVATE SEQ BREAK CHANNEL CC
CAC	72	0053	CLEAR ALL CHANNELS
CAL	16	YYYY	CALL SUBROUTINE
CBS	72	0056	CLEAR SEQ BREAK SYSTEM
CDF	74	6000	CLEAR IO, TRANSFER (PF) TO IO
CKS	72	0033	CHECK STATUS [EXCLUDING TELETYPES]
CLA	76	0200	CLEAR AC
CLF	76	000F	CLEAR SELECTED PROGRAM FLAGS
CLI	76	4000	CLEAR IO
CLO	65	1600	CLEAR OVERFLOW
CMA	76	1000	COMPLEMENT (AC) [ONES COMPLEMENT]
CMI	77	0000	COMPLEMENT (IO) [ONES COMPLEMENT]
CTF	75	2000	CLEAR AND TRANSFER FLAGS <CLEAR RNG>
DAC	24	YYYY	DEPOSIT (AC)
DAP	26	YYYY	DEPOSIT ADDRESS PART
DBA	72	2061	DRUM BREAK ADDRESS
DCH	14	YYYY	DEPOSIT A CHARACTER
DCL	72	0063	DRUM CORE LOC"N
DCT	72	0710	IBM DISK CONTROL
DEN	72	0110	IBM DISK END
DFI	74	4000	DEPOSIT (PF) IN IO [INCLUSIVE OR]
DIA	72	0061	DRUM INITIAL ADDRESS
DIO	32	YYYY	DEPOSIT (IO)
DIP	30	YYYY	DEPOSIT INSTRUCTION PART
DIV	56	YYYY	DIVIDE [WITH POSSIBLE SKIP]
DPY	72	0007	DISPLAY ONE POINT ON CRT
DRA	72	2062	DRUM REQUEST ADDRESS
DSC	72	CC50	DEACTIVATE SEQ BREAK CHANNEL CC
DRD	72	0510	IBM DISK READ
DRS	72	0010	IBM DISK RESET
DSN	72	0410	IBM DISK SENSE
DWC	72	0062	DRUM WORD COUNT
DWR	72	0610	IBM DISK WRITE
DZM	34	YYYY	DEPOSIT ZERO IN MEMORY
EEM	72	4074	ENTER EXTEND MODE
ERM	72	0065	ENTER RESTRICT MODE
ESM	72	0055	ENTER SEQ BREAK MODE
GCF	72	0127	CLEAR LIGHT-PEN STATUS BIT
GLF	72	2026	LOAD SYMBOL GEN FORMAT
GPL	72	2027	GENERATOR PLOT LEFT
GPR	72	0027	GENERATOR PLOT RIGHT
GSP	72	0026	GENERATOR SPACE
HLT	76	0400	HALT
I90	72	0446	INITIATE "90 DD INTERRUPT
IDC	74	1000	INDEX CHARACTER
IDX	44	YYYY	INDEX
IOR	04	YYYY	INCLUSIVE OR
IOT	72	NNNN	IN-OUT TRANSFER GROUP
ISB	72	CC52	INITIATE SEQ BREAK ON CHANNEL CC
ISP	46	YYYY	INDEX AND SKIP IF POSITIVE

JDA	17 YYYY	JUMP AND DEPOSIT (AC)
JMP	60 YYYY	JUMP
JSP	62 YYYY	JUMP AND SAVE PROGRAM COUNTER
LAC	20 YYYY	LOAD AC
LAI	76 0040	LOAD AC WITH (IO)
LAP	76 0100	LOAD AC WITH (PC)
LAT	76 2200	LOAD AC FROM TEST WORD
LAW	70 NNNN	LOAD AC WITH NNNN
LAW	71 NNNN	LOAD AC WITH -NNNN
LCH	12 YYYY	LOAD A CHARACTER
LEM	72 0074	LEAVE EXTEND MODE
LIA	76 0020	LOAD IO WITH (AC)
LIO	22 YYYY	LOAD IO
LRM	72 0064	LEAVE RESTRICT MODE
LSM	72 0054	LEAVE SEQ BREAK MODE
MUL	54 YYYY	MULTIPLY
NOP	76 0000	NO OPERATION
OPR	76 NNNN	OPERATE GROUP
PPA	72 0005	PUNCH PERF TAPE, ALPHA
PPB	72 0006	PUNCH PERF TAPE, BINARY
RAL	66 1NNN	ROTATE AC LEFT
RAR	67 1NNN	ROTATE AC RIGHT
RCK	72 0032	READ MILLISEC CLOCK
RCL	66 3NNN	ROTATE COMBINED AC AND IO LEFT
RCR	67 3NNN	ROTATE COMBINED AC AND IO RIGHT
RCV	72 0031	READ A/D CONVERTER
RHL	72 0035	READ HIGH-ENERGY LABS DATA LINES
RIL	66 2NNN	ROTATE IO LEFT
RIR	67 2NNN	ROTATE IO RIGHT
RKB	72 0037	READ KEYBOARD BUFFER
RLD	72 1336	READ LOCATION COUNTER [131D-"90]
RLM	72 0036	READ LOCATION COUNTER [131M-PHILCO]
RNG	75 0100	ENTER RNG MODE IF IO BIT 11=1
RNG	75 2100	ENTER RNG MODE IF IO BIT 11=1
RPA	72 0001	READ PERF TAPE, ALPHA
RPB	72 0002	READ PERF TAPE, BINARY
RPR	72 1U12	READ IBM PROJECTOR UNIT U
RPS	72 0012	READ IBM PROJECTOR STATUS
RRB	72 0030	READ READER BUFFER
RSR	72 0011	READ SWITCH REGISTER
RTS	72 0X34	READ TELETYPE STATUS REGISTER
RTY	72 1U34	READ TELETYPE UNIT U
SAD	50 YYYY	SKIP IF (AC) ≠ (YYYY)
SAL	66 5NNN	SHIFT AC LEFT
SAR	67 5NNN	SHIFT AC RIGHT
SAS	52 YYYY	SKIP IF (AC) = (YYYY)
SCL	66 7NNN	SHIFT COMBINED AC AND IO LEFT
SCR	67 7NNN	SHIFT COMBINED AC AND IO RIGHT
SDB	72 2007	SET DISPLAY BUFFER, NO INTENSITY
SDF	72 0146	STOP DATA FLOW [131M-"90]
SFT	66 NNNN	SHIFT GROUP
SID	72 1346	SET INITIAL ADDRESS [131D-"90]
SIL	66 6NNN	SHIFT IO LEFT
SIM	72 0346	SET INITIAL ADDRESS [131M-PHILCO]
SIR	67 6NNN	SHIFT IO RIGHT

SKP	64 NNNN	SKIP GROUP
SMA	64 0400	SKIP IF (AC) < 0
SNI	64 4000	SKIP IF (IO) ≠ 0
SPA	64 0200	SKIP IF (AC) ≥ 0
SPI	64 2000	SKIP IF (IO) ≥ 0
SRB	72 0021	SET RELAY BUFFER
STF	76 001F	SET SELECTED PROGRAM FLAG
SUB	42 YYYY	SUBTRACT
SWD	72 0546	SET WORD COUNT [131D-"90]
SWM	72 X046	SET WORD COUNT [131M-PHILCO]
SWP	76 0060	SWAP (AC) AND (IO)
SZA	64 0100	SKIP IF (AC) = +0
SZF	64 000F	SKIP IF SELECTED FLAG = 0
SZI	65 4000	SKIP IF (IO) = +0
SZO	64 1000	SKIP IF OVERFLOW = 0, CLEAR OVFLO
SZS	64 00S0	SKIP IF SWITCH S = 0
TIF	75 0000	TRANSFER IO TO PROG FLAGS [INCL OR]
TYI	72 0004	TYPE IN
TYO	72 0003	TYPE OUT
WPR	72 2U12	WRITE IBM PROJECTOR UNIT U
WTY	72 1U66	WRITE TELETYPE UNIT U
XCT	10 YYYY	EXECUTE INSTRUCTION IN YYYY
XOR	06 YYYY	EXCLUSIVE OR

PDP OP-CODES, NUMERIC

AND	02 YYYY	LOGICAL AND
IOR	04 YYYY	INCLUSIVE OR
XOR	06 YYYY	EXCLUSIVE OR
XCT	10 YYYY	EXECUTE INSTRUCTION IN YYYY
LCH	12 YYYY	LOAD A CHARACTER
DCH	14 YYYY	DEPOSIT A CHARACTER
CAL	16 YYYY	CALL SUBROUTINE
JDA	17 YYYY	JUMP AND DEPOSIT (AC)
LAC	20 YYYY	LOAD AC
LIO	22 YYYY	LOAD IO
DAC	24 YYYY	DEPOSIT (AC)
DAP	26 YYYY	DEPOSIT ADDRESS PART
DIP	30 YYYY	DEPOSIT INSTRUCTION PART
DIO	32 YYYY	DEPOSIT (IO)
DZM	34 YYYY	DEPOSIT ZERO IN MEMORY
ADD	40 YYYY	ADD
SUB	42 YYYY	SUBTRACT
IDX	44 YYYY	INDEX
ISP	46 YYYY	INDEX AND SKIP IF POSITIVE
SAD	50 YYYY	SKIP IF (AC) ≠ (YYYY)
SAS	52 YYYY	SKIP IF (AC) = (YYYY)
MUL	54 YYYY	MULTIPLY
DIV	56 YYYY	DIVIDE [WITH POSSIBLE SKIP]
JMP	60 YYYY	JUMP
JSP	62 YYYY	JUMP AND SAVE PROGRAM COUNTER
SZF	64 000F	SKIP IF SELECTED FLAG = 0
SZS	64 00S0	SKIP IF SWITCH S = 0
SZA	64 0100	SKIP IF (AC) = +0
SPA	64 0200	SKIP IF (AC) ≥ 0
SMA	64 0400	SKIP IF (AC) < 0
SZO	64 1000	SKIP IF OVERFLOW = 0, CLEAR OVFLO

SPI	64 2000	SKIP IF (IO) ≥ 0
SNI	64 4000	SKIP IF (IO) ≠ 0
SKP	64 NNNN	SKIP GROUP
CLO	65 1600	CLEAR OVERFLOW
SZI	65 4000	SKIP IF (IO) = +0
RAL	66 1NNN	ROTATE AC LEFT
RIL	66 2NNN	ROTATE IO LEFT
RCL	66 3NNN	ROTATE COMBINED AC AND IO LEFT
SAL	66 5NNN	SHIFT AC LEFT
SIL	66 6NNN	SHIFT IO LEFT
SCL	66 7NNN	SHIFT COMBINED AC AND IO LEFT
SFT	66 NNNN	SHIFT GROUP
RAR	67 1NNN	ROTATE AC RIGHT
RIR	67 2NNN	ROTATE IO RIGHT
RCR	67 3NNN	ROTATE COMBINED AC AND IO RIGHT
SAR	67 5NNN	SHIFT AC RIGHT
SIR	67 6NNN	SHIFT IO RIGHT
SCR	67 7NNN	SHIFT COMBINED AC AND IO RIGHT
LAW	70 NNNN	LOAD AC WITH NNNN
LAW	71 NNNN	LOAD AC WITH -NNNN
RPA	72 0001	READ PERF TAPE, ALPHA
RPB	72 0002	READ PERF TAPE, BINARY
TYO	72 0003	TYPE OUT
TYI	72 0004	TYPE IN
PPA	72 0005	PUNCH PERF TAPE, ALPHA
PPB	72 0006	PUNCH PERF TAPE, BINARY
DPY	72 0007	DISPLAY ONE POINT ON CRT
DRS	72 0010	IBM DISK RESET
RSR	72 0011	READ SWITCH REGISTER
RPS	72 0012	READ IBM PROJECTOR STATUS
SRB	72 0021	SET RELAY BUFFER
GSP	72 0026	GENERATOR SPACE
GPR	72 0027	GENERATOR PLOT RIGHT
RRB	72 0030	READ READER BUFFER
RCV	72 0031	READ A/D CONVERTER
RCK	72 0032	READ MILLISEC CLOCK
CKS	72 0033	CHECK STATUS [EXCLUDING TELETYPES]
RHL	72 0035	READ HIGH-ENERGY LABS DATA LINES
RLM	72 0036	READ LOCATION COUNTER [131M-PHILCO]
RKB	72 0037	READ KEYBOARD BUFFER
CAC	72 0053	CLEAR ALL CHANNELS
LSM	72 0054	LEAVE SEQ BREAK MODE
ESM	72 0055	ENTER SEQ BREAK MODE
CBS	72 0056	CLEAR SEQ BREAK SYSTEM
DIA	72 0061	DRUM INITIAL ADDRESS
DWC	72 0062	DRUM WORD COUNT
DCL	72 0063	DRUM CORE LOC"N
LRM	72 0064	LEAVE RESTRICT MODE
ERM	72 0065	ENTER RESTRICT MODE
LEM	72 0074	LEAVE EXTEND MODE
DEN	72 0110	IBM DISK END
GCF	72 0127	CLEAR LIGHT-PEN STATUS BIT
SDF	72 0146	STOP DATA FLOW [131M-"90]
SIM	72 0346	SET INITIAL ADDRESS [131M-PHILCO]
DSN	72 0410	IBM DISK SENSE

I90	72 0446	INITIATE "90 DD INTERRUPT
DRD	72 0510	IBM DISK READ
SWD	72 0546	SET WORD COUNT [131D-"90]
DWR	72 0610	IBM DISK WRITE
DCT	72 0710	IBM DISK CONTROL
RTS	72 0X34	READ TELETYPE STATUS REGISTER
RLD	72 1336	READ LOCATION COUNTER [131D-"90]
SID	72 1346	SET INITIAL ADDRESS [131D-"90]
RPR	72 1U12	READ IBM PROJECTOR UNIT U
RTY	72 1U34	READ TELETYPE UNIT U
WTY	72 1U66	WRITE TELETYPE UNIT U
SDB	72 2007	SET DISPLAY BUFFER, NO INTENSITY
GLF	72 2026	LOAD SYMBOL GEN FORMAT
GPL	72 2027	GENERATOR PLOT LEFT
DBA	72 2061	DRUM BREAK ADDRESS
DRA	72 2062	DRUM REQUEST ADDRESS
WPR	72 2U12	WRITE IBM PROJECTOR UNIT U
EEM	72 4074	ENTER EXTEND MODE
DSC	72 CC50	DEACTIVATE SEQ BREAK CHANNEL CC
ASC	72 CC51	ACTIVATE SEQ BREAK CHANNEL CC
ISB	72 CC52	INITIATE SEQ BREAK ON CHANNEL CC
IOT	72 NNNN	IN-OUT TRANSFER GROUP
SWM	72 X046	SET WORD COUNT [131M-PHILCO]
IDC	74 1000	INDEX CHARACTER
DFI	74 4000	DEPOSIT (PF) IN IO [INCLUSIVE OR]
CDF	74 6000	CLEAR IO, TRANSFER (PF) TO IO
TIF	75 0000	TRANSFER IO TO PROG FLAGS [INCL OR]
RNG	75 0100	ENTER RNG MODE IF IO BIT 11=1
CTF	75 2000	CLEAR AND TRANSFER FLAGS <CLEAR RNG>
RNG	75 2100	ENTER RNG MODE IF IO BIT 11=1
NOP	76 0000	NO OPERATION
CLF	76 000F	CLEAR SELECTED PROGRAM FLAGS
STF	76 001F	SET SELECTED PROGRAM FLAG
LIA	76 0020	LOAD IO WITH (AC)
LAI	76 0040	LOAD AC WITH (IO)
SWP	76 0060	SWAP (AC) AND (IO)
LAP	76 0100	LOAD AC WITH (PC)
CLA	76 0200	CLEAR AC
HLT	76 0400	HALT
CMA	76 1000	COMPLEMENT (AC) [ONES COMPLEMENT]
LAT	76 2200	LOAD AC FROM TEST WORD
CLI	76 4000	CLEAR IO
OPR	76 NNNN	OPERATE GROUP
CMI	77 0000	COMPLEMENT (IO) [ONES COMPLEMENT]
	PDP-1 IOT INSTRUCTIONS, BY CLASS	
RPA	72 0001	READ PERF TAPE, ALPHA
RPB	72 0002	READ PERF TAPE, BINARY
TYO	72 0003	TYPE OUT
TYI	72 0004	TYPE IN
PPA	72 0005	PUNCH PERF TAPE, ALPHA
PPB	72 0006	PUNCH PERF TAPE, BINARY
DPY	72 0007	DISPLAY ONE POINT ON CRT
SDB	72 2007	SET DISPLAY BUFFER, NO INTENSITY
DRS	72 0010	IBM DISK RESET
DEN	72 0110	IBM DISK END

DSN	72 0410	IBM DISK SENSE
DRD	72 0510	IBM DISK READ
DWR	72 0610	IBM DISK WRITE
DCT	72 0710	IBM DISK CONTROL
RSR	72 0011	READ SWITCH REGISTER
RPS	72 0012	READ IBM PROJECTOR STATUS
RPR	72 1U12	READ IBM PROJECTOR UNIT U
WPR	72 2U12	WRITE IBM PROJECTOR UNIT U
SRB	72 0021	SET RELAY BUFFER
GSP	72 0026	GENERATOR SPACE
GLF	72 2026	LOAD SYMBOL GEN FORMAT
GPR	72 0027	GENERATOR PLOT RIGHT
GCF	72 0127	CLEAR LIGHT-PEN STATUS BIT
GPL	72 2027	GENERATOR PLOT LEFT
RRB	72 0030	READ READER BUFFER
RCV	72 0031	READ A/D CONVERTER
RCK	72 0032	READ MILLISEC CLOCK
CKS	72 0033	CHECK STATUS [EXCLUDING TELETYPES]
RTS	72 0X34	READ TELETYPE STATUS REGISTER
RTY	72 1U34	READ TELETYPE UNIT U
RHL	72 0035	READ HIGH-ENERGY LABS DATA LINES
RLM	72 0036	READ LOCATION COUNTER [131M-PHILCO]
RLD	72 1336	READ LOCATION COUNTER [131D-"90]
RKB	72 0037	READ KEYBOARD BUFFER
SDF	72 0146	STOP DATA FLOW [131M-"90]
SIM	72 0346	SET INITIAL ADDRESS [131M-PHILCO]
I90	72 0446	INITIATE "90 DD INTERRUPT
SWD	72 0546	SET WORD COUNT [131D-"90]
SID	72 1346	SET INITIAL ADDRESS [131D-"90]
SWM	72 X046	SET WORD COUNT [131M-PHILCO]
DSC	72 CC50	DEACTIVATE SEQ BREAK CHANNEL CC
ASC	72 CC51	ACTIVATE SEQ BREAK CHANNEL CC
ISB	72 CC52	INITIATE SEQ BREAK ON CHANNEL CC
CAC	72 0053	CLEAR ALL CHANNELS
LSM	72 0054	LEAVE SEQ BREAK MODE
ESM	72 0055	ENTER SEQ BREAK MODE
CBS	72 0056	CLEAR SEQ BREAK SYSTEM
DIA	72 0061	DRUM INITIAL ADDRESS
DBA	72 2061	DRUM BREAK ADDRESS
DWC	72 0062	DRUM WORD COUNT
DRA	72 2062	DRUM REQUEST ADDRESS
DCL	72 0063	DRUM CORE LOC"N
LRM	72 0064	LEAVE RESTRICT MODE
ERM	72 0065	ENTER RESTRICT MODE
WTY	72 1U66	WRITE TELETYPE UNIT U
LEM	72 0074	LEAVE EXTEND MODE
EEM	72 4074	ENTER EXTEND MODE

APPENDIX III

The following is a list and explanation of the transliteration between the teletype and typewriter characters under the ODIN Time Sharing System.

A - Z and 0 - 9 are translated directly.
The special characters are as follows:

Teletype	Concise(Typewriter)
! (Exclamation Point)	| (Vertical Bar)
"	"
#	~
$	⊃
%	∨
&	∧
'	'
>	>
<	<
+	+
=	=
-	-
;	˙,(Center Dot Comma)
:	˙.(Center Dot Period)
*	x (Times Sign)
,	,
.	.
?	?
/	/
←	→
@	– (Over Strike)
TAB	TAB
CARRIAGE RETURN	CARRIAGE RETURN
RU	_ (Under Bar)
\	· (Center Dot)
LINE FEED	NULL
(	(

)		)
[	(Shift K)	[
]	(Shift M)	]
↑		↑
ALT MODE		UPPER CASE SHIFT (ALPHABETIC ONLY)
RUBOUT		BACKSPACE

All other teletype characters (e.g. 'BELL', 'EOT', Etc.) are translated NULL and are invisible to the 'TYI' instruction.

'Alt Mode' acts as upper case shift for alphabetic characters only: All upper case special characters are handled automatically. It is necessary to type 'Alt Mode' before each upper case alphabetic character desired, because a downshift is automatically inserted before the second alphabetic after an 'Alt Mode'. For example, to insert 'TITLE' from a teletype, one would type 'Alt Mode' 't' 'Alt Mode' 'i' 'Alt Mode' 't' 'Alt Mode' 'l' 'Alt Mode' 'e'.

Upper case type out materializes as the character preceded by backward slash, hence the example word would be typed out as \T\I\T\L\E on the teletype.

The 'Shift' key on the teletype changes a key from the lower character to the upper, as printed on the key. The 'control' key causes the generation on those control functions indicated in writing on the top half of the key. The only 'control' character used by ODIN is 'tab; which translates to typewriter 'tab' and materializes as a number of spaces. All others are ignored.

It is important not to confuse the use of the 'Shift' key and 'Alt Mode'. 'Shift' changes a key from one teletype character to another, whereas 'Alt Mode' causes the character to be in typewriter upper case. (e.g., 'Shift' n is the character '↑', while 'Alt Mode; n is N.)

APPENDIX IV

Limits on the user drum file space

User 0	(teletype)	110000	-	157777
User 1	(teletype)	160000	-	227777
User 2	(teletype)	300000	-	317777
User 3	(TWX')	320000	-	337777
User 4	(typewriter)	230000	-	277777

APPENDIX V

It is proposed to provide limited system services, under ODIN, to aid users in running the Philco displays. These services are:

1. Sorting and buffering of input from the keyboards. (User program executes an "RKB" instruction and receives the next character from his keyboard.

2. System maintenance of all displays. (The user is responsible for setting up the display buffer).

Each of ODIN's consoles will have one Philco display and keyboard associated with it. All input from the keyboard will be placed in that keyboard's input buffer. Only the user associated with that display will be able to read its keyboard. The "RKB" instruction will take the next character, including "Special Button Bits" but not unit number bits, from the buffer. If there is no character in the buffer the program will be dismissed until a character is typed.

Each user will have a display buffer area in core 3. He will be able to reference it in extend mode as if he had the bare machine. The buffer for each user will extend from 30001_8 to 31000_8. (The system will relocate the address part of all instructions appropriately for each user). Any extend mode references outside this area will cause a system error message and discontinuance of the program.

The system will maintain the display buffer as directed by the program. The display area will always begin at the start of the buffer, but the word count can be set by the "sum" instruction. If the word count is set to zero the display will be stopped. Any word count greater than 1000_8 will be taken to be 1000_8.

Addendum No. 1 to TS Memo No. 23
(User's Manual to the ODIN Time
Sharing System)

November 17, 1964

Additions and Changes to the ODIN Time Sharing System

by Harold Gilman

A number of additions and changes to the ODIN system have been made in order to allow the use of the Philco displays and keyboards within the system. There are 12 displays, six in the new building and six in the area around the PDP-1. Each of the five users in the ODIN system is allotted two of the displays and keyboards, with the extra two being given to the two less frequently used consoles in order to facilitate testing of the displays themselves. Each user has at least one display and keyboard in each of the rooms.

Each user has also been allotted a fixed area in core 3 to be used as a display buffer. This area is referenced in extend mode as if it were locations 30000 - 30773; the system relocates all references so that they refer to the proper location. All instructions which look at or change these locations may be performed, but jumps or other transfers of control will be treated as illegal instructions and interrupt execution of the program. Any attempt to reference locations outside the proper area will cause execution to be interrupted and the error message "ilg mem ref" to be printed by the system.

The contents of the buffer should be the same as if the Philco's were being run on the bare machine. (See memo 17, Supplement to the PDP-1 Handbook, by Gary Feldman). In order to prevent one user from displaying on another user's console, and also to insure that the proper consoles are selected for each user, the system maintains four words of console selecting code at the beginning of each buffer. This area is inaccessible to the user. Any Philco control word (a word which has the form 74xxxx) which is placed in the buffer area will be changed so as to have the form 7476xx. This allows the user to change mode, brightness, and size; but prevents him from changing the console select. Whether or not the buffer area is being displayed the system will make that change and not say a word about it.

There are two instructions which affect the display of information on the Philco screens. The user may tell the system to begin displaying at a certain location by placing that location in the I-O register and executing the "sim" instruction (720346). This sets the initial location being displayed. This will not change until the next "sim" instruction is executed. The location specified must be between 30000 and 30773 inclusive. The word count of the displayed information is set with the "swm" instruction (720046). A word count of zero will cause no display.

The desired word count is loaded in the I-O and the "swm" executed. The word count will remain the same until changed by another "swm". The word count may be as large as 773_8, but the sum of word count and initial location must not be greater than 30773. If the word count is too large, the system will adjust it so that it is the maximum allowable for the current initial location. Once the location and word count are set, the system will continue displaying with those specifications at a thirty cycle repetition rate until either is changed. The display can be stopped only by setting the word count to zero.

Keyboard input is interrogated with the "rkb" instruction (720037). When "rkb" is executed, the next character from the keyboards associated with the user's displays will be placed in the I-O register. The input is untranslated eight bit code of the form sscccccc, where the s bits are the special buttons on the keyboard and cccccc is the six bit character code. The character is placed in the low order end of the I-O. If there are no more characters in the input buffer, the program will be dismissed upon execution of "rkb". At present it is impossible for a program to get typewriter concise code from the Philco keyboards. The instruction 720417 is available to test the state of the Philco input buffers. If the buffer is empty, the instruction will skip the next instruction. (730417 will skip if the buffer is not empty.) This instruction should only be used if it is imperative that a program run while waiting for keyboard input.

In addition to the aforementioned changes, two new features have been added to the general operations of ODIN. In order that programs may be written to be compatible with both time-sharing and the bare machine, a "skip on ODIN" instruction (720617) has been implemented. This instruction is a "nop" on the bare machine, but causes the next instruction to be skipped when the program is running under ODIN.

The cliche feature of ODIN has also been expanded slightly to add versatility. Skip instruction tests of the A-C, I-O, and program flags may now be included within cliches. If the instruction would skip the next instruction in a program, it will skip the next system command in the current cliche. This command has the form "s,xxxxxx", where xxxxxx is the octal for a skip instruction. For example:

A user defines the following cliche to the system:

```
c,foo,(s,640006
e,foo1
e,foo2)
```

During the execution of this cliche, if program flag six were zero, the cliche would skip the execution of 'foo1' and proceed with the execution of 'foo2'. (640006 is octal for "szf 6").

It is possible with this system to have program control of cliches, since a program could go "stf 6", "hlt", and cause 'fool' to be executed within the cliche 'foo'.

Any questions or problems about this new version of ODIN should be referred to Gary Feldman or Harold Gilman.

NOTE: In order to prevent accidental destruction of files, the system command "k <carriage return >" has been replaced with "k,all < carriage return >".