Burdock User's Guide

by J. Pitts Jarvis, III

February 2, 1981

XEROX
Office Products Division
3450 Hillview Avenue / Palo Alto / California 94304

Introduction

Burdock provides the tools necessary to load and debug programs written for the Dandelion. Dandelion consists of two processors, the IOP and the CP. The IOP is an Intel 8085 microcomputer with 32K bytes of PROM and 16K bytes of RAM. The CP is a 2901 bit slice microprogrammed processor with 16 bit wide data paths, and a control store of 4096 48 bit words.

Burdock runs on an Alto connected through its HyType port, hereafter called the umbilical, to the IOP. No direct connection between the Alto and the CP exists. Burdock communicates through the umbilical with the IOP kernel, a program running in the IOP. The IOP kernel in turn communicates through a similar interface to the CP kernel. In addition, the IOP can read and write the control store and a limited set of other CP registers.

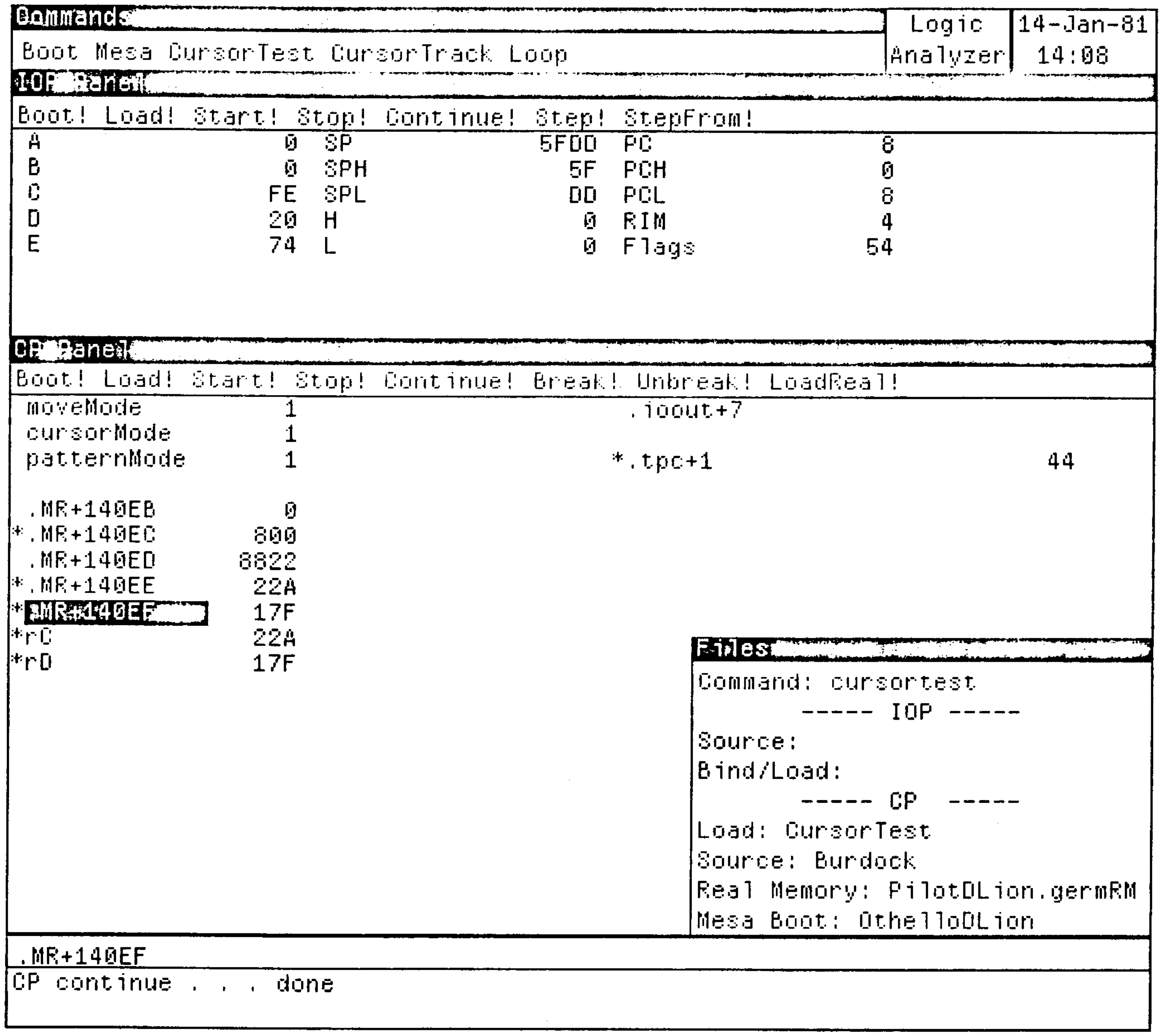
Tajo provides the foundation upon which Burdock builds. This manual assumes that the reader is at least familiar with the Tajo user interface, can manipulate windows, and jockey the cursor around the display. [Reference TUG]

Burdock consists of four main parts, the communication window, the CP panel, the IOP panel, and the logic analyzer. All type in and status messages appear in the communication window. The CP and IOP panels allow the user to examine and alter the state of the respective processors. The logic analyzer formats and displays data captured by the Tektronix 7D01 Logic Analyzer.

Burdock provides two other auxiliary windows: one for the IOP source, and the other for file names. The IOP source provides functions necessary for source level debugging. Burdock commands which need files from the disk get their names from the file window. Users can use the standard Tajo mechanisms to alter these names. Usually this window is tiny and does not occupy valuable display real estate. Menu items are associated with each name so that the user can invoke commands without necessarily making the window large. Any time the cursor is in the file window and a name is selected, typing CR also invokes the associated command.

Communication Window

The communication window occupies the bottom of the display. The top line, hereafter called the type in, collects characters typed from the keyboard when the cursor lies in the background or either of the two panel windows. The user can edit the type in using all of the standard Tajo editing facilities. The bottom two lines are for the timely reporting of status and errors.



```
Commands                                                     Logic     14-Jan-81
Boot Mesa CursorTest CursorTrack Loop                        Analyzer   14:08
IOP Panel
Boot! Load! Start! Stop! Continue! Step! StepFrom!
 A                 0  SP            5FDD  PC            8
 B                 0  SPH             5F  PCH           0
 C                FE  SPL             DD  PCL           8
 D                20  H                0  RIM           4
 E                74  L                0  Flags        54

CP Panel
Boot! Load! Start! Stop! Continue! Break! Unbreak! LoadReal!
 moveMode          1                    .ioout+7
 cursorMode        1
 patternMode       1                   *.tpc+1                    44

 .MR+140EB         0
*.MR+140EC       800
 .MR+140ED      8822
*.MR+140EE       22A
*.MR+140EF       17F
*rC              22A                            Files
*rD              17F                            Command: cursortest
                                                      ----- IOP -----
                                                Source:
                                                Bind/Load:
                                                      ----- CP  -----
                                                Load: CursorTest
                                                Source: Burdock
                                                Real Memory: PilotDLion.germRM
                                                Mesa Boot: OthelloDLion
.MR+140EF
CP continue . . . done
```

Figure 1. An example of the display.

Panel Windows

A panel is a rectangular array of tiles. Each tile displays the name of a register and its contents. A tile consists of three fields: in reading order these are flag, left, and right. A tilde, "~", in the flag means the data in the right field is invalid for some reason. An asterisk, "*", means the right field changed since the register was last read. When the cursor enters a panel, Burdock inverts the tile field occupied by the cursor.

Most panel operations take the type in as an argument. Burdock evaluates arithmetic expressions in the type in strictly from left to right using long (32 bit) cardinal arithmetic. No parentheses, no operator precedence, and no unary operations are allowed. Legal operators are addition (+), subtraction (-), multiplication (*), division (/), and remainder (\). The evaluator also recognizes space for addition.

To display a register, move the cursor to the left field of a tile and click red. Burdock evaluates the expression in the type in as the name of a register and displays the name and contents in the tile. To alter the contents of a register, move the cursor over the right field of a tile and click red. Burdock takes the type in, evaluates it, and puts the result in the register.

The other two mouse buttons are also useful. Clicking blue moves the text in the inverted field to the type in. Holding down yellow gets the panel menu.

To increment an address, move the cursor over the left field of a tile; hold down the left shift key and click the red mouse button. If the tile is not empty, Burdock increments the left field of the tile and displays the contents of that register in the right field. If the tile is empty, Burdock uses the type in rather than the tile. Decrementing works by clicking the blue button with the left shift key down.

Clicking shift-yellow over an empty left field clears the column below the tile. If the left field was not clear, Burdock fills the column below by successively incrementing the left field of the original tile. Burdock fills less than a column if it finds a number in the type in. In that case, Burdock only fills the number of tiles indicated. Clicking shift-yellow over a right field copies that right field down the column.

The user can alter the size of the panel using the standard Tajo mechanisms. When shrinking the window, any tiles which disappear are thrown away. When the window grows, all newly uncovered tiles are blank. Making the window tiny does not disturb any tile.

IOP

Raw numbers in the left field are interpreted as addresses in IOP main storage. The names Burdock uses for the 8085 internal registers are listed in the following table. The numbers in parentheses indicate the size, in bits, of the register.

A(8), B(8), C(8), D(8), Flags(8), E(8), H(8), L(8), PC(16), PCH(8), PCL(8), RIM(8), SP(16), SPH(8), SPL(8)

Notice that A, B, C, D, and E are all legal hex constants as well as internal 8085 registers. Burdock handles these as special cases and always displays the contents of the internal register in the right field of its tile. To examine main memory locations A through E, use a preceeding 0.

The top line of the panel provides a list of commands which execute with a click of the red mouse button. These commands are listed in the table below in the order they appear on the display.

Boot	boot the IOP kernel
Load	load the IOP
Start	use the selection in the IOP source and start the IOP
Stop	stop the IOP
Continue	continue the IOP
Step	execute a single instruction
StepFrom	use the selection in the IOP source and execute that instruction

The functions Boot, Start, Continue, Step, StepFrom, and Stop are also available from the menu.

Update is an additional function available from the menu. Update rereads registers and redisplays all tiles in the panel.

If communication breaks down between the IOP kernel and Burdock over the umbilical, any of the following error messages might appear in the communication window: LSAckStillOn, LSAckStillOff, or LSReqStillOn. If booting the IOP does not restore communication, hardware problems probably exist in either the IOP or the umbilical.

Source Level Debugging

All break points must be set and cleared through the IOP source window. Also, Start and StepFrom take their parameters as a text selection in the IOP source window. The assembler and binder both produce files used by the source level debugging features. The assembler derives the address map (.am) from the source (.asm). The address map holds the values of symbols defined in the source and keeps the correspondence between the text and relocatable binary. The binder produces a file map (.fm) and an absolute binary (.bin). The file map has the values of globally defined symbols and the correspondence between source files and absolute storage locations.

To change the file displayed in the IOP source (.asm), use the IOP Source line of the file window. Switching the source also switches the symbols. When using the panel, remember that symbols defined locally are accessible only when the proper source is loaded. (This, of course, does not apply to globally defined symbols.) Whenever switching to a source file not in the file map, Burdock insists on a red button click for confirmation. The symbols remain unchanged after an explicit confirmation. Clicking yellow aborts the switch.

To set a break point, move the cursor to the line where the break point is to go in the IOP source window. Get the Breakpoint menu and select Set. A grey underline will appear on the line where the break point will go. A subwindow will appear at the bottom of the source window. If you decide not to set the break point after all, bug Never Mind and all is forgiven. Before confirming by bugging Set, you can specify conditions and Trace Items. After setting the break point, Burdock highlights the line with a grey background.

To clear a break point use the cursor to point to the line and use Clear from the Breakpoints menu. Burdock clears the break point and removes the grey highlight from the display.

Beware of a few irregularities. Do not set break points on calls or try to step through a return instructions. Break points do not work when SP points into ROM or nonexistent memory.

CP

Raw numbers in the left field are interpreted as virtual addresses in CP main storage. The 48-bit microinstructions are displayed as ordered triples of sixteen-bit hex constants separated by commas. The names Burdock uses for the processor registers are listed in the following table. See the DMR for detailed descriptions of these registers. [Reference DMR]. The numbers in parentheses indicate the number of registers of that type and the word size. All numbers are in hex.

.cR (1000 x 30)	control store real address
.cV (1000 x 30)	control store virtual address
.EKErr (1 x 2)	emulator/kernel error register
.IB (3 x 8)	Mesa byte code buffer
.IBPtr (1 x 2)	IB buffer index

.IOOut (10 x 10)	(write only) device control control registers and output buffers
.IOXin (10 x 10)	(read only) device status registers and data input buffers
.Link (8 x 4)	microcode subroutine linkage registers
.Map (4000 x 10)	main storage page map
.MInt (1 x 1)	Mesa interrupt flag
.MR (30000 x 10)	main storage real address
.MV (400000 x 10)	main storage virtual address
.pc16 (1 x 1)	Mesa byte code pc extension selects byte in word
.Q (1 x 10)	internal 2901 register
.R (10 x 10)	high speed scratch pad
.RH (10 x 8)	main storage address extension (.RH+F used by the kernel)
.stackP (1 x 4)	Mesa evaluation stack pointer
.TC (8 x 4)	task dispatch/branch bits
.TPC (8 x C)	task program counters (.TPC+6 reserved by IOP, .TPC+7 used by kernel)
.U (100 x 10)	low speed scratch pad

The top line of the panel provides a list of commands which execute with the click of the red mouse button. These commands are listed in the table below in the order they appear on the display.

Boot	load and start the CP kernel, boots the IOP only if necessary
Load	load control store (.fb) and symbol table (.st)
Start	start the CP, use the type in as the starting address
Stop	stop the CP
Continue	continue the CP
Break	use the type in as an address and set a break point
Unbreak	use the type in and clear the break point
LoadReal	load main storage from a .cpr format file

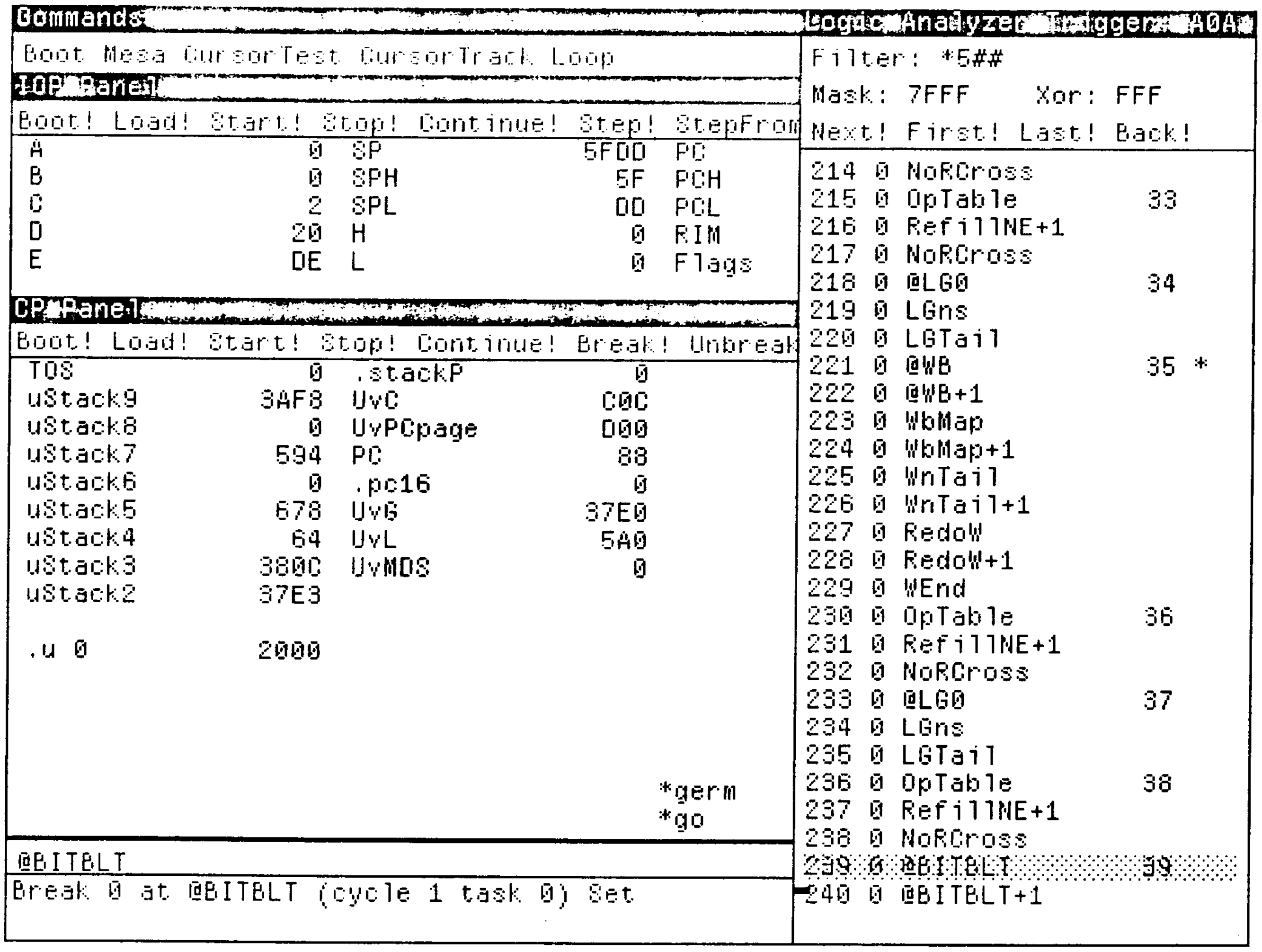
The functions available from the menu are listed in the following table.

Resume	continues both the IOP and CP
Unbreak *	clears the last break point hit by the CP
List breaks	lists, in the Exec window, all the break points currently set, indicates the last break point hit by the CP with a *
Alternate	display an alternate representation of the tile field in the type in, uses the tile field selected when the yellow button was pushed
ClrPanel	clears all tiles in the panel
ClrBreaks	clears all break points
Reset	disables all tasks, except the IOP, and sets TPCs to their initial values
Update	rereads the contents of registers and redisplays all tiles in the panel

The Logic Analyzer

The logic analyzer window displays and formats data captured by the Tektronix 7D01 logic analyzer. The window is divided into two parts: the top part contains parameters used to format and display the data. The bottom part of the window contains the data captured by the logic analyzer. This data is standard read only Tajo text; scrolling, thumbing, and selecting text all work. To copy logic analyzer data, use the Tajo text selection and stuffing mechanisms.

When the analyzer is not triggered, the data display is blank and the name frame says "Untriggered". When the analyzer triggers, Burdock displays the data in its window, highlighting the trigger line with a grey background, and writes the raw trigger pattern in the name frame. Even though the 7D01 allows different word sizes, Burdock formats only sixteen bit words. Any other setting of the word size switch on the 7D01 provokes protocol errors. Burdock indicates this by putting "Protocol Error" in the name frame.



```
Commands
Boot Mesa CursorTest CursorTrack Loop
IOP Panel
Boot! Load! Start! Stop! Continue! Step! StepFrom
 A                 0  SP            5FDD  PC
 B                 0  SPH             5F  PCH
 C                 2  SPL             DD  PCL
 D                20  H                0  RIM
 E                DE  L                0  Flags
CP Panel
Boot! Load! Start! Stop! Continue! Break! Unbreak
 TOS               0  .stackP          0
 uStack9        3AF8  UvC            C0C
 uStack8           0  UvPCpage       D00
 uStack7         594  PC              88
 uStack6           0  .pc16            0
 uStack5         678  UvG           37E0
 uStack4          64  UvL            5A0
 uStack3        380C  UvMDS            0
 uStack2        37E3

 .u 0           2000



                                     *germ
                                     *go
@BITBLT
Break 0 at @BITBLT (cycle 1 task 0) Set
```

```
Logic Analyzer Trigger: A0A
Filter: *5##
Mask: 7FFF    Xor: FFF
Next! First! Last! Back!
214 0 NoRCross
215 0 OpTable          33
216 0 RefillNE+1
217 0 NoRCross
218 0 @LG0             34
219 0 LGns
220 0 LGTail
221 0 @WB              35 *
222 0 @WB+1
223 0 WbMap
224 0 WbMap+1
225 0 WnTail
226 0 WnTail+1
227 0 RedoW
228 0 RedoW+1
229 0 WEnd
230 0 OpTable          36
231 0 RefillNE+1
232 0 NoRCross
233 0 @LG0             37
234 0 LGns
235 0 LGTail
236 0 OpTable          38
237 0 RefillNE+1
238 0 NoRCross
239 0 @BITBLT          39
240 0 @BITBLT+1
```

Figure 2. The logic analyzer in NIA mode.

Each line of the data display has two fields followed by two optional fields. The first field is the line number of the display in decimal. The second field is the formatted data. The first optional field is the match number in decimal (see the explanation of filtering below). An asterisk at the end of the line indicates the current match.

Items on the logic analyzer menu are listed in the following table.

Octal	display data in octal
Hex	display data in hex
Decimal	display data in decimal
Binary	display data in binary
NIA	(the default) disassemble the low order twelve bits as symbolic CP control store addresses, radix is hex, display the other four bits as a hex digit (see Figure 2)
Test	toggle the logic analyzer test mode (only for wizards)
Read	read the analyzer and display the data
Reset	reset the logic analyzer trigger, clear the data

Burdock uses the Mask and Xor parameters as hex numbers to mask and invert bits on each line of the display. Burdock first does the xor then ands the mask with the data before displaying. The default values (7FFF for Mask and FFF for Xor) were chosen for display of CP control store addresses. Typing a CR when changing the Mask or the Xor redisplays the data.

Burdock provides facilities to search for patterns in the words of data. The pattern goes in the parameter labeled Filter. With Filter selected, typing CR builds a new list of matches and redisplays the data. All matches are numbered in decimal starting at 0. If a line matches the pattern, the match number appears at the end of the line. An asterisk after the match number indicates the current match.

Two characters in a pattern have special meaning. An asterisk (*) matches any substring including the null subtring. A sharp (#) matches any single character. When searching the data, Burdock first converts the pattern to a string in the current radix. If the pattern is not a legal number with wild cards (asterisks and sharps) Burdock tries to evaluate the string as an arithmetic expression using the CP symbol table. If the pattern proves illegal, the display blinks and Burdock gives up. To explicitly overide the current radix when the pattern is a number with wild cards, append any of the following characters to the pattern: exclamation point (!) for binary, single quote (') for octal, decimal point (.) for decimal, or X (either upper or lower case) for hex.

After computing the pattern, Burdock converts each word of the data to a character string and tests it against the pattern. If the word matches, Burdock adds it to the match list. After seaching through all the data, Burdock sets the current match to the first match in the list and makes it visible in the display. The functions Next, First, Last, and Back move the current match through the match list. If any of these functions move the current match off the display, Burdock redisplays so that the current match is one quarter of the way from the top. Last moves to the last match in the list. First moves to the first match. Next moves to the next match. Back moves back one match in the list.

The Ether Kludge

Burdock can act as an ethernet controller for the Dandelion. When the Dandelion requests ethernet service, Burdock enters ether mode by turning off the display and setting the ether mode cursor. The cursor inverts for each requested StartIO. While in ether mode, Burdock ignores all keyboard and mouse input except DEL. Typing DEL stops the IOP, turns the display back on, and forces Burdock back to command level.

Command Files

When Burdock starts, it will take a command file name from the command line. To execute a command file after Burdock has already started, select the "Command" parameter in the files window and type the file name ending with a CR. (In both cases, Burdock assumes ".burdock" for a file name extension). Execution begins immediately after the CR. To stop a command file while it executes, type DEL. This gets Burdock listening to keyboard commands again; however, you cannot continue your command file. Burdock ignores all other keyboard and mouse input while a command file runs.

Command files contain three types of statements: function calls, tile definition, and conditionals. All statements must be separated by semicolons. Any statement may begin with one or more labels; a label consists of an identifier followed by a colon (:). Spaces and carriage returns are ignored. Matching vertical bars (|) delimit comments. Comments may be any length and span more than one line.

A tile definition statement fills a panel tile and has one of two forms (optional fields are in bold face):

TileName = *PanelLoc* ! ***LeftField*** ← ***RightField***
or
TileLoc ! *LeftField* ← ***RightField.***

A *TileName* is simply an identifier used to name a tile. *TileNames* are useful in conditionals. *PanelLocs* are specified by an ordered pair of hex constants indicating row and column. IOP[0, 0] is the upper left tile of the IOP panel. CP[11, 2] indicates the lower right hand tile of the initial CP panel. A *TileLoc* is either a *TileName* or a *PanelLoc*. *LeftField* and *RightField* are the initial contents of the tile. They can be any strings that do not contain the characters left arrow (←), colon (:), semicolon (;), quote ("), equal (=), sharp (#), greater than (>), less than (<), or open bracket ([). The left arrow (←) and *RightField* go together and are optional.

A conditional has the following form:

IF *Operand Relational Operand* THEN {*statements*}.

The *Relationals* include less than (<), less than or equal (<=), equal (=), not equal (#), greater than (>), and greater than or equal (>=). An *Operand* is either a hex constant or a *TileName*. Burdock evaluates the relation using integer (signed) sixteen bit arithmetic. If false the conditional skips the stuff between the matching braces. If true Burdock starts executing *statements*. Conditionals can be nested to any level.

A function call consists of the function name followed by its parameters in square brackets. A function with no parameters still requires square brackets after the function name. Use a comma (,) to separate the parameters. Any parameter which has characters that might confuse the scanner must be enclosed by double quotes (").

Command file functions (arguments in bold face are optional).

AlterAddress[*TileName*, *string*]	appends *string* to the left half of the tile and redisplays tile
Boot[]	boots the CP kernel
BootIOP[]	boots the IOP kernel, displays internal registers in panel
BootMesa[*string*]	loads CP main storage from a .boot format file
Break[*string*]	sets CP break point at specified location

BreakIOP[*string*]	sets IOP break point at specified location
CP[*number, number*]	CP tile location row and column
ClearCPPanel[]	clears the CP panel
ClearIOPPanel[]	clears the IOP panel
Close[]	close the log file
Continue[***timeOut***]	continues CP, time out is in milliseconds
ContinueIOP[***timeOut***]	continues IOP, time out is in milliseconds
GoTo[*label*]	goes to label in command file
IOP[*number, number*]	IOP tile location row and column
Load[*string*]	loads CP control store
LoadIOP[*string*]	loads IOP storage, might switch IOP source
LoadReal[*string*]	loads CP main storage from .cpr format file
NoSymbolsLoad[*string*]	load CP control store, no symbols are loaded
OpenToAppend[*string*]	opens log file, appends to the end of the file
OpenToWrite[*string*]	opens log file, previous contents of file are lost
Pick[*TileName*]	replaces the type in by the right field of tile
PickAppend[*TileName*]	appends the right field of tile to the type in
Start[*string*, ***timeOut***]	starts the IOP at *string*, time out is in milliseconds
StartIOP[*string*, ***timeOut***]	starts the IOP at *string*, time out is in milliseconds
Stop[]	stops the CP
StopIOP[]	stops the IOP
StuffAddress[*TileName*]	stuffs type in into left field of tile
StuffRegister[*TileName*]	stuffs type in into right field of tile
TypeIn[*string*]	replaces the type in by *string*
TypeInAppend[*string*]	appends *string* to the type in
Unbreak[*string*]	clears specified CP break point
UnbreakIOP[*string*]	clears specified IOP break point
Write[*string*]	write *string* into log file
WriteLine[*string*]	writes *string* followed by a CR into log file
WriteLoc[*TileName*]	writes right half of tile into the log file
WriteTime[]	writes time into the log file

All time outs are specfied in milliseconds. If a function times out, the command file stops immediately and Burdock returns to keyboard command level. A function with a time out of zero can never time out. Omitting a time out altogether means the function returns immediately and command file processing continues.

Consider the following example of a command file.

```
R= CP[0, 0]! .r; |define symbolic panel locations|
CP[0, 1]! .u+9;  |this form works too!|
U= CP[1, 0];
M= CP[2, 0];
X= CP[3, 0];
TypeIn["1"]; StuffRegister[R];  |puts a 1 .r|
AlterAddress[R, "+1"]; |display .r+1 in R|
AlterAddress[R, "-1"]; |display .r in R|
Pick[R]; StuffAddress[M]; |display virtual memory location 1 at M|
TypeIn[".tpc+"]; PickAppend[R]; StuffAddress[X]; |show .tpc+1 at
X|
TypeInAppend["+4"]; StuffAddress[X]; |display .tpc+5 at X|
```

TypeIn takes a single argument and puts that into the type in. TypeInAppend appends to rather than replaces the type in.

StuffRegister takes a single argument, a symbolic panel location, and puts the type in into the right field of the tile.

AlterAddress takes two arguments, a symbolic panel location and a string. Burdock takes the left field of the tile appends the string argument and puts the result back into the left field.

Pick takes a symbolic panel location and puts the contents of the right field into the type in. PickAppend works in a similar fashion, but appends rather than replaces the type in.

StuffAddress takes a panel location and puts the type in into the left field.

The Command Window

CommandWindow provides facilities for running any one of an array of Burdock command files at the click of a mouse button. The window itself contains names separated by spaces. To run a comand file, use the cursor to point at a name and click red. (Click only once!) Burdock takes the name, assumes an extension of .burdock, opens that file, and executes it.

You must start CommandWindow in a running Burdock using the Create-Tool command available from the background menu. The initial text in the command window comes from the commands line of the Burdock section of user.cm. (See the section on practical matters for more details.) To alter the contents of the window, set the insertion point, and type. Use BS and BW for editing; no other editing commands are available.

The tool is available from [Iris]<Workstation>Jarvis>CommandWindow.bcd.

Practical Matters, or Making Burdock Work on Your Disk

To get Burdock up and running automatically, retrieve [Iris]<Workstation>Jarvis>Burdock60.cm on to your disk and execute it as an Alto executive command file. This gets Burdock running on your Alto. Arrange the windows in a pleasing manner and make a checkpoint. The rest of this section explains how Burdock60.cm works: if you are not interested, skip it.

Before Burdock can function, it requires that several files be on your disk; these are listed in the following table.

RunMesa.run	loads and starts Mesa.image
Mesa.image	the Mesa runtime environment
MesaFont.strike	display font definition used by Mesa and Tajo
SmallAltoTajo.bcd	the tools environment
Burdock.bcd	Burdock
8085asm.ops	definition of the 8085 symbols
Kernel.fb	the CP kernel
Burdock.overlays	more of the CP kernel

Burdock gets several hints from user.cm. The example below can be found in [Iris]<Workstation>Jarvis>Burdock.cmSlice.

```
[TOOLS]
Bitmap: [x: 32, y: 112, w: 544, h: 591]

[Burdock]
Commands: Boot Mesa CursorTest CursorTrack
Symbols: Swatee
```

The larger the display bitmap, the less room Burdock has for symbol tables and code. Conversely, working with a cramped bitmap can be difficult. The bitmap above seems to be a good compromise. (The size of the bitmap is not as important for Altos with more than 128K.) The commands window gets its initial contents from the commands line. The commands window reads through user.cm only at creation time; it does not look everytime Burdock runs. Burdock uses Burdock.scratchSym as a swapping store as it runs; to change this file, use the symbols line. Using Swatee instead of Burdock.scratchSym saves 50-60 pages on your disk.

To start a fresh Burdock, type the following command line from the Alto executive.

```
Mesa.image SmallAltoTajo Burdock/l <optional-command-file-name>
```

This starts Burdock running and executes the command file (if specified). To include other tools, insert their names before the Burdock/l.

last edit by Jarvis February 2, 1981 12:41 PM

Break! sets a break at the current selection.

(IOP)Stack! displays the top ClumpSize words on the stack.

(IOP)Instructions! Takes the current selection, and prints out ClumpSize instructions.

Panels:

Note that there is a separate TypeIn for each panel.

If the TypeIn is empty, the current selection is used.

Mouse operations (possibly LeftShifted) work as in Alto Burdock. (I think.)
  Point (left mouse button) stores TypeIn into the blackened tile slot.
  Adjust appends the current tile slot to TypeIn.
  LeftShift+Point bumps the contents of the LH by +1.
  LeftShift+Adjust bumps the contents of the LH by -1.
  LeftShift+Point+Adjust (Menu) fills the column by adding +1 each step.

Builtin symbols:

IOP registers: B, C, D, E, H, L, M, A, PCH, PCL, SPH, SPL, Flags, RIM.
Register pairs: BC, DE, HL, SP, PSW, PC.

If an expression (in the LH of a tile in the IOP panel) for an address in IOP memory begins with a $ the address will be displayed in word mode. (The contents of register pairs are automatically displayed in word mode.) A % displays an instruction.

CP: .CR (control store real), .CV (control store virtural), .RH, .R, .U, .TPC, .TC, .LINK, .MAP, .MR (main memory real), .MV, .IOIN, .IOOUT, .Q, .IB, .PC16, .STACKP, .IBPTR, .MINT, .EKERR, .UI.

Bugs:

It will probably crash if you get to a breakpoint in the wrong cycle. (This hasn't happened to me yet.)

Burdock automatically appends the "right" extension when it goes to read a file. This is a pain for loading the germ because it wants PilotDLion.cpr rather than PilotDLion.germRM. You will have to edit your Germ.burdock and/or rename/copy your PilotDLion.germRM.

Occasionally a tile goes into hyperspace. The symptom is that the RH is blank, and doesn't get fixed up on repaint. To recover the screen area, delete that tile (store empty into the LH) and start over again.

See also Burdock.restrictions.

// Burdock.restrictions, HGM, 18-Oct-82 13:35:08

Pending branch/disps are lost on c2 or c3 breaks. (Store them into UKSaveDisp by hand if you know what's going on and/or you are desperate.)

c2 breaks on write clicks won't write anything.

c2 or c3 breaks on read clicks will get trash from ←MD when you continue.

You can't put breaks on c*, or on instructions without symbols.

You can't put breaks on instructions used by more than one task.

You can't set a TPC to anything other than an instruction in c1.

c2 or c3 breaks in IO tasks may not take when expected since the task has to run one more click to get to the kernel. (If the current click turns off wakeups so it doesn't get scheduled again....)

Breaks after IBDisp won't work.

Breaks during bank switching will get confused.

Continue after breaking after AltUAddr won't work.

You can't write into a task's TC bits.

You can't read the 2 (MagTape) input registers that are located in the "output" clump of the function fields.

Things will get all confused if you get 2 breaks at the same time. This can happen if the first break in in c2 or c3 since the EtnerKernel doesn't happen until the next click.