January 1974

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

WORKING PAPER #59

GT40 Utility Programs and the LISP Display Slave

by

Michael Beeler, Joseph D. Cohen, John L. White

Abstract

This memo describes two GT40 programs: URUG, an octal micro-debugger; and VT07, a Datapoint simulator and general display package. There is also a description of the MITAI LISP display slave, and how it uses VT07 as a remote graphics slave.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contact number N00014-70-A-0362-0005.

Contents

Introduction

This memo describes two general utility programs for the GT40: URUG, Michael Beeler's micro-debugger, and VT07, a two part program consisting of a datapoint simulator, and a general graphics package, written by Pitts Jarvis and Joe Cohen.

URUG is a small (1000 byte) octal debugger with limited breakpoint facility, useful for debugging programs in PDP11's with little spare core.

VT07 is used to make the GT40 into an alphanumerics terminal and a graphics display. VT07 was designed to meet the needs of higher level languages like LISP. It draws heavily on our experience with LOGO, where much thought has been given to manipulation of displays within a general high level language. This experience led to the incorporation of advanced features, such as display list subroutining. On the MIT AI system, the GT40 is used graphically by the DRAW program and by LISP.

This memo discusses all the GT40 programs (URUG, ROM Loader, VT07) in detail, and briefly outlines the use of VT07 by LISP.

MICRORUG USER'S MANUAL

M. Beeler, August 1973

Microrug is an octal debugger for the DEC GT-40 computer/display. The user should be familiar with the PDP-11/05, which is the minicomputer in the GT-40.

Microrug (URUG for short) occupies 1000 (octal) consecutive bytes of memory, and uses the keyboard for input and the CRT display for output. The information displayed is in four lines. New information appears on the bottom line, and when that line is completed, the lines are scrolled upward; the previous top line is lost, and the new bottom line is blank. To help the user tell which line is which, a dot is displayed at the left of the screen between the top and second lines.

URUG is used for three different functions: examining (and depositing into) memory locations, setting breakpoints, and starting up programs. There are two versions of URUG, one which uses the JMP instruction for breakpoints and one which uses BPT. They are similar, so the JMP version is described below, and then differences in the BPT version are noted.

To examine a location, type its address in octal, and a space. URUG types the current contents of that location. To change the contents, type the desired contents and a space. The new contents are deposited in the location, and the display scrolled to present a fresh line for more commands.

If no argument precedes the space, then nothing is deposited, and the display is scrolled as usual. (This also applies to typing a space as the first character on a fresh line; no action is taken except to scroll the display.)

If a mistake is made in typing a numeric argument (such as an address or new contents), typing rubout resets URUG as if none of the argument were typed.

If a linefeed is typed instead of a space, URUG takes the same action but then also types the address and contents of the location after the last location examined.

If you try to examine an odd-numbered location (i.e., the left byte of a word), URUG acts as if you typed an address one less.

URUG's starting address, BEG, is the first location in URUG, often

assembled to be 37000. When URUG is started, it saves general registers R0 through R6, and restores them when you ask it to start up a program. Thus, while URUG is running, 177700 through 177707 contain URUG's values and not those of the user's program. The user's general registers are referenced by typing R0 through R6 as addresses. Since the locations actually examined are inside URUG, linefeed acts strangely. Specifically, user register contents are stored in every other word, so two linefeeds after R0 is the same as typing R1. Examining non-existent locations (such as above 37776) causes a timeout trap through location 4. To avoid losing URUG's copy of the user general registers, restart URUG at RESUME instead of BEG (that is, 40 bytes later).

A breakpoint consists of the instruction, JMP BEG, where BEG is the starting address of URUG. A breakpoint is set by typing B. If no argument is given, the breakpoint is placed at zero.

If a number precedes the B, that number is used as an address at which to set the breakpoint. Evenness of the B argument is not checked, so trying to set a breakpoint at an odd address causes an odd address error trap through location 4 (again, restart URUG at BEG+40).

When URUG is started (or restarted, at BEG+40), it checks whether any

breakpoint is set. If not, it merely scrolls the display and waits for commands. If so, it restores the contents of the breakpoint location (and following word), scrolls the display, and automatically examines the breakpoint location to let you know it's just hit and restored the breakpoint.

Typing B when the breakpoint (there is only one) is already set is the same as restarting URUG at BEG+40; it restores the breakpoint contents, etc.

Typing G, preceded by a number, restores the general registers to the user's values, and transfers to the argument of the G. G with no argument transfers to the location last G'ed. URUG's memory of what this location is may be examined and modified as R7.

The G command, like B, does not check for evenness of its argument. If you try to G to an odd address, the user values are already restored to the machine's general registers, so it does not matter whether you restart at BEG or BEG+40.

Two details of URUG are equivalent characters and typing too much. Because of its simple type-in routines, carriage return and tab, as well as various other control characters, have the same effect as

space. Although R, B, and G are echoed as upper case, they must be typed as lower case. Except for these characters, and the ten digits 0 - 9, all characters = rubout.

If more than 24 (decimal) characters are typed, by URUG and/or the user, before a scrolling occurs, then display commands will be overwritten and URUG will probably have to be reloaded to be useful. To be exact, URUG could still be used to replace the clobbered display instructions at the very top of itself (as long as the garbaged commands aren't making the PDP-11 trap). 22 characters will cause the next command line to possibly be run-on, but this is temporary and easily ignored.

If the user wants to resume his own display program before starting up his PDP-11 program, the following sequence is suggested:

```
R7      xxxxxx  USRBEG  (set G address)
172000  xxxxxx  USRDPC  (start user's display program)
G                       (start user's PDP-11 program)
```

BPT VERSION -- DIFFERENCES FROM JMP VERSION

The location RESUME, at which to start after time-out and odd address traps, is BEG+34 instead of BEG+40.

A breakpoint in this version consists of the instruction, BPT.

It requires that:

(1) the user must have set up an appropriate BPT trap vector, namely:
    14 contains BEG+162 (=HITBRK), URUG's breakpoint handler
    16 contains 340, to set CPU priority to 7

(2) the user must have set up an appropriate stack pointer in R6, such that the BPT can push its 2 more words without trapping

(3) R6 must be so set up before any G command, since G'ing is done by pushing PS, PC, and executing an RTI

It has the advantages that:

(1) the breakpoint occupies only one word

(2) the BPT trap vector automatically protects URUG from interrupts

(3) the N, Z, V and C condition codes are preserved through a breakpoint-and-proceed sequence

(4) the user's processor status may be examined and modified

(5) URUG code saved allows automatic display restarting (see below)

In the BPT version, the saved PS may be examined and modified as R8. While URUG is running, the user's PC and PS are popped off the stack (into pseudo-R7 and R8), so examining R6 yields the user's R6 before

the BPT, and this may be modified without losing the PC or PS. URUG does not use a stack, except for the breakpoint handler and setting up the RTI for the G command. (The JMP version of URUG never uses a stack.)

The BPT version's G command also restarts the display before it starts the user's program. The address loaded into the Display Program Counter may be examined and modified as R9. It is initially that of URUG's own display program.

Some thought has been given to making URUG position-independent. It seems unlikely. Instead, it is suggested that you re-assemble it for whatever starting address you want. It has assemble-time teletype input to specify JMP or BPT version, as well as starting address. A starting address not ending in zero may cause it to use a few more than 1000 bytes.

GT-40 BOOTSTRAP LOADER ("version S09, release R01")

The main point of this discussion is explaining what core locations the loader clobbers.

The loader operates in two modes. First, it echoes like a teletype, sending each character from the keyboard to the PDP-10, and displaying

each character from the PDP-10 on the screen.

The loader (in both modes) starts its stack at 15770 and works down. The echoing mode uses 2+2n stack words, where n is the number of keyboard characters typed which have not yet been sent to the PDP-10. Thus it uses at least 4 words, and if typing is not rapid, only 4. This is fewer words than used in the loading mode, as discussed below.

The echoing mode also uses several locations in low core, mainly for display words, but also for a power-fail vector. Each character from the PDP-10 is stored in a separate word with left byte = 0, starting at location 32. Thus, after receiving c characters from the PDP-10, the following locations in low core will be clobbered:

```
  0 <= 160000       ;DISPLAY JUMP
  2 <= 166756       ;TO "DISPRG" IN LOADER

 24 <= 166010       ;POWER FAIL PC = START+10
 26 <= 0            ;POWER FAIL STATUS
 30 <= 0            ;START OF DISPLAY LIST
 32 <= char1        ;FIRST CHARACTER
 34 <= char2        ;SECOND CHARACTER
etc.
```

```
30+2c <= char c        ;c-th CHARACTER
32+2c <= 160000        ;DISPLAY JUMP
34+2c <= 0             ;TO 0
36+2c <= 160000        ;DISPLAY JUMP
40+2c <= 0             ;TO 0
```

42+2c and following are not clobbered

The loading mode uses 8 stack words. The stack pointer is initialized to 15770 instead of 15770+2, so the first location on the stack, at 15770, is not clobbered. The next lower 8 locations, 15750 through 15766 inclusive, are clobbered during loading. Trying to load into these locations will usually cause random data to be loaded, or else transfer to random locations when a loader RTS PC pops loaded data into the PC.

Both the echoing and the loading modes use three particular locations as I/O character buffers:

15772 (P10IC) is clobbered by characters read

15774 is also clobbered by characters read

15776 (P10OC) is used for characters to be sent to the PDP-10, and will contain 0 after loading

When the loader switches from echoing to loading mode, it turns off

the display and no longer references low core locations (except to load data into them). Thus, all the display words and the power-fail vector can be overwritten by the loading mode. The difficulty is if one wishes to merge-load two or more programs. If one of the programs before the last one occupies these low core locations, then restarting the bootstrap will clobber it. There are two simple alternatives; both involve telling the PDP-10 all the programs you want merged, before any are loaded. The PDP-10 can then either:

(1) concatenate all the programs and not give the "end of load" signal until all are loaded; or

(2) at the end of each program, give the "start up program" signal with address 166520 ("LOAD"), which simply restarts the loading mode. After the last program it can give the "end of load" signal.

[Note on a GT-40 (PDP-11/05) quirk: 177700 through 177707 are R0 through R7 as far as the examine and deposit switches are concerned, but MOV Rm,@#17770n always acts like n = 0 (i.e., deposits in R0 = 177700). The data also shows up in 177710, apparently a CPU temporary register.]

n = 0/1 = word/byte

ee = offset, 0 - 77

ff = offset, 0 - 377

0	HALT	1000ff	BPL
1	WAIT	1004ff	BMI
2	RTI	1010ff	BHI
3	BPT	1014ff	BLOS
4	IOT	1020ff	BVC
5	RESET	1024ff	BVS
6	RTT	1030ff	BCC = BHIS
7 - 77	UUO	1034ff	BCS = BLO
1DD	JMP		
20R	RTS		
210 - 227	UUO		
23m	SPL		
240 - 277	CC's: 20 0/1 = clr/set		
3DD	SWAB	10 N	
4ff	BR	4 Z	
10ff	BNE	2 V	
14ff	BEQ	1 C	
20ff	BGE		
24ff	BLT	104000	
30ff	BGT	to	EMT
34ff	BLE	104377	
4RDD	JSR		
n050DD	CLR(B)	104400	

n051DD	COM(B)		to	TRAP
n052DD	INC(B)	104777		
n053DD	DEC(B)			
n054DD	NEG(B)			
n055DD	ADC(B)			
n056DD	SBC(B)			
n057DD	TST(B)			
n060DD	ROR(B)			
n061DD	ROL(B)			
n062DD	ASR(B)			
n063DD	ASL(B)			
64mm	MARK	106400 - 106477		UUO
65SS	MFPI	1065SS	MFPD	
66DD	MTPI	1066DD	MTPD	
67DD	SXT	106700 - 107777		UUO
7000 - 7777	UUO			
n1SSDD	MOV(B)			
n2SSDD	CMP(B)			
n3SSDD	BIT(B)			
n4SSDD	BIC(B)			
n5SSDD	BIS(B)			
6SSDD	ADD	16SSDD	SUB	
70RSS	MUL			

71RSS DIV

72RSS ASH

73RSS ASHC 17xxxx FPP's

74RDD XOR

75000 - 76777 UUO

77Ree SOB

VT07

VT07 can be thought of as two programs in one: a Datapoint simulator and a general purpose display package. The strange name "VT07" is derived from DEC's alphanumeric display terminal, "VT05". (The etymology of that name is unknown to us.) Our Datapoint simulator and simple graphics package was called VT06. The Datapoint simulator and complicated graphics package is called VT07.

Datapoint

The Datapoint simulator works just like a Datapoint, except:
the GT40 has both upper and lower case character display and keyboard (the AI Lab's only such display);
screen size is 73. characters wide by 32. characters high;
various bits of the console switch register add different non-datapoint options when on:

bit 0 -- don't blink cursor;

bit 1 -- italicize all characters;

bit 2 -- insert backspaces (10) into the display list, instead of moving the cursor back. This allows overstriking of characters. When in this mode, VT07 will space forward one column (not character) when it gets a move cursor forward character (30); a move cursor back (31) moves back one character (not column); and line feed and move cursor up (32) move to the same character position (not column) in the adjacent line.

VT07 does not need padding, (in fact it ignores rubout (177) the normal Datapoint padding character); and most important, ^P (20) is an escape character indicating that the next character is to be interpreted as a command by the graphics package.

General Display Package

In this section, character means a teletype character; word means a PDP-11 16. bit word; and byte means a PDP-11 8. bit byte. Characters are transmitted as characters. Words are transmitted as three characters: low order 6 bits right justified; next 4 bits right justified; and the high order 6 bits right justified. Bytes are only transmitted as part of words.

The first character after the escape character (^P) is a command to the graphics package. The length and arguments to commands are determined by each command. The only part of a command transmitted as a character is the command itself. Everything else is words.

The commands are:

0 A command further decoded by the next character.

This command exists to make VT07 compatible with existing display proram software. If the next character is:

1 Load Item command identical with 1 below.

2 Delete Item command with the following words being: size of the command, item numbers to be deleted, and a checksum.

1 Load Item. Followed by: a count of words from the count word, to (but excluding) the checksum; item number; words in the display item; and a checksum. If there is already an item with the same number, it will be deleted. VT07 expects the first three words in your list to be: a set graphics mode to set point; the x coordinate; and the y coordinate. The item is initially shown on the screen.

2 Delete Item. Followed by item number to be deleted. Expunges the item from the GT40.

3 Reset. Restarts the GT40. The GT40 jumps to its starting address, and sets everything up all over again. It takes a finite amount of time before the GT40 can accept another character after this command, so wait a while (a thirtieth of a second?) before sending more goodies. To just wipe out all display items in the GT40, treat it like a Datapoint, and send a home up (35), followed by a clear to end

of file (37).

4 Turn On Item. Followed by the item number. Puts the item on the screen.

5 Turn Off Item. Followed by item number. Takes the item off the screen.

6 Copy Item. Followed by item number of original, item number of the copy. Creates an identical copy with the new item number.

7 Move Item. Followed by new x coordinate word, new y coordinate word. This clobbers the x and y coordinates into the second and third word of your original list. The name of this command implies that the whole item moves in response to the command. Of course, this is only the case if there are no set point commands in the item (other than the expected initial three words).

8 Change Mode of Item. Followed by item number, new mode word. This puts the low order 11. bits of the new mode word into the low order 11. bits of the first word of the item (which is assumed to be a set point command.) By disabling intensity, light pen interrupts, blinking, and

line type, everywhere in the item except the first word, one can control these modes for the entire item with this command.

9 Add to Item. Followed by: size of command (see Load Item), item number, additions, checksum.

10. Subroutinize Item. Followed by: number of calling item, number of subroutine item. The subroutinized item will be inserted into the calling item, in the manner of a LOGO snap (not a LISP link). I.E., an item can be called as a subroutine many times by one list and by more than one list. This subroutining is limited only by the size of the display pushdown stack (currently about 10. deep). (This is done through a simulated display pushdown instruction in VT07.)

If the inferior has no set point display commands (other than the expected first three words), and has no modes enabled (see Change Mode) the effect will be to add a copy of the inferior to the end of the superior, with the subroutine acquiring the intensity, blinking, and other mode atributes of the superior.

Whenever the inferior is changed, (by means of the Add to Item command), it will change every place it is called as a subroutine.

11. Unsubroutinize Item. Followed by calling item number, subroutine item number. The first call to the subroutine from the calling item is deleted.

Error Codes

VT07 sends error messages back to the VT07 sends error messages to indicate internal error conditions. The error characters are:

0 -- Free storage full; and

34 -- other internal error.

VT07 does no handshaking on these errors, and does not abort a command in which they occurred, so the entire command must be sent.

LISP Slave System In order to use the GT40 as a LISP display slave, the initial DISINI, (which seizes the slave,) should be given two arguments, the second of which is the quoted ITS device of the GT40. For instance,

(DISINI 0 'T34)

will try to grab the GT40 on T34. The GT40 has less display space than the PDP6 or PDP10, and is probably less efficient in size for most display lists. The GT40 slave's advantage is that ITS time is not used to maintain the display, and that it makes several displays available.

The GT40 slave works exactly like the 6 and 10 slaves, with exceptions noted in the next two sections: "Discrepancies" describes real, permanent differences in the slaves; "Lies, All Lies" describes temporary, socialist realism type differences which will shortly disappear.

The rest of this section is a description of some of the internal workings and structure of the GT40 slave, which can be ignored by anyone not interested in

delving in its guts. In order to understand the rest of the differences beetween 340 and GT40 pictures, it's necessary to have some idea of how 340 and GT40 lists are created, stored, and modified.

The LISP GT40 slave is the same program as the 10 slave. If the initial DISINI has a zero or no second argument, the program tries to sieze and use the 340, otherwise, it doesn't try to grab the 340, and instead sends commands to the GT40 through the specified ITS device. GT40 display lists are created and amended at the same time and place as 340 display lists. Those 340 display lists sit around in core, but are never displayed. This explains why DISGORGE works for the GT40, (those 340 lists are already there); and why DISGOBBLEs can't be displayed (the slave doesn't have the opportunity to create GT40 lists corresponding to the 340 lists being gobbled from disk).

VT07 does not do compacting garbage collects on its display storage space. This means that large items will cause display memory to become full, where many smaller items of the same total size would not. VT07

uses a modification of Knuth's "first-fit method" with doubly linked liberation with boundary tags. (Donald E. Knuth, The Art of Computer Programming, Volume 1, Fundamental Algorithms, (Reading, Mass., 1968).) Knuth observes, (page 447), as a result of some experiments, that if M is the total number of memory locations available,

> "The memory was able to become over 90% filled when the block size was small compared to M, but when the block sizes were allowed to exceed 1/3M (as well as taking on much smaller values) the memory tended to become "full" when less than 1/2M locations were in fact needed. Empirical evidence strongly suggests that, block sizes larger than 1/10M should not be used with dynamic storage allocation if effective operation is expected."

(Currently, free storage starts at about 12000 and runs to the top of core, making M=26000. Each item incurs an overhead of 24 bytes in a block seperate from the display list.)

340 lists have point mode words scattered throughout. When one does a DISLOCATE the slave actually grovels through the entire display item, searches out point mode words, and modifies them. Because of core limitations, it was decided that VT07 wouldn't have this capability. Programs using VT07 compensate by not storing any point mode words in their display lists. There's only one such word at the head of the list, and everything else is relative to previous display instructions. Thus when the GT40 does a (DISLOCATE) it only zaps two words at the start of the display item.

Intensity level information, however, is stored by LISP throughout the list. This is so that different parts of the same item can have different intensities. There is no way that the intensity level of a list can be changed without going through the entire item and finding and changing each intensity word, which is why DISCHANGE doesn't work. (It is suggested that other programs which use VT07 store brightness information in only the first word, thus enabling the intensity of an entire item to be changed with the Change Mode

command.)

The GT40 has no scaling hardware, so it doesn't try to scale. It does however, have blinking hardware, and blink information is stored at the head of each list, and can be turned on and off by the Change Mode command (LISP DISBLINK).

Features of the GT40 and VT07 that LISP does not (presently) take advantage of are:
variable line types;
italics;
display list subroutining.

Discrepancies

The GT40 screen is a window on the lower 3/4 of the 340 screen, i.e., the GT40 is 2000 points wide by 1400 points high.

The LISP GT40 slave cannot:

display the results of a DISGOBBLE, (but it can DISGORGE);

scale characters or vectors. GT40 characters are a bit bigger (it is impossible to be exact about this because of bugs in the 340,) than 340 scale 1 characters. Also there is no vertical character mode on the GT40.

wraparound;

DISCHANGE.

Lies, All Lies

There aren't any.

Appendix 1 -- LISP Display Slave Commands

IN ORDER TO USE THE SLAVE, IT'S BEST TO HAVE AVAILABLE THE PDP6. THERE IS A VERSION THAT WILL ALSO RUN ON THE PDP10 UNDER ITS, AT SOME DEGRADATION IN PERFORMANCE (BOTH OF THE SLAVE AND ITS). THE PDP6 SHOULD BE IN THE RUNNING STATE, AND IF SIMPLY HITTING THE START SWITCH DOESN'T KEEP THE RUN LIGHT ON, DEPOSIT ZEROS INTO LOCATIONS 40 AND 41 AND START UP AT 40
THE REMAINDER OF THIS DESCRIPTION OF LISP FUNCTIONS FOR THE DISPLAY SLAVE USES THE FOLLOWING CONVENTIONS:

X, Y	ARE ASSUMED TO BE INTEGER ARGUMENTS TO LINE DRAWING, POINT INSERTING, AND OTHER SUCH FUNCTIONS
N	IS A FIXED-POINT NUMERICAL ARGUMENT DESCRIBED UNDER PARTICULAR FUNCTIONS
ITEM	IS ASSUMED TO BE THE NUMERICAL INDEX OF SOME DISPLAY SLAVE ITEM. IT IS A QUANTITY SUCH AS IS RETURNED BY DISCREATE.
BRITE	EACH ITEM HAS A BRIGHTNESS LEVEL ASSOCIATED WITH IT, RANGING BETWEEN 1 AND 8. DEFAULT VALUE = 8.
SCALE	EACH ITEM HAS A SCALE, OR MAGNIFICATION, FACTOR ASSOCIATED WITH IT, RANGEING BETWEEN 1 AND 4. DEFAULT, AND NORMAL, IS 1; 2 DOUBLES THE LENGTH OF DRAWN LINES AND TEXT, 3 QUADRUPLES, AND 4 MULTIPLIES

BY 8. TEXT LOOKS MUCH NICER IF IT IS DRAWN WITH A LITTLE MAGNIFICATION; GENERALLY 2 IS APPROPRIATE.

FLAG IS AN INDICATOR TELLING WHETHER A GIVEN ACTION IS TO BE DONE (ON NON-NIL) OR UNDONE

BSL IS EITHER NIL, IN WHICH CASE THERE IS NO CHANGE, OR IS A LIST LIKE (BRITE SCALE) INDICATING A SETTING OF LEVELS FOR A GIVEN ACTION

121. A WELL-KNOWN INTEGER, EASILY RECOGNIZED TO BE THE SQUARE OF THE FIFTH PRIME, BUT NOT SO EASILY SEEN AS SUCH WHEN EXPRESSED IN OCTAL AS 171 - THUS WE USE OCTAL NOTATION EXCEPT WHEN THE STRING OF DIGITS IS FOLLOWED BY A .

EACH ITEM HAS ASSOCIATED WITH IT VARIABLES DETERMINING THE BRIGHTNESS, SCALE, AND VISIBILITY OF POINT AND LINE INSERTION REQUESTS; LIKE THE LOGO TURTLE, WE THINK OF THE ITEM AS HAVING A PEN WHICH CAN BE "DOWN" SO THAT A LINE IS VISIBLE WHEN THE TURTLE IS REQUESTED TO GO FROM ONE PLACE TO ANOTHER, OR "UP" SO THAT NO MARK IS SEEN. FOR THE COMMANDS TO AFFECT BRIGHTNESS, SCALE, OR THE PENUP STATUS, 0 GENERALLY MEANS NO CHANGE. COMMANDS WHICH TAKE AN OPTIONAL BSL ARGUMENT - NAMELY DISAPOINT, DISCUSS, AND DISALINE - WILL TREAT IT AS A TEMPORARY SETTING FOR THESE VALUES, AND UPON EXIT WILL RESTORE THESE VARIABLES TO THEIR VALUES PRIOR TO THE CALL. SIMILARLY, THE OPTIONAL PENUP ARGUMENT TO DISALINE IS TREATED

AS TEMPORARY.

ARGUMENTS THAT ARE INTENDED TO SPECIFY LOCATIONS ON THE 340 SCREEN FOR THE FUNCTIONS DISALINE, DISAPOINT, AND DISCUSS, ARE INTERPRETED IN ONE OF FOUR WAYS DEPENDING ON THE SETTING OF THE SLAVE VARIABLE "ASTATE":

- 0 RELATIVE MODE - THE POINT SPECIFIED IS IN RELATION TO THE HOME OF THE ITEM ON WHICH THE COMMAND IS ACTING.
- 1 ABSOLUTE MODE - X AND Y ARE DIRECTLY INTERPRETED IN THE CO-ORDINATES OF THE 340 SCREEN, MOD 1024., WITH THE LOWER-LEFT CORNER BEING [0,0]
- 2 INCREMENTAL MODE - THE POINT SPECIFIED IS IN RELATION TO THE CURRENT POSITION OF THE PEN OF THE ITEM ON WHICH THE COMMAND IS ACTING.
- 3 POLAR MODE - LIKE INCREMENTAL, BUT THE ARGUMENTS, WHICH MUST BE FLOATING POINT, ARE CONSIDERED AS THE RADIUS AND ANGLE FOR A POLAR COORDINATE SYSTEM CENTERED ABOUT THE CURRENT PENPOSITION (WITH ZERO DEGREES BEING HORIZONTAL TO THE RIGHT).

TO EMPHASIZE THE ASTATE MAPPING OF THESE ARGUMENTS, WE WILL WRITE ASTATE[X,Y] TO MEAN THE POINT SPECIFIED BY X AND Y.

N.B.: FUNCTIONS LIKE DISCREATE, DISLOCATE, AND DISMOTION, WHICH PLACE FOR AN ITEM'S HOME IN SOME SPECIFIED LOCATION, ALWAYS INTERPRET THE SPECIFICATION IN ABSOLUTE MODE.

TYPICAL CALLS	FUNCTION TYPE	EXPLANATION
(DISCREATE X Y) (DISCREATE)	LSUBR	CREATE A DISPLAY ITEM WITH HOME AT [X,Y] ON THE 340 SCREEN. DEFAULT OPTION IS TO PLACE HOME AT [0,0] IF X AND Y NOT GIVEN. RETURNS ITEM NUMBER OF NEWLY CREATED ITEM.
(DISINI) (DISINI N) (DISINI N DEVICE)	LSUBR	SEIZE AND INITIALIZE SLAVE. IF USER ALREADY HAS SLAVE, THEN REINITIALIZE, AND SET ASTATE TO GIVEN ARGUMENT. ALWAYS RETURNS PREVIOUS VALUE OF ASTATE, BUT NO ARG GIVEN, OR ARG NOT AMONG 0,1,2,3 MAKES NO CHANGE IN ASTATE. INITIAL ASTATE = 0. IF SEIZING SLAVE AND SECOND ARG GIVEN, USE GT40 THROUGH ARG ITS DEVICE, EG, (DISINI 0 'T34)
(DISPLAY ITEM FLAG)	SUBR	ITEM ON OR OFF DISPLAY - I.E. MAKE VISIBLE ON SCREEN OR NOT. DISCREATE, DISCOPY, AND DISGOBBLE PLACE THEIR ITEMS ON DISPLAY, EVEN WHEN NULL.

WHEN OFF DISPLAY, THE ITEM IS STILL REMEMBERED BY THE SLAVE UNTIL FLUSHED

(DISFLUSH)	LSUBR	NO ARG GIVEN MEANS FLUSH WHOLE SLAVE
(DISFLUSH ITEM1 . . . ITEMN)		OTHERWISE SIMPLY KILL ITEMS.
(DISLOCATE ITEM X Y)	SUBR	MOVE ITEM'S HOME TO LOCATION [X,Y]
(DISBLINK ITEM FLAG)	SUBR	SELF EXPLANATORY
(DISCOPY ITEM)	SUBR	MAKE A COPY OF ITEM, AS A NEW ITEM WITH HOME AT SAME LOCATION. RETURN NEW ITEM NUMBER.
(DISMARK ITEM N)	SUBR	IF N=0, REMOVE MARKER FROM ITEM. IF N<0, INSERT STANDARD MARKER IF N> , USE ITEM WITH #N AS MARKER
(DISCRIBE ITEM)	SUBR	GET LIST OF (XHOME,YHOME,XPENPOS, YPENPOS,BRITE,SCALE,PENUP,MARKER) FROM ITEM
(DISCHANGE ITEM BRITE SCALE)		

	SUBR	BRITE AND SCALE ARE INCREMENTS TO BE ADDED TO THE PARTS OF ITEM
(DISLINK ITEM1 ITEM2 FLAG)		
	SUBR	LINK OR UNLINK ITEM1 TO ITEM2 ITEM2 IS THE "INFERIOR" OF ITEM1, AND WILL BE DISLOCATED, DCHANGED, DISBLINKED, AND DISPLAYED AS A SUBPART OF ITEM1 WHENEVER THESE OPERATIONS ARE PERFORMED ON ITEM1.
(DISLIST)	LSUBR	RETURN LIST OF ALL ITEMS ON DISPLAY
(DISLIST ITEM)		RETURN LIST OF ALL INFERIORS OF ITEM
(DISET ITEM N BSL)	SUBR	SETS THE DEFAULT VALUES FOR PENUP, BRIGHTNESS, AND SCALE PARAMETERS FOR THE ITEM. IF N IS -1, PUT PEN DOWN; IF +1, LIFT UP PEN; IF 0, LEAVE PEN ALONE. SET BRITE AND SCALE FROM BSL [FOR MEANING OF BSL, SEE CONVENTIONS DISCUSSED ABOVE] WHEN CREATED, THE ITEM'S DEFAULTS ARE: PEN IS DOWN, BRIGHTNESS IS 8., AND SCALE IS 1.

(DISALINE ITEM X Y)

(DISALINE ITEM X Y N)

(DISALINE ITEM X Y BSL)

(DISALINE ITEM X Y BSL N)

LSUBR — SET PENUP AND BSL AS INDICATED BY N AND BSL (SEE DISET ABOVE), THEN GO FROM CURRENT PEN POSITION TO ASTATE[X,Y], LEAVING A VISIBLE LINE ONLY IF THE PEN IS DOWN, AND THEN RESTORE THE PENUP AND BSL PARAMETERS

(DISAPOINT ITEM X Y)

(DISAPOINT ITEM X Y BSL)

LSUBR — DISPLAY A POINT AT ASTATE[X,Y]. DOES AFFECT ITEM'S PENUP OR BSL PARAMETERS

(DISCUSS ITEM X Y TEXT)

(DISCUSS ITEM X Y TEXT BSL)

LSUBR — THE CHARACTERS OF THE VALUE OF TEXT ARE INSERTED, AS IF PRINC'ED, INTO THE INTO ITEM BEGINNING AT POINT ASTATE[X,Y]. NO CHANGE IN ITEM'S

PENUP AND BSL PARAMETERS.

(DISMOTION ITEM X Y SPD)

SUBR CAUSES ITEM TO BE SLOWLY DISLOCATED SO THAT ITS HOME IS AT [X,Y]. IF EITHER X OR Y IS NEGATIVE THEN PLACES ITEM UNDER CONTROL OF SPACE WAR CONSOLE 1. THE BUTTON RETURNS CONTROL TO THE TTY. SPD IS AN INVERSE MEASURE OF THE SPEED AT WHICH THE ITEM WILL MOVE. SPD = 0 IS MAXIMUM. NOTE WELL: ALTHOUGH THE SPACE-WAR CONSOLE CONTROL WILL WORK FOR ANY DISPLAY ITEM, THE AUTOMATIC SLOW MOTION WILL CURRENTLY WORK ONLY FOR ITEMS CONSISTING SOLELY OF LINES DRAWN BY DISALINE.

(DISGORGE ITEM) SUBR CREATES A (GENSYM'D) LISP ARRAY AND FILLS IT WITH THE 340 CODE FROM ITEM.

(DISGOBBLE ARRAYNAM) SUBR TAKES THE ENTRIES OF THE LISP ARRAY ARRAYNAM AND CREATES A DISPLAY SLAVE

ITEM WITH THOSE ENTRIES.

EXAMPLES

A SUBROUTINE TO DRAW A LIGHT BOX WITH A MEDIUM POINT INSIDE IT AT THE CENTER OF THE SCREEN, RETURNING A DESCRIPTION OF THE SLAVE ITEM:

```
((LAMBDA (OASTATE B)
        (DISALINE B -100 -100 1)          ;GO TO LOWER-LEFT CORNER
        (DISET B 0 (LIST 3 BOXSCL))       ;GLOBAL VARIABLE FOR SCALE,
                                          ;NO CHANGE TO PENUP STATUS
        (DISALINE B 0 200)
        (DISALINE B 200 0)                ;SEE HOW EASY IT IS IN
        (DISALINE B 0 -200)               ;INCREMENTAL MODE!
        (DISALINE B -200 0)
        (DISINI 0)                        ;BUT EASIER TO PUT IN POINT
        (DISAPOINT B 0 0 '(6 0))          ;IN RELATIVE MODE.  NOTE THAT
                                          ;SCALE IS NOT USED HERE
        (DISINI OASTATE)                  ;RESTORE ASTATE
        (DISCRIBE B))
  (DISINI 2)  (DISCREATE 1000 1000))      ;CREATES B, HOME AT CENTER
```

TO ADD SOME TEXT ON THE TOP OF THE BOX, ASSUMING ASTATE=0 AND THAT B'S VALUE IS THE NUMBER OF THE ABOVE ITEM:

```
(DISCUSS B -200 207 '(HERE IS THE BOX - SEE THE BOX) '(6 2))
```

TO MOVE THE BOX B RIGHT 100 UNITS:

```
(SETQ FOO (DISCRIBE B))
(SETQ FOO (LIST (CAR FOO) (CADR FOO)))
(DISLOCATE B (+ 100 (CAR FOO)) (CADR FOO))
```

TO PUT A CROSS WHERE THE PEN IS NOW, AND SOME TEXT WHERE IT USED TO BE BEFORE THE MOVE:

```
(DISMARK B -1)
(DISCUSS B (CADDR FOO) (CADDDR FOO) '(TURTLE SLEPT HERE))
```

TO BRIGHTEN UP THE BOX AND POINT [BUT THE TEXT "(TURTLE SLEPT HERE)" WAS ALREADY IN BRIGHTEST MODE, SO IT REMAINS UNCHANGED]:

```
(DISCHANGE B 2 0)
```

TO FLUSH THE BOX: (DISFLUSH B)

TO FLUSH ALL ITEMS ON THE LIST L: (APPLY 'DISFLUSH L)

TO GIVE UP THE SLAVE: (DISFLUSH)

Appendix 2 -- Relevant ITS Files

JDC;VTMEM >	TJ6 source for this memo
JDC;VTMEM (MEMO)	TJ6 output (for teletypes)
JDC;VTMEM XGPED	TJ6 output for xerographic printer
MB;URUG >	Microdebugger PALX source (BPT version)
MB;UMEMO >	Microdebugger documentation
GT40;URUG BIN	Microdugger PALX assembled binary file
GT40;VT07 >	VT07 PALX source
GT40;VT07 BIN	VT07 Binary
GT40;VT07 CREF	VT07 cross-referenced PALX listing
PJ;DITS >	DRAW SAIL source
SYSENG;LD10 >	MIDAS source for LISP display slave
JDC;FDITS >	file .INSRTed into above containing most GT40 code
SYS;ATSIGN 10SLAV	LISP PDP10 display slave
.INFO.;LISP ARCHIV	All changes to LISP for past several years

Appendix 3 -- Relevant Humans, Credits

Michael Beeler -- Wrote and maintains microdebugger. Wrote microdebugger section of this memo.

Joseph D. Cohen -- Helped write VT07, adapted LISP display slave for GT40, wrote all but microdebugger and Appendix 1 of this memo.

Pitts Jarvis -- Helped write VT07. Adapted DRAW for VT07

Jerome Lerman -- Wrote LISP display slave.

Guy Steele, John L. White -- Maintain ITS LISP and its documentation.