WORKING PAPER 64

MINI-ROBOT GROUP USER'S GUIDE

by

MEYER A. BILLMERS

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

March, 1974

Abstract

This working paper describes the facilities of the mini-robot group and the software available to persons using those facilities.

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 Defence and monitored by the Office of Naval Research under Contract number N00014-70-A-0362-0005.

CONTENTS

0. Introduction

This guide describes the resources available to users of the Mini-Robot group PDP 11/40 system. Chapter 1 describes the configuration of peripherals associated with the 11/40, and chapters 2 through 7 describe the software available. The operating system on the 11/40 is DOS, with some modifications described herein; consequently, a familiarity with the DOS system manuals is assumed.

Questions and comments about this guide should be directed to Meyer Billmers, room NE43-903.

1. The Configuration

The Mini-Robot PDP-11 system consists of a processor with 24K of core and the extended instruction set (EIS) option, a GT40 graphic display terminal and associated PDP 11/05 processor with 8K of core, two RK05 disks with 1.2 million words of bulk storage apiece, a high speed paper tape reader and a General Electric Terminet 1200 printer/keyboard. Specialized devices include a Vidicon video input device, an Analogic D/A and A/D converter, a high precision X-Y table, and numerous peripheral devices which may be connected to the Analogic converter and thus take advantage of its software and interfacing. Finally, there is T40, a 4800 baud TTY line to ITS.

Directly below the Vidicon interface is a patch panel which determines the terminal connections for the system. There are two input jacks at each labelled site on the panel. With the exception of 11-SYS, only the rightmost jack is active. The sites have the following interpretations:

11-SYS	-- PDP 11/40 system console
T40	-- 4800 baud TTY line to ITS
GE1200	-- GE Terminet 1200 printer/keyboard
DC11	-- interface reserved for T40
GT40	-- interface reserved for the GT40 graphic terminal
DL11	-- interface reserved for GE Terminet 1200

The following set of connections will be referred to as the standard configuration:

11-SYS	--	GT40
T40	--	DC11
GE1200	--	DL11

The standard configuration will be the configuration normally in effect, and will be assumed for the remainder of this guide unless otherwise noted. The standard configuration is wired into the patch board, so that if all the patch cables are removed it will still be in effect. Making a non-standard patch will override

these defaults.

In the standard configuration, the GT40 graphic terminal is the system console, known to DOS as device KB for both input and output. Other DOS devices known to the system are:

PR - high speed paper tape reader
DK0,DK1 - RK05 disks
GK - GE Terminet 1200 keyboard (input only)
GE - GE Terminet 1200 printer (output only)

A number of non-standard configurations is possible. Two of the more useful possibilities are described below.

NSC1: By connecting GE1200 to T40, the Terminet 1200 may be used as an upper/lower case ITS terminal off-line, depriving the 11/40 user of devices GE and GK, and obviating several system programs which refer to the Terminet and to T40.

NSC2: By connecting GE1200 to 11-SYS and T40 to GT40, the GT40 graphic display terminal may be used as a datapoint for ITS while an 11/40 user may be simultaneously using the Terminet as the 11/40 system console. In this case, device KB is the Terminet and devices GE and GK are once again not available.

In general, when using non-standard configurations, there will be a line speed problem which the user will have to correct. The Terminet is incapable of running at baud rates higher than 1200. Thus in NSC1 it will be necessary to change the speed of T40 to 1200 baud; this may be done as follows:

a) Using the GT40 as a datapoint (see chapters 2,3), type to ITS

```
LOCK^K
_40SPEED INPUT SPEED=1200  OUTPUT SPEED=1200 <CR>
_
```

b) Change the patch cables to NSC1, and at the terminet type:

```
^Z
:TCTYPE GE120 <CR>
```

c) When finished, type:

```
:TCTYPE GT H 36 W 110 <CR>
LOCK^K
_40SPEED INPUT SPEED=4800  OUTPUT SPEED=4800 <CR>
```

and then the patch board may be placed back in the standard configuration. In NSC2, the speed of T40 already will be correct for the GT40 when it is being used as a datapoint, but the line speed of 11-SYS must be changed to 1200 baud. This is done by inserting one end of a spare patch cord into the left-hand jack of 11-SYS.

2. File Transfer and Backup

Hardware troubles have plagued the PDP 11/40 system and caused numerous system crashes and extensive file loss. To protect against this, a number of backup procedures have been made available to the user. Forms of backup fall into three classes: disk backup on a PDP 11/40 backup disk, backup on ITS, and paper tape.

2.1 COPY

The copy command, RUN COPY, facilitates backup on a PDP 11 backup disk. It will copy from the disk loaded on drive DKn to the disk loaded on drive DKm, where n,m < 2. COPY will first query the user as to the direction of copy desired (DK0 to DK1, or the reverse); the user may then choose to backup selected files, or he may copy his entire disk. This latter process takes 70 seconds, and the copy is verified for correctness as it is done.

In normal operation, DK0 will contain the system disk (labelled SYSTEM) and is not to be written on, while DK1 will contain the user's personal files. It is not possible to COPY individual files unless DK0 contains a system; the user may COPY individual files from DK1, then mount his backup disk on DK1 and copy the files from DK0 to his backup disk. Files placed on the system disk for such temporary storage should be immediately deleted after the backup is complete.

When using COPY to backup an entire disk (this is normally the fastest way), the above restriction about the system being on drive 0 no longer applies. A user may type RUN COPY to get COPY into core, then dismount the system disk and mount his backup disk on DK0 and issue the command for COPY to begin the transfer. It is wise to run VERIFY before copying a disk, however, because the bit maps or file structure could be incorrect. See chapter 4 for details on how to use VERIFY.

2.2 ITS

ITS is a program which will make a software connection between T40 and the PDP 11 while still in the standard configuration. Typing the command RUN ITS to DOS will cause every character typed at the system console to be transmitted to ITS, and will cause every character received from ITS to be displayed on the GT40. Control-C is an exception -- it will cause the software connection to be broken (this does NOT log the user off ITS) and return control to DOS. Control-Q is a quote character; it is not transmitted, but causes the next character typed to be transmitted, and thus it is the only way to transmit a control-C to ITS.

2.3 SEND and BACKUP

SEND and BACKUP are programs available for transmitting files to ITS over a standard teletype line. A datapoint simulator should be running on the GT40 (see DISP, section 3.42) and the user should be logged in to ITS. To transmit a file, the following steps are needed: 1) Type RUN ITS, and place the GT40 in datapoint mode. 2) Type BACKUP^K. 3) Type control-C to break the connection and cause a return to DOS. 4) Type RUN SEND to DOS. SEND will prompt with a "#" when it is ready to accept a file specification. 5) Type the device, if applicable (default is DK0), the file name, the extension, and the user identification code (UIC). The default for UIC is the UIC of the user currently logged in. The file so specified will then be transmitted to ITS, and entered on the directory of the user logged in. Rubouts and control-u are recognized while typing the file specification to SEND. When the transmission is complete, SEND will prompt the user with another "#".

When all files are transferred, the user should RUN ITS, log off from ITS, and type a control-C to return to DOS.

SEND is useful for backing up files on your ITS directory, and also for obtaining assembly listings of ".LST" files produced by MACRO. SEND will transmit both ASCII and binary files, but

binary files are first encoded by SEND into a "safe" form which contains no dangerous control characters (like control-z) by coding every 3 eight bit bytes into 4 six bit ASCII characters. SEND will decide whether to encode a file based on its extension. Files with ".MAC" or ".LST" extensions are assumed to be ASCII and are not encoded; all others are assumed to be binary. The user may override these defaults with the /A (ASCII) ,/B (binary), and /P (picture) switches. For example,

```
$RUN SEND
#FOO.MAC
#DK1:TEST.LDA[1,1]
#PICT.1/P
#PAPER.TXT/A
```

will send the ASCII file FOO.MAC, the binary load module TEST.LDA on DK1, the picture file PICT.1, and the text file PAPER.TXT, with this last being send in ASCII mode.

SEND will give the message SYNTAX ERROR if the file specification was not meaningfully parseable, and the message I-O ERROR if the file specified does not exist or is read protected. In each case, SEND will prompt the user with another "#" and wait for a new file specification.

2.4 RECEIV and RESTOR

RECEIV and RESTOR move files from a disk directory on ITS to the PDP-11/40 disks. Their operation is a dual to SEND and BACKUP. The user must first make the connection to ITS by typing RUN ITS, log in to ITS, type RESTOR^K, return to DOS by typing control-C, issue RUN RECEIV to DOS, and type the name of the file which he wants transmitted. The user is reminded that DOS will not allow him to write on any directory except his own, unless he is logged in on the system UIC, [1,1], in which case he may specify any UIC for the file specification.

An implementation note: because the PDP-11 runs in real time, RECEIV can use larger buffers than SEND, and thus transfer

of large files in the 10 to 11 direction is faster than the reverse.

RECEIV uses the same coding conventions as SEND, so that a binary file encoded and sent up by SEND will be decoded by RECEIV. But if a /A or /B switch was specified with SEND in order to override a coding default, the same switch should be used with RECEIV. The user is warned that RECEIV will not work correctly with picture files.

2.5 PUNCH

The surest means of file backup possible is paper tape. The PDP 11/40 has no high speed punch, but programs may be put on the user's ITS directory using SEND and then punched out onto paper tape from ITS using the command PUNCH^K. PUNCH will type FILE: and then wait for a file specification (currently PUNCH will only punch files from the directory of the user who is logged in). PUNCH will punch one fold of leader, the name of the file in block letters, the date and time, a fold of leader, the file itself, and three folds of leader -- and then ask for another FILE: . PUNCH will make the same assumptions about coding as SEND and RECEIV, and will decode a binary file before punching it out. Again, the /A and /B switches will override these defaults. Thus,

```
PUNCH^K
FILE: PAPER TXT/A
```

will punch out the ASCII file PAPER TXT on your directory. If punching an ASCII file, PUNCH will look for the .END statement which must terminate all MACRO-11 source files, and if it is missing, PUNCH will print the warning message NO END STATEMENT ENCOUNTERED.

PUNCH is designed to work with SEND and to punch out files prepared on the 11/40 and SENT to ITS. It will, of course, work correctly on any ASCII files, but binary files not already in coded format should only be punched out using the appropriate TECO commands.

3. Scope Support Programs

The GT40 scope has been integrated into the DOS system as a standard output device with device name KB. It is an ASCII device, and so all output to it must be line oriented and must end with a line terminator (LF, FF, or VT). The keyboard attached to the GT40 is also a standard, ASCII, input device with device name KB. It will process rubouts and control-u's, and buffer one line of input at all times. KB will buffer up to one line of input (and process rubouts and control-u's) even if no .READ has been done. Characters so buffered will be echoed when the next .READ occurs, and if a line terminator was seen, the line will immediately be given to the monitor.

Connections exist between the GT40 and the 11/40, and also between the GT40 and ITS. When this latter connection is made, the GT40 may be used as a high speed (4800 baud) datapoint with rubout processing capability and upper and lower case letters.

3.1 GTLOAD

In order to use the GT40 as described above, the appropriate program must be loaded into its core. The PDP11/05 processor has a read only memory (ROM) loader which has a start address of 166000. This is an absolute loader, designed to load programs in over the TTY interface, which is seven bits wide, plus a parity bit. Since binary programs require all 8 bits in each byte, the ROM loader assumes that it will be given absolute loader format files in an encoded form, so that they will fit on the interface. The program GTLOAD will encode absolute loader files into the appropriate form, and load them into the 11/05. The command sequence:

```
(start the ROM loader at 166000)
$RUN GTLOAD
#FOO.BIN
```

will encode FOO.BIN and load it into the 11/05 processor.

GTLOAD has the ability to load more than one program into the 11/05. In general, the command sequence

```
$RUN GTLOAD
#FILE1.BIN,FILE2.BIN,FILE3.BIN,FILE4.LDA/R
```

will cause FILE1,FILE2, and FILE3 to be loaded sequentially into the 11/05 processor. Since the run option (/R) has been specified for FILE4, it will be run on the 11/40 after the loading is complete. If any file is to have the /R option, it must be the last one, and that file must be in the correct format to be run on the 11/40 (e.g. a .LDA file).

When GTLOAD is used to load multiple files into the 11/05, the ROM loader will halt after each file is loaded. In order to keep the two processors in synchronization, GTLOAD will also halt the 11/40. When this occurs, the user should restart the ROM, then press CONTINUE on the 11/40 and the next file will be loaded.

3.2 GTMAC

The PDP11/05 processor is, with only minor differences, capable of executing the same instruction set as the 11/40. Thus 11/05 programs may be written as if they were to run on the 11/40, assembled in absolute mode (with the Assembler switch .ENABL ABS) and then loaded into the 11/05 with GTLOAD. The instruction set for the GT40 scope processor is, of course, markedly different, and the MACRO-11 assembler does not recognize any standard mnemonics for the GT40 scope instructions.

To enable the user to program easily for the scope processor, the macro file GTMAC.MAC[3,3] has been made available. GTMAC.MAC contains macros which expand into the various scope instructions. These macros are:

SGM MODE,INTEN,K1,K2,K3 -- set graphic mode. MODE may be CHAR (character mode), SVEC (short vector), LVEC (long vector), PNT (point mode), RPNT (relative point), GRFX (graphplot x), or GRFY (graphplot y). It is a required argument; all other

arguments to SGM are optional. INTEN sets the intensity level. If present, it must be an integer from 0 to 7. K1,K2, and K3 are keywords. None or all three may appear; note that if INTEN is omitted, the first keyword must still be the third parameter. Their order is arbitrary, and they may be chosen from the following three sets:

LPI,NOLPI -- enables or disables light pen interrupts.
BLK,NOBLK -- blinking is on, off.
SOLID,LDASH,SDASH,DDASH -- line type (solid, long dash, short dash, dotted dash).

LDA K1,K2,K3,K4,K5 -- load status register A. The keywords K1-K5 may be chosen in any order and number from the following sets:

HALT - stops the scope
HI,NOHI - interrupt on scope halt is enabled, disabled.
LPY,NOLPY - the point of light pen interaction is intensified, not intensified.
ITX,NOITX - characters are italicized, not italicized.
SYNC - halt scope and restart on next 60 HZ clock pulse.

LDB NUM -- load status register B. NUM is the 6 bit positive graphplot increment.

The following group of macros uses the parameter conventions:

X6,Y6 are signed, 6 bit coordinates
X10,Y10 are signed, 10 bit coordinates
X10A,Y10A are positive 10 bit coordinates
HIDE is an optional keyword that specifies that the item is not to be intensified.

SVEC X6,Y6 -- short vector mode instruction
LVEC X10,Y10 -- long vector mode instruction
PNT X10A,Y10A -- point mode instruction
RPNT X6,Y6 -- relative point mode instruction
GRFX X10A -- graphplot x instruction
GRFY Y10A -- graphplot y instruction

As in MACRO programs, the .ASCII / text / statement is used to insert characters into a scope program.

To use these macros in your scope program, specify the following command string to the assembler:

```
$RUN MACRO
#DK1:,DK1:<GTMAC.MAC[3,3],DK1:FOO.MAC
```

and your file, FOO.MAC, will be assembled with GTMAC.MAC and all these macros will be defined.

3.3 GTROS

GTROS, the GT40 Trivial Operating System, is available to the user as GTROS.BIN[3,3], and as the following macros in GTMAC:

SJMPR - position independent scope jump

SJSR - jump to scope subroutine.

SJSRR - position independent SJSR.

SRTS - return from scope subroutine.

SINT - interrupt the 11/05. The argument to this macro specifies the address at which the 11/05 is to be started.

SINTR - position independent SINT.

SINTH - interrupt the 11/05, and halt the scope program.

SINTHR - position independent SINTH.

BELL - ring the console bell.

SEXEC - start execution of the scope processor at the specified address.

SREXEC - restart execution of the scope processor at the last point at which it was stopped by a SINTH or SINTHR.

To use the macros in GTMAC which are part of GTROS, the user must first load GTROS into the 11/05. The procedure is as follows:

```
(start the ROM loader at 166000)
$RUN GTLOAD
#GTROS.BIN[3,3],FOO.BIN
(restart the ROM loader and press CONTINUE
on the 11/40 when halted)
```

will load GTROS and then your program which makes use of the GTROS macros. GTROS is loaded from 320 to 740.

3.4 DISP

DISP is a general purpose display program which runs on the 11/05. To load DISP, the command sequence is:

```
(start the ROM loader at 166000)
$RUN GTLOAD
#DISP.BIN[3,3]
```

The starting address for DISP is 400. DISP has three modes, and the mode of operation is determined by the contents of the 11/05 switch register when it is started.

3.41 Literal DISP

SWR Bit 0=1. Whenever bit 0 of the 11/05 switch register is up, literal DISP will be selected. Literal DISP displays every character sent over to it on the TTY interface, so that no characters are interpreted. Literal DISP echoes over the TTY interface every character typed at its keyboard, and again no characters are interpreted. Literal DISP is useful for debugging purposes, by enabling the user to see every character his program outputs. Literal DISP will display control characters as an

up-arrow followed by the character. If the ASCII character 136 is sent to DISP, it will display as a carat, so that it will not be confused with a control character.

3.42 Datapoint DISP

SWR Bit 1=1. Whenever bit 1 of the 11/05 switch register is up, datapoint mode DISP is selected. By making the connection of the GT40 to the ITS line, it will function as a datapoint (at 4800 baud) with these differences: 1) Whenever bit 15 is zero, all letters will be shifted to upper case. 2) Whenever bit 15 is 1, lower and upper case letters will be available. 3) Rubouts are processed, i.e. rubbing out a character will cause it to be erased from the screen.

3.43 Display mode DISP

SWR Bits 0-2 all zero. Whenever bits 0-2 of the 11/05 switch register are zero, DISP will assume normal display mode operation. A square cursor will appear on the screen, marking the location at which the next character typed to DISP will be displayed. In this mode, DISP will display characters sent to it, interpret tabs, carriage returns and line feeds, echo characters typed on the GT40 keyboard across the TTY interface, interpret rubouts by erasing the previous character, and interpret control-u's by erasing an entire line. If bit 15 is 0, all letters will be converted to upper case. DISP keeps a buffer of several thousand characters, and so it is possible to scroll through the text at all times. There are two modes: home mode and echo mode. To enter home mode, type HOME on the GT40 keyboard side panel. To leave home mode, type LOCK-EOL. When in home mode, the effects of the keys on the side panel are:

(up-arrow) -- scrolls up ten lines in the buffer.
(down-arrow) -- scrolls down ten lines in the buffer.
(left,right-arrow) -- scrolls (left,right) 32 characters.
   This is useful for examining a line that is too long
   to fit on the GT40 screen.
(LOCK-EOS) -- jumps to the bottom of the buffer. Has the
   same effect as clearing the screen.

When in echo mode, keys on the side panel will echo their own ASCII values, so that (up-arrow) will echo as control-z and (down-arrow) will echo as control-k. Conversely, when in home mode, control-k will cause the screen to scroll up 10 lines.

DISP keeps its buffer in a ring, so that as a large amount of text is sent to DISP, it will overwrite the oldest text with the newest, and always display the last (buffersize) number of characters received. As DISP is receiving information, it will scroll up a line for each new line received. If the user scrolls away from the last (most current) line by typing (down-arrow), DISP's display will become "unstuck" from the bottom of the buffer and will allow the user to look at one screenfull of information while DISP continues to receive more characters and fills up its buffer. But when DISP wraps around and needs to overwrite the screenfull currently being displayed, it will begin to lose information being sent to it, and will ring the console bell. Scrolling forward (up-arrow) will allow DISP to resume writing into the buffer, and scrolling to the line currently being received will cause DISP to "stick" again and continue scrolling new lines onto the screen as they are received.

Display mode DISP will normally be running in the 11/05 when the GT40 is being used as the system console.

3.44 Other DISP options

Turning switch register bit 15 on will allow upper and lower case letters to be typed, while bit 15=0 causes all alphabetic characters to be upper-cased. Raising bit 2 will start execution of the ROM loader (DISP will not run again without being reloaded).

3.45 DISPLD

DISP may be loaded from ITS by the command :MAB;DISPLD while the ROM loader is running.

4. System Programs

4.1 Graphic editor, Version 5

The DOS editor, EDIT, now uses the GT40 to display 29 lines of buffer after each editor command is executed. The command RUN EDIT will cause the graphic editor Version 5 to identify itself. This editor is identical to the DOS editor (see the EDIT-11 manual) with one exception: a new command, nQ, has been added. The nQ command causes the last line of commands typed at EDIT to be executed n times. When the GE Terminet is the system console, EDIT will compensate by displaying only three lines of buffer after each edit command.

4.2 LISP

PDP11 LISP provides the user with most of the features of a full scale LISP. Atom print names are restricted to being three characters in length, and the print names should be alpha-numeric. This is a shallow binding LISP with saved values of the variables kept on the stack, along with function calls and associated information (such as function return addresses and parameters being passed). There is only one stack. Currently there is no way for a LISP program to do its own file structured I-O.

The command RUN LISP will cause mini-robot LISP to identify itself and prompt the user with a vertical bar. Expressions may then be evaluated directly at top level, or the user may type a control-G, which causes LISP to print a "#" and wait for a set of file specifications. The general form of the file specifications is:

```
#DV:OFILE1.EXT[uic],OFILE2.EXT[uic]<IFILE.EXT[uic]
```

where all standard defaults are recoginized (DV defaults to DK0: and [uic] defaults to the current user).

In this notation, OFILE1 is the primary output file, OFILE2 is the secondary output file, and IFILE is the primary input file. If a non-file structured device is specified, no filename, extension or uic need be supplied. The primary input file and one of the output files must be supplied.

Once this file specification has been typed in, followed by <CR>, control will be transferred to LISP. If the primary input file was KB: or GK:, LISP will prompt the input device with a vertical bar and wait for an S-expression. <CR> and <LF> may be used as atom name separators, but only a matching number of right parentheses will close an expression. LISP will then ignore any other characters typed except a following <CR>, evaluate the expression, and send its value to the primary and secondary output files. When the output is complete, LISP will once again prompt the input device with a vertical bar.

If the input file specified was other than GK: or KB:, LISP will read in, evaluate, and output each s-expression on the input file. When the last s-expression has been read and evaluated, LISP will prompt the user with another vertical bar. Bindings established from previous inputs remain in effect.

4.21 LISP control characters

LISP recognizes the following control characters on an interrupt level, regardless of the device open for input.

CONTROL-D returns control to the monitor.

CONTROL-P suppresses output to the primary output file for the duration of the s-expression currently being EVAL'd. If output is off, CONTROL-P will resume output.

CONTROL-S similar to CONTROL-P, but for the secondary output file.

CONTROL-G terminates computation of LISP, closes all input and output files, prints a "#" and waits for a new set of file

specifications. When <CR> is typed, LISP will resume its computation, but it will take input from the new input file.

CONTROL-H is the debugging interrupt. It interrupts LISP but preserves the state of the LISP computation and the state of all I-O. It will temporarily open the GK: or KB: (whichever was open last) for input. LISP will respond by issuing the error message ERROR: EXTERNAL INTERRUPT, and come to top level with a vertical bar. The user may then examine the state of his program, change bindings, and type (CNT) to close the keyboard for input, and resume computation and I-O from the previously opened files.

4.22 File Specification Error Messages

LISP maintains the following set of error messages which pertain to the file specification entered at a "#":

SYNTAX ERROR

This error indicates a non-parseable file specification.

LINKBLOCK ERROR

This error indicates that the monitor did not have sufficient space for a buffer for one of the devices specified.

I-O ERROR
FILE TYPE: PI ERROR CODE: B

This error indicates a problem with one of the files in the file specification. PI indicates that the file causing the error was the primary input file; PO and SO are the other possibilities. The error code is the ASCII character obtained by adding 100 to the error code in the FILEBLOCK (see DOS monitor manual, page 3-82, for a complete list of these error codes). The error codes which can appear as a result of an improper file specification to LISP are:

@ - an attempt was made to open a file for output twice.

B - file does not exist, or an attempt was made to open a file for both input and output.

F - file protection violation.

J - the UIC referenced is not known to the system.

L - an attempt was made to create an output file with an illegal name.

O - the file extension was ".CIL".

END OF FILE ENCOUNTERED
BEFORE END OF EXPR

This error indicates that the input file did not contain enough right parentheses.

4.23 LISP Functions

The following are standard LISP functions:

LM -- lambda

NLM -- nlambda

EVL -- eval

' -- quote

+ -- addition

- -- subtraction

* -- multiplication

/ -- division

GT -- greater than

GE -- greater than or equal to

NOT -- boolean not

CAR -- car

CDR -- cdr

RPA -- replace car

RPD -- replace cdr

CNS -- cons

NUL -- the null predicate

ATM -- the atom predicate

NUM -- the number predicate

LST -- the list predicate

OTH -- the none of the above predicate

= -- the equality predicate

STQ -- quoted assignment

SET -- assignment

AND -- boolean and

OR -- boolean inclusive or

CND -- cond

LIS -- list. Makes a list of its arguments (may take an indefinite number).

PRG -- prog. Takes two arguments, a list of local variables and the prog itself.

RET -- returns a value from a prog.

GO -- jumps to a label within a prog.

GBG -- garbage collector. The value returned is the number of free cells.

CL -- characters to list. This will take input from the input device in the form of a character stream, build a list, and return the list as its value.

LC -- list to characters. Takes a list as argument, and outputs the character stream corresponding to that list on the current output devices.

SYS -- regenerates and restructures the system. (SYS) should be called only in dire instances. SYS takes five arguments, which are the size of the hash table, of atom space, of list space, of the "margins", and the stack. The margins are areas not normally available to the system, but are reserved in the event of a list space overflow or a stack overflow. When an overflow occurs, a soft error message will be generated, and the user will have access to the margin space while he tries to free more space. Overflowing the margin causes a hard overflow error, which is irrecoverable. Null arguments to SYS leave the corresponding argument position unchanged.

RST -- restore. Flushes the stack and restores top level bindings to all atoms, then returns to top level evaluation.

BRK -- break. Used to set break points in program execution. It prints out an ID which is determined by its first argument. If the first argument to BRK is between one and twenty, it will print out one of the twenty standard system error messages, and if the

first argument is outside this range it will print out BREAK: USER DEFINED, followed by the value of the first argument. In any case, BRK will then print out the values of all other arguments passed to it, and then will return to the top level evaluator.

CNT -- continue. Restarts the program from its state at the occurrence of the last BRK. CNT flushes the stack down to the last BRK block, but restores no atom bindings (thus, there should be no open lambdas on the stack), and then continues operation. CNT may be used after a user defined break point, or after a CONTROL-H interrupt. If called with an argument, CNT returns the value of that argument as the value of the last BRK.

TOP -- top level evaluator

BT -- backtrace. This is a diagnostic routine which has three modes: 1) no arguments. BT will print out the name of every function call on the stack. 2) two numeric arguments. BT will print out everything on the stack from the beginning of the (arg1th) function call and continuing for (arg2) function calls. 3) two addresses. BT will dump core between the two addresses, interpreting everything between (i.e. pointers into list space will be followed, and the corresponding lists will be printed out).

LISP's internal representations for numbers and for addresses are different; numbers are represented internally by odd integers (N is represented by 2*N+1) while addresses are always divisible by four. When communicating with LISP, the user should prefix all addresses (for example, the arguments to BT in dump mode) with @; all other numbers will be assumed by LISP to be numbers. LISP will also use this convention when outputting.

4.3 RUG

RUG is a symbolic debugger which replaces the DDS debugger ODT. The command RUN RUG will cause RUG to identify itself and prompt with a "#", indicating that a file specification is expected. Typing

```
#FILE1
```

will cause RUG to take the following actions: 1) A search will be initiated for FILE1.LST (keep in mind that FILE1 represents a general file specification, which may include a UIC or device); this file must be present, or RUG will give an error message and prompt with another "#". 2) The symbol table in FILE1.LST will be searched and the symbols found therein will be defined so that they may be referenced by name during the debugging process. 3) RUG will attempt to locate FILE1.MAP. If found, RUG will read the relocation constants for FILE1 from it, which will be added to all relocatable symbols in FILE1's symbol table (those which have an "R" next to them in the symbol table portion of the listing file). If no .MAP file is found, RUG will warn the user of this fact, and then assume a relocation constant of zero. (Recall that the relocation constant applies only to relocatable symbols; if FILE1 is an .ASECT with no .CSECT, then the relocation constant is never used.) 4) A search will be made for FILE1.LDA, and if it is found it will be loaded into core. If this file does not exist, RUG will print an error message and another "#".

NOTE1: The file FILE1.LST must exist, but it need not be a full listing file. If the following string is typed to MACRO when assembling FILE1:

```
#DK1:,DK1:/NL<DK1:FILE1
```

then a listing file will be created consisting solely of a symbol table for FILE1, which is enough to satisfy RUG.

NOTE2: If FILE1.MAP does not exist, a relocation constant may still be supplied to RUG by typing

```
#FILE1/RC:42642
```

NOTE3: If FILE1.LDA was obtained by linking together the separately assembled modules FILE1.OBJ,FILE2.OBJ, and FILE3.OBJ, the user may type

```
#FILE1,FILE2,FILE3
```

and RUG will load FILE1.LDA, use FILE1.MAP for the three relocation constants (one for each module), and read in all the symbols from FILE1.LST, FILE2.LST, and FILE3.LST.

NOTE4: When RUG prints its "#", the user may type just a <CR>. This will cause RUG to enter debugging mode without defining any symbols or loading any user programs into core.

NOTE5: RUG will allow the user to define approximately 600 symbols, and then will print the message "TOO MANY SYMBOLS -- THERE WERE N UNDEFINED SYMBOLS". Extra symbols will merely be ignored by RUG.

NOTE6: All symbols beginning with the letter "Z" and all symbols whose value is less than 100 will not be defined by RUG. This allows the user to have "garbage" symbols in his program which will not occupy space in RUG's symbol table.

When RUG has completed the process of defining user symbols and loading the user program, it will enter debugging mode and prompt with a "*". At this point the user may execute any of the RUG commands described below. Debugging mode may be entered at any time by starting the processor at 24000 if there is a copy of RUG in core.

4.31 Location Opening Commands

The following commands govern the opening and closing of core locations.

foo/	opens location foo and closes currently open location
foo\	same as above, but opens foo in byte mode
/	when typed at an open location, opens the contents of that location and closes the original location.

^ closes the currently open location and opens the previous location

<CR> closes the currently open location

<LF> same as <CR>, and additionally opens the next location

[same as /, but opens the left hand (source) argument of the currently open location or of the last location opened if none are open

] same as [, but opens the right hand (destination) argument

< undoes an indirection chain (of [,/, and] commands) and opens the original open location.

When RUG opens a location, it types out the contents of that location and then allows the user to change the contents by typing something to be entered there. Anything the user types at an open location which is not a RUG command and which makes sense to RUG will be stored in the open location when it is next closed. If the location is closed without anything being typed, its contents will be unchanged. Note that only one location may be open at a time.

4.32 Typeout Modes

RUG has a number of typeout modes which govern the way in which the contents of a location being opened are to be interpreted. Preceeding a typeout mode with one altmode (ESC) will set that mode temporarily (so that it will only affect the currently open location), while preceeding it with two altmodes sets the mode permanently. When RUG is started, it is in instruction mode.

The typeout modes are:

I - instruction mode
S - symbolic mode
C - constant (numeric) mode
A - ASCII mode
R - Radix50 mode
- decimal constant mode

4.33 Typein Modes

Regardless of the typeout mode, RUG will always allow instructions, octal numbers, symbols, or expressions composed of symbols, numbers and "+" and "-" signs to be typed in to any open location. There are a number of special typein modes which are available, however:

& - enters up to three radix50 characters
' - enters one ASCII character
" - enters two ASCII characters
37. - enters the decimal number 37

4.34 Breakpoints

A breakpoint is a tagged instruction in a user program which, when executed, causes control to return to RUG so that the state of the program may be examined. The following RUG commands govern the use of breakpoints:

$B	sets a breakpoint at the current value of "." (the location currently open, or the last location to have been opened)
foo$B	sets a breakpoint at foo
$D	deletes all breakpoints
foo$D	deletes the breakpoint at foo
$G	starts execution of the user program at the program's

start address

foo$G starts execution of the user program at foo

$P proceeds with the execution of the user program. This i's only valid after at least one breakpoint has been encountered.

n$P same as $P, but sets a proceed count of n for the breakpoint being proceeded past. This count will be decremented each time the breakpoint is encountered, and RUG will not receive control until the count has gone to zero, so that the breakpoint will effectively be proceeded past n times.

$N single step. Executes one instruction from the user program and then re-enters RUG.

n$N Executes n instructions in the user program.

In RUG the user is allowed to define 8 breakpoints, and when a breakpoint is encountered RUG will type B3;FOO/ INC R3, for example, if breakpoint number three were set at FOO. The user should note that the instruction at FOO (INC R3 in our example) will not have been executed. Other messages RUG will type are SS; when single stepping, MPV; for a memory protect violation (attempt to reference non-existant memory or an odd boundary with a word instruction), IOC; for an attempt to execute an illegal opcode, and BE; for a bad entry into RUG caused by the user program's executing a BPT instruction or setting bit 4 of the processor status word. Including a BPT in the user program is often a convenient way to assemble in a breakpoint.

4.35 Miscellaneous RUG Commands

n$T types out the contents of the next n locations on the GE Terminet

foo$E searches for words which reference effective address foo (such as the offset word in an index mode instruction)

foo$W searches for words which contain foo

foo: when typed at an open location, causes the symbol foo to be defined and equal to the address of the open location

N!foo defines the symbol foo to be equal to N

foo^K half-kills foo, so that RUG will suppress its use when doing typeout but will still accept it when doing typein

^O returns to the monitor

The following are some useful RUG locations:

.H -- highest core address
.B+n -- location of breakpoint number n (if not set, this location will contain the symbol NOBKPT)
.C+n -- proceed count for breakpoint n, normally set by doing m$P
.P -- RUG's program status word
.S -- user program's program status word
.M -- mask used for $E and $W searches, to be and'ed with items being matched
.M+1 -- low core limit for searches
.M+2 -- high core limit for searches

Note on searches -- if a $E or $W search succeeds, it will type the address at which it first succeeds. To continue the search, type "!", and to terminate the search, type <CR>.

4.4 VERIFY

Hardware problems are frequent on the 11/40 system, and often are responsible for the corruption of the file structure on a disk. VERIFY is a program which checks the file structure and reports on any problems it finds. If there is any doubt about the data on a disk, or if the system begins to act strangely, VERIFY should be run to ensure that the disk has not been clobbered. It is wise always to run VERIFY before doing a disk copy, because a bad file structure will be copied onto the backup disk, corrupting it as well.

The sequence of commands is as follows:

```
$AS GE:,5
(place the Terminet on line)
$RUN VERIFY
WHICH DEVICE (SY,DK,DF,DC,DT)?
DK
UNIT NUMBER?
1
MODE (NORMAL, LIST, FIX, SEARCH, OR ALL)?
N
```

VERIFY will return to the monitor when it is done. There should be no output. Any output by VERIFY indicates that the disk has been corrupted; the output should be brought to Meyer Billmers (ITS mail address: MAB) as soon as possible. DO NOT ATTEMPT TO DELETE FILES, OR OTHERWISE FIX THE PROBLEM YOURSELF, UNLESS YOU SPECIFICALLY KNOW WHAT YOU ARE DOING. This may make the situation worse and may cause unexpected loss of parts of the disk.

4.5 LIST

LIST is a program for spooling listings onto the GE Terminet, allowing the user to perform other tasks while his listing is being printed. Typing the command RUN LIST will cause a "#" to appear. When a standard file specification has been entered, LIST

will begin to print a two page identification header. When this bigprint is finished, LIST will return control to the monitor, and the user may RUN any other program he wishes, with a few exceptions:

1) The user may not log out. 2) Typing control-C and then KI at any running program will terminate LIST's output. If a control-C is unintentionally typed, CO will return to the running program without interrupting the listing in progress. (ITS is an exception; control-C typed at it is the normal return to monitor -- see section 2.2). 3) Control-D is normally used to return control to the monitor from a running program, and it will not interfere with a listing in progress. 4) The user may not attempt to do output to the Terminet while a listing is in progress. 5) The user is cautioned that running MACRO and attempting to produce a large listing file may interfere with a listing in progress, but specifying the /NL switch in the listing file position to MACRO is always safe. 6) All other system programs will time share normally with the lister, although RUG will temporarily allow the definition of fewer symbols.

4.6 USERS

The command RUN USERS will display the current authorized user list for the 11/40 system, along with UIC's and ITS login names for each user.

4.7 ABSLDR

ABSLDR is an absolute loader, used to load .BIN files from paper tape. The tape should be mounted and the reader turned on before the command RUN ABSLDR is issued.

5. Picture Processing

Currently the mini-robot 11/40 has a Vidicon picture scanner/ digitizer as its only means of picture input. The Vidicon will scan a picture 30 times a second, and can digitize as many as four points on the same vertical sweep in 65 usec., making it possible to digitize an entire picture in two seconds. In practice, a somewhat slower rate is used.

5.1 VIDIN

VIDIN is a program used to take pictures from the Vidicon scanner, digitize them, and store them as contiguous (random access) files on disk. The calling sequence is

```
$RUN VIDIN
DEVICE NAME?
VC
ENTER DATASET SPECIFICATION
DK1: PICT.1
ENTER WINDOW COORDS
2,3,5,7
ENTER WINDOW COORDS
<CR>
ENTER HEADER INFO
THIS IS A PICTURE OF A CIRCUIT BOARD,
DAYLIGHT ILLUMINATION, TAKEN MAR 17
<CR>
OK, ALL DONE

$
```

The device name VC indicates that the Vidicon has been selected as the input device, and PICT.1 is the desired name of the picture. The window coordinates are the x,y coordinates of the upper left hand sector of the picture to be taken (here: 2,3) and the x,y coordinates of the lower right hand sector of the picture (5,7). This enables the user to specify a rectangular

picture he wishes taken, from 1 to 64 sectors in area. Each sector is 64x64 points on the screen and requires 8 disk blocks of storage. The screen is 8 sectors on a side, and sector 0,0 is in the lower left hand corner. Each set of coordinates given to VIDIN will be scanned once, but the picture will not be taken until a line with only a <CR> is typed, so that one may see which sections of the picture will be taken, and be able to change one's mind before "clicking the shutter". The header info is an ASCII string used to identify and comment the picture. It too is terminated by a line with only a <CR>.

5.2 PROFIL

PROFIL is a program intended to be used before VIDIN to obtain information about the picture on the monitor. If 11/40 switch register bit 13 is a 1, PROFIL will print a grid of VIDIN sectors. Otherwise, PROFIL will allow the user to select a cross section of the picture, and it will display on the monitor such information as an intensity plot of the points on the cross section, or a histogram or cumulative histogram of the intensities on the cross section. PROFIL is useful in setting the black and white level adjustments in the Vidicon digitizer, and for deciding which sectors of the picture to take with VIDIN. PROFIL explains its options when it is started.

5.3 MAPPER

MAPPER is a program for inspection of the picture files taken by VIDIN. In MAPPER one can select a specific sector of a picture file, obtain statistics about that sector (such as maximum and minimum intensities, average intensity and standard deviation of the intensity distribution), obtain a histogram of the intensity values for a sector or a single x or y value of a sector, and then make a map of the sector. The maps are made by assigning ASCII characters to each of ten specific intensity ranges, and can be printed on either the GT40 or the Terminet. When MAPPER is started it lists the available commands, and many of the commands offer instructions in their use. A detailed description of MAPPER

is available on ITS as DHT;MAPPER INFO.

6. The Devices

In addition to the Vidicon, the 11/40 system has two other devices for which software exists: an x-y table and an Analogic A/D and D/A converter which drives all the other devices (e.g. the mechanical arm, the mirror deflection system, and the modulatable laser).

6.1 The Analogic Converter

Two system macros exist for use with the Analogic converter. These are .A2D and .D2A. To define these macros in your program, the assembler directive is

```
.MCALL .D2A,.A2D
```

These macros will now be defined, and when called in a user program they will expand into code necessary to do digital-to-analog and analog-to-digital conversions. .D2A takes two arguements:

```
.D2A CHNUM,VALUE
```

where CHNUM is some variable which contains the channel number (from 0 to 7), and VALUE is a variable which contains the number to be converted. The contents of VALUE may range from 3777 (which will produce nearly +10VDC on the desired channel) to -4000 (-10VDC). This macro requires approximately 5 usec. delay time for the conversion.

.A2D takes two arguments:

```
.A2D CHNUM,VALUE
```

and it will convert the voltage on the channel contained in variable CHNUM, and will do a busy wait until the conversion is complete (this takes about 3 usec.) The converted value is left in the variable VALUE.

6.2 The X-Y Table

The assembler directive

```
.MCALL .TABLE
```

will define a macro called .TABLE which, when called, expands into a set of subroutines for moving the x-y table. These routines are called using the convention JSR PC,SUBR. The table subroutines are:

CALTBL — calibrates the x-y table and leaves it in position (0,0)

VELTBL — sets up the velocity for the next table movement. R0 should contain the velocity for x and R1 should contain the velocity for y

ABSTBL — moves the table to the absolute location (x,y), where x is contained in R0 and y is contained in R1

RELTBL — causes the table to move relative to its current location. The x and y in R0 and R1 respectively are taken as offsets for the relative motion and are preserved so that successive calls of RELTBL will reference them.

Note: neither ABSTBL nor RELTBL wait for the table to finish moving. Neither should be called if there is a chance that the table is in motion without first calling WTTBL.

WTTBL — waits for the table's motion to finish. WTTBL will take a skip return if the table motion completes normally (without running into a limit stop). If a limit stop is encountered, WTTBL will take a non-skip return. Thus:

```
JSR PC,WTTBL
(error return)
(normal return)
```

WHRTBL returns the table's x position in R0 and its y position in R1.

NOTE: These macros will protect the user from moving the table to a negative position; motion will stop at zero, and the table will not have been decalibrated. Similarly, attempting to move the table too far forward in either x or y will result in a cessation of the table's motion without running into the physical limit stops or decalibrating the table. WTTBL will take the error return whenever such a premature stoppage occurs. Note also that all coordinates kept by the .TABLE routines are relative to the calibration point, and thus CALTBL always should be the first routine called.

7. Miscellany

7.1 System Bugs

There is a known bug in LINK. If a .MAP file is being produced, the name of the file must be specified, even if it is the same as the input file. Doing #DK1:,DK1:<DK1:FILE/E will not only fail to produce FILE.MAP, but it will also destroy a randomly selected portion of the disk.

7.2 SUPERD

Do you have a file you can't delete with PIP? You say you've tried unlocking it (#FILE.EXT/UN) and you still can't get it out of your life? Try SUPERD, the super-deleter. Due to some unclear hardware/software problems, EDIT and MACRO will occasionally produce files which cannot be deleted with PIP even after being unlocked. This seems to be especially true of the EDITor after it mysteriously dies. SUPERD takes one file name (it must be on the directory of the person logged in) and hopefully will delete even these stubborn cases. If it fails to work, leave some mail on ITS for MAB. It's probably wise to do a VERIFY after using SUPERD, especially if it fails to work.

APPENDIX 1

GETTING ON THE 11/40

1. If necessary, turn the power on. Place the RUN/LOAD switch of the uppermost disk drive into the RUN position. Wait for all three lights to come on.

2. Ensure that the WT PROT light is off on that drive.

3. Place the HALT/ENABLE switch on the PDP 11/40 processor into the ENABLE position, and press CONTINUE.

4. Repeat step 3 on the PDP 11/05 processor (under the GT40 scope).

5. If a square cursor appears on the scope, the GT40 is in a happy state. If not, press HALT on the 11/05, place 400 in the 11/05 switch register, press LOAD ADDRESS, raise the HALT/ENABLE switch to ENABLE, and press start. This should produce a square cursor on the GT40. If it fails to do so, go to step 5A. If the cursor does appear, go to step 6.

5A. Press HALT on the 1105 processor, place 166000 in the 11/05 switch register, press LOAD ADDRESS, raise the HALT/ENABLE switch to ENABLE, and press START. Turn the brightness all the way up.

6. Type a <CR> on the GT40 keyboard. A "$" should appear. If not, press control-c and then KI. If still no "$" appears, go to step 7. Otherwise, go to step 8.

7. To bootstrap DOS, HALT the 11/40 processor. Place 173170 in the switch register, and press LOAD ADDRESS. Move the HALT/ENABLE switch to ENABLE, and press START. The mini-robot group message and a "$" should appear on the GT40.

8. If you failed to get a cursor in step 6, set the switch register on the 11/05 to zero, and type the following:

```
$RUN GTLOAD <CR>
#DISP.BIN[3,3] <CR>
```

and a "$" and a cursor should appear presently. The above should be typed in upper case letters, and for this shifting will be necessary. Further, the 11/05 processor must first have been started as in step 5.

9. If you have still failed to raise the system correctly, find someone who knows how!!

APPENDIX 2

INDEX TO PROGRAMS