 MICROCOPY RESOLUTION TEST CHART
 NATIONAL BUREAU OF STANDARDS 1963-A
A.I. Center
Research Programmer Support

Final Report
September 15, 1986

JAYCOR Contract #6255

In Response to:
Contract #N00014-85-C-2552

Prepared For:
Naval Research Laboratory
Washington, D.C. 20375-5000

Prepared By:
JAYCOR
1608 Spring Hill Road
Vienna, Virginia 22180
1. INTRODUCTION

JAYCOR is pleased to submit this final report summarizing the tasks performed by JAYCOR at the Navy Center for Applied Research in Artificial Intelligence (NCARAI) of the Naval Research Laboratory, under Contract #N00014-85-C-2552. This report gives an overview of the work performed to meet the tasks, location on NCARAI computers where the software is stored, documentation of the software, and pertinent information on any problems encountered during the performance of the tasks.

The report is organized on a task by task basis, each task briefly explained and the work summarized. Any software produced, obtained, or otherwise installed to satisfy task requirements is included in source form in Appendix. In addition to this summary final report, the reader is referred to the monthly reports submitted over the entire period of the contract to more thoroughly document the work performed.

2. TASKS AND WORK PERFORMED

In addition to the maintenance of the UNIX system on the VAX computers, JAYCOR implemented functions and programs for a number of peripherals. Background information on these peripherals and key points of interest about the software is summarized below. If the software was modified from previously existing software rather than completely
rewritten by JAYCOR personnel, pointers to the original source are also given. Location of online source code and documentation is noted.

2.1. 3/4" VIDEO TAPE PLAYER (VAX and LMI)

During the previous system programmer contract period, the AI Center took delivery on a programmable 3/4" video tape player. This player has much the same functionality as the 1/2" player which JAYCOR programmed under that contract. This software was thus usable in very slightly modified form with the larger tape size.

To use and program the player from the VAX, the user connects it to the TERABIT computer interface according to the TERABIT manual. To use from the LMI LAMBDA computer, the user connects the player to the TERABIT interface, then connects the TERABIT interface to the LAMBDA serial port labeled "SDU-Serial-B". One then runs the software written by JAYCOR personnel under the LISP environment as documented in the manual in the Appendix. Each function can be individually invoked or can be combined into a complex program of function calls as suits the user.

All software for this task is contained in the VAX-11/780 directory "/aic2/smith/contracts/sys2" under appropriate subdirectories. Additional copies of the LMI-specific software are also available on the LMI computer.
2.2. 1/2" VIDEO TAPE AND DISK PLAYERS (LMI)

The tape and disk player software programmed under the previous contract for the VAX was transported to the LMI computer both to allow continuity of interface and to prevent duplication of previous programming effort. During the transportation modifications to the FRANZ LISP code were made to make it COMMON LISP compatible. Full functionality was maintained during this transfer.

The user interface is quite similar for both of these peripherals to that of the 3/4" tape player, hence users are not presented with a confusing change in command structure. The Appendix contains copies of the manuals with small examples of usage demonstrating this similarity.

The software and documentation for these peripherals is located in the VAX-11/780 directory "/aic2/smith/contracts/sys2" under appropriate subdirectories. Where duplicate copies of software were possible, hard file links have been used instead.

2.3. RACAL-VADIC MODEM DRIVER (LMI)

The LMI LAMBDA computers come with a version of the public domain KERMIT program. This program uses whatever modem (or simple connection) is attached to the "SDU-Serial-B" rs232 port. For this task, JAYCOR personnel connected the Racal-Vadic modem to the serial port, enabled the port by running the KERMIT program, and tested out the
functionality of the modem through connections from the LMI to the VAX computers. The KERMIT program was successfully tested by transferring files from the LMI to the main VAX computer. All Racal-Vadic internal operations were accessible through the KERMIT software interface.

To use the KERMIT program while logged into the LMI computer, type in <SYSTEM>K, thus starting the KERMIT program. Then, using the mouse in the standard manner (explained in LMI documentation), "click" on the "Connect" menu selection. One can then use the Racal-Vadic modem programming commands as if connected through a terminal.

The KERMIT software (enabled, tested, and verified, but NOT supplied by JAYCOR) is located on the LAMBDA computer under the system source directory. Due to its large size it is not duplicated in this summary report.

2.4. LA50 PRINTER DRIVER (LMI)

Two versions of printer drivers were written by JAYCOR personnel for this task. Both versions are capable of printing rectangular screen areas onto an LA50 printer connected to the "SDU-Serial-B" rs232 port. The areas give a somewhat accurate graphic rendition of the actual screen display, however the aspect ratio is, as expected, not 1:1.

One major problem with the use of the serial port and the LA50 printer was uncovered during the implementation of this software, namely the lack of adequate flow control
during the printing process. For some larger sized rectangular regions, the LA50 can not keep up with the LAMBDA's stream of data, hence attempts to use XON/XOFF flow control to synchronize. Unfortunately, the LAMBDA's serial port software does not use flow control in its current configuration so some data can be lost, resulting in both skew of and superfluous patterns in the output image.

The utility of the LA50-based printing has been superceded by the much higher resolution printing available with the Ethernet-connected IMAGEN laser printer. This printer produces much finer images much more quickly over the high speed network link. All software is kept on the LAMBDA computer.

2.5. RESTRICTED SHELL

As JAYCOR noted in its proposal for the subject contract, this task is met by having a "shell" field in the password entry for a user which will only invoke a specific program, logging off on exit. Due to the nature of the library system installed by the NCARAI, this method of access restriction will not be necessary at this writing, instead a unique login ID and password are used with NRL's library system (explained in the following sections dealing with library tasks).

2.6. RAMTEK SOFTWARE

Under the initial system programmer contract awarded to
JAYCOR, graphics functions were implemented which allowed users to interface their C programs to the Ramtek devices on both the VAX-11/780 and VAX-11/750 (ATE machine). This software provided line and rectangular region graphics as well as text in colors selected by the user. These functions have been extended and an interface to the FRANZ LISP environment has been implemented to meet the requirements of this task. In addition, certain image processing functions have been implemented and a sample set of images obtained for testing purposes.

All software for this task is stored in the VAX-11/780 directory "aic2/smith/contracts/sys2" under an appropriate subdirectory. The functions typically are all self documenting due to their simple usage, but other documentation is also in the same location.

2.7. LIBRARY DATABASE

The library database was tested thoroughly during the initial period of this contract. After this testing and communication with other libraries in the area (Library of Congress, NRL main library), it was decided to obtain the standardized OCLC/DIALOG/LS2000 accessing permissions, thus maintaining compatibility with libraries throughout the country. An OCLC terminal (actually an IBM PC with appropriate hardware and software enhancements) was received and installed. It has proven to be a valuable asset to the library. When the main NRL library LS2000 system comes
fully online, the NCARAI library will be able to access and use the main NRL database for such things as verifying the availability of references, checking out of references, and doing subject/author/keyword searches. Arrangements have been made with the main library to obtain "zebra codes" for the current references in the NCARAI library. These will be attached to the spines of the books and allow the computerized checking out of references to be performed.

2.8. LIBRARY ORGANIZATION

The library has undergone considerable rearrangement since contract award. References have been organized to more closely follow standard library practice, circulation for periodicals has been totally redone to provide more timely circulation, and the physical arrangement of the library has been changed to allow easier access to the stacks.

In addition to the above, the design of an expanded library has been submitted to the COTR. This new library takes into account the tremendous influx of new references and the expanding space necessary to hold them. Using the current "community" room in the AI Center, the library will be able to handle acquisitions at the current rate for a number of years. At the same time, the study atmosphere will be improved significantly through the presence of natural light from the large window area and the ability to use study carrels.
2.9. DOCUMENT ORGANIZATION

A large number of documents have been ordered and received from DTIC. These documents comprise the collection of works which in some manner reference AI. The documents were obtained in both hardcopy and Microfiche form. The current plan for cataloging these documents is to order the DTIC listing from which they were ordered. This would be in addition to the ordered storage of both the fiche and the hardcopy (see the Inventory section below).

2.10. POINT OF CONTACT

Throughout the period of the contract the JAYCOR librarian has interacted considerably with the user community. This interaction has covered the range of duties from simple reference lookup to the arrangement of interlibrary loans. Professional relationships have been developed with the NRL library as well as with the Library of Congress. These relationships are ongoing.

2.11. INVENTORY AND REQUIREMENTS ANALYSIS

The JAYCOR librarian has been closely involved with the COTR in performing a requirements analysis for the library. From this analysis the needs of the library were discovered to exceed current library capabilities. This led to the planning of the new expanded library space using the NCARAI community room. This room will be much more conducive to study and research.
Another product of the library analysis was the acquisition of a Microfiche hardcopy printer. This machine has been very well received. Using standard fiche, the user can view a document and, if desired, make a hardcopy version at will. Through the use of fiche for much of the documents received by the Center, considerable space will be saved.
APPENDIX

This Appendix contains copies of the source code of all software used to satisfy the requirements of the tasks of the Statement of Work. This software was either written, modified, or acquired by JAYCOR as noted.
Video Tape Player

Naveen Hota
JAYCOR

Usage

The video tape player can be operated from lisp. The routines are written in C. These routines should be loaded in to lisp before using them. To use them do the following:

1. switch on the Terabit (VM-820).
2. switch on the video player.
3. if a SONY tape player is used, set remote select switch on the back side of the tapeplayer to 300 position.

On VAX

(load 'tapefns)
(vp-open)
--- whatever ---
(vp-close)

On LM1

(load 'tapefnslm)
(with-open-file (*video* "sduserial-b:")
  (send *video* ':set-baud-rate 9600)
  --- whatever ---
)

Some times this won't work due to the serial-port being opened by someone and forgot to close. In that case use the following:

(with-open-file (*video* (steal-port-b))
  (send *video* ':set-baud-rate 9600)
  ---- whatever ---- )

On Symbolics

(load 'discfnslm)
  ---- whatever ---- )
Commands

Following is a description of each of the functions that can be called from lisp. All the commands return an integer code. A value of 0 indicates success. For other values, an error message is printed and an integer between 1 and 9 is returned.

**Name:** vp-retid
**Arguments:** none
**Description:** vp-retid is always called (internally) after the execution of each command. This function returns an integer indicating the status. A value of 0 indicates success. For any value other than 0, an appropriate error message is printed and an integer is returned.

**Name:** vp-open
**Arguments:** none
**Description:** Video player open (only on VAX). Enables device communication.

**Name:** vp-close
**Arguments:** none
**Description:** Video player close (only on VAX). Disables device communication.

**Name:** vp-home
**Arguments:** none
**Description:** Home.

To place VM-820 in a known state. This should be the first command sent to VM-820.

**Name:** vp-initialize
**Arguments:** none
**Description:** Initialize.

Positions the tape at the first valid frame. Audio and video are turned off during this command.

**Name:** vp-quit
**Arguments:** none
**Description:** Quit

Terminates active command.

**Name:** vp-play-forward
**Arguments:** n1 n2
**Description:** Play forward.

Plays the tape from n1 to n2. both n1 and n2 should be integers.

n2 = 0 => plays till the end. (default)
$n_1 < 0 \Rightarrow \text{plays from the beginning frame.}$

$n_1 = 0 \Rightarrow \text{plays from the current frame. (default)}$

**Name:** vp-fast-forward  
**Arguments:** none  
**Description:** Fast forward.  
Fast forwards toward the end of the tape.

**Name:** vp-rewind  
**Arguments:** none  
**Description:** Rewind.  
Rewinds toward the beginning of the tape.

**Name:** vp-go-to-frame  
**Arguments:** n1 n2  
**Description:** Go to a frame number.  
Advances to the frame number $n_1$. (default: beginning)
$n_2$ specifies the video end condition.
$n_2 = 0 \Rightarrow \text{video stops after reaching } n_1$. (default)
$n_2 = 1 \Rightarrow \text{video plays after reaching } n_1$.

**Name:** vp-single-frame  
**Arguments:** none  
**Description:** Single frame.  
Plays forward a single frame.

**Name:** vp-frame-number  
**Arguments:** none  
**Description:** Current frame number.  
Returns the current frame number.

**Name:** vp-hold-enable  
**Arguments:** none  
**Description:** Hold enable.  
Pauses the tape at the current location.

**Name:** vp-hold-disable  
**Arguments:** none  
**Description:** Hold disable.  
Unpauses the tape.

**Name:** vp-last-command  
**Arguments:** none  
**Description:** Last command.  
Returns the last active command.

0 - no active command
1 - vd-FF (fast forward)
2 - vd-G (Go to frame)
3 - vd-R (reject)
4 - vd-I (initialize)
5 - vd-PF (play forward)
6 - vd-HE (hold enable)
7 - vd-HE (hold disable)
8 - vd-SF (single frame)
9 - vd-FR (rewind).

Name: vp-last-command-status
Arguments: none
Description: Last command status.
Returns the status of the last command.
A value of 0 indicates completion.

Name: vp-environment-read
Arguments: none
(lisp mc can take an optional arg: oneof (video audio1 audio2
videoend continuous)
Description: Environment read.
Returns the present environment setting as an integer.
Returned integer positions = 87654321.
position  description (1-on : 0-off)
8  video
7  audio ch.1
6  audio ch.2
5  video end condition.
4  Continuous Frames.
3,2,1 -- always 0s (not used).
on lisp mc
if noarg, returns a list (videostatus ch1sta' ch2sta' videoendst'
contsta')
if arg returns arg's status as 0/1.

Name: vp-getc
Arguments: arg1
Description: Get the current status of arg1.
Returns the present environment setting of arg1 as t/nil indicating
on/off.
Arg1 should be one of the following:
video, audio1, audio2, videoend, continuous.

Note: only on VAX. For lisp mc, see vp-environment-read.
Name: vp-environment-write
Arguments: n1 n2 n3 n4 n5
Description: Environment write.
To write the environment settings.
   n1 - video (1-on, 0-off)
   n2 - audio ch.1 (1-on, 0-off)
   n3 - audio ch.2 (1-on, 0-off)
   n4 - video end cond. (1-on, 0-off)
   n5 - Continuous frame. (1-on, 0-off)

Name: vp-setc
Arguments: arg1 arg2
Description: Set the condition of arg1 to arg2.
Arg1 should be one of the following:
   video, audio1, audio2, videoend, continuous.
Arg2 should be on/off/1/0.
Returns 0.

Name: vp-mode-computer
Arguments: none
Description: Mode computer.
To set the VM-820 to computer mode.
This is the initial mode after the power on of the VM-820 or a reset.

Name: vp-mode-terminal
Arguments: none
Description: Mode terminal.
To set the VM-820 to terminal mode.

Name: vp-sendnum
Arguments: n1
Description: Sends a number (integer) n1.
vp-sendnum is used to send an integer.

Name: vp-eject-tape
Arguments: none
Description: Eject tape.
Unloads the tape.

Name: vp-putchars
Arguments: n1
Description: send the chars obtained by evaluating n1 (VAX).
lispmc: n1 should be list and sends all the chars in n1.
Used to explicitly send chars corresponding to commands.
Example: (vp-putchars 'FF@) to fast forward the tape.

Note: Carriage return is denoted by @ character.

Name: vp-getchars

Arguments: n1

Description: Reads n1 chars from the video line.
            n1 = 0 implies read till end of line.
            Example: (vp-getchars 4) to read 4 chars from the video line.

Note:
These functions are used to load in the C code and bind LISP functions to the particular C routines. Run once at startup time.

```lisp
(defun mapname (x)
  (cond ((equal x "video") 1)
        ((equal x "audio") 2)
        ((equal x "vga") 3)
        ((equal x "continuous") 5)
        (t (msg "Illegal first argument to vp-setc/getc: "))))

(defun vp-getc (x)
  (case x
    (vga (vp-getc))
    (default (vp-getc (mapname x)))))

(defun vp-setc (x)
  (case x
    (vga (vp-setc))
    (default (vp-setc (mapname x)))))

(defun on-off (st)
  (cond ((equal st "on") 1)
        (t 0)))

(defun vp-go-to-frame (expr)
  (cond (null expr) (vp-go-to-frame (car expr) 0))
  (t (vp-go-to-frame (car expr) (cadr expr)))))

(defun vp-play-forward (expr)
  (cond (null expr) (vp-play-forward (list expr)
          (length (explode cchars))
          (apply 'vectori-byte (explode cchars))))

(defun vp-getchars (cchars)
  (let ((cchars (new-vectori-byte 50 "")))
    (ansi)
    (tl 0)
    (vp-getcs sum cchars)
    (while (and (tl 50) (not (= (rvector-byte cchars tl) 94)))
      (setq ana (cons (rvector-byte cchars tl) ana)
              tl (+ tl 1)))
    (reverse (apply 'append
                        (cond ((= x 94) nil)
                              (t (list (maxana (list x))))))))

(defun vp-help)
  (msg ("The following functions are available")
    "vp-close" "vp-open"
    "vp-one" "vp-quit"
    "vp-become" "vp-ininit"
    "vp-save-forward" "vp-repeat"
    "vp-go-to-frame" "vp-reset-frame"
    "vp-bold-enable" "vp-bold-disable"
    "vp-restore-frame" "vp-frame-number"
    "vp-bold-enable" "vp-bold-disable"
    "vp-last-command-status" "vp-setting-term"
    "vp-environment-read" "vp-setc"
    "vp-environment-write" "vp-setc"
    "vp-mode-computer" "vp-setc"
    "vp-sound-aux" "vp-getcs"
    "vp-getch" "vp-getchars")
```
/*
 * vplay.c
 * Loaded in by the lisp calls in 'tapefa.l'; these are the actual
 * routines that interface to the tape players. All lisp functions
 * call these to do their work. Compile with:
 * cc -O -c vplay.c
 */

#include <asygty.h>
#define RETURN "-0.05"
static int video;
static char buff[251];
/* to open the video player */
void open()
{
  int SetTerm();
  if ((video = open("/dev/tty0", 1)) < 0) return (-1);
  SetTerm();
  return(0);
}
/* to close the video player */
void close()
{
  int SetTerm();
  SetTerm();
  close(video);
  return(0);
}
struct asgtyb oldmodes, newmodes;
/* to set the line to rawmode */
void SetTerm()
{
  ioctl(video, TIOCGETP, oldmodes);
  ioctl(video, TIOCGETP, newmodes);
  newmodes sq_flags = 480;
  newmodes sq_speed = 9600;
  newmodes sq_speed = 38400;
  ioctl(video, TIOCSET, newmodes);
}
/* to set the line back to normal mode */
void ResetTerm()
{
  ioctl(video, TIOCSET, oldmodes);
}
/* to get the return id and print the err msg, if any */
int v_readid()
{
  int c, nl;
  read(video, buff, nl);
  switch (buff[nl])
  {
    case '0': return(0);
    case '1': printf("[Illegal Command for this machine: 1c to tcn]\n", buff[0], buff[1], buff[2]);
      return(1);
    case '2': printf("Bad command form: 2c to tcn\n", buff[0], buff[1], buff[2]);
      return(2);
    case '3': printf("Bad option form: 3c to tcn\n", buff[0], buff[1], buff[2]);
      return(3);
    case '4': printf("Last active command is still active: 4c to tcn\n", buff[0], buff[1], buff[2]);
      return(4);
    case '5': printf("Command failed: 5c to tcn\n", buff[0], buff[1], buff[2]);
      return(5);
    case '6': printf("Bad machine code: 6c to tcn\n", buff[0], buff[1], buff[2]);
      return(6);
    case '7': printf("Bad machine code: 7c to tcn\n", buff[0], buff[1], buff[2]);
      return(7);
    case '8': printf("8c to tcn\n", buff[0], buff[1], buff[2]);
      return(8);
    case '9': printf("Returning undefined code: 9c to tcn\n", buff[0], buff[1]);
      return(9);
    default:
      printf("Invalid return code, 0c to tn\n", buff[0], buff[1], buff[2]);
      return(10);
  }
  return(11);
}
/* to change 'b' to integer */
int t_to_int(char *a)
{
  int i;
  int l = 0;
  int n = 0;
  while (*a != 0)
  {
    if (*a == '+') ++n;
    elif (*a == '-') --n;
    else
    {
      a = 10 * a + *a - 0;
    }
  }
  return(n);
}
/* to send a number */
int v_sendnum(char *a)
{
  int b1 = 14;
  int y;
  char buff[15];
/* start forward */
int f forward(
    /* start reverse */
int f reverse(
    int nld, int nld)
    /* initialize */
int f initialize(
    int nld, int nld)
    /* play forward */
int f play forward(
    int nld, int nld)
    /* reject */
int f reject(
    int nld)
int v_retid();
buffer = video_buf, return(v_retid());
/* single step forward a frame */
int v_retid();
buffer[0] = 'S';
buffer[1] = 'E';
buffer[2] = return(v_retid());
/* some VM 020 */
int v_retid();
buffer[0] = 'S';
buffer[1] = 'E';
buffer[2] = return(v_retid());
/* environment read */
int v_retid();
buffer[0] = 'E';
buffer[1] = 'S';
buffer[2] = return(v_retid());
/* environment write */
int v_retid();
buffer[0] = 'S';
buffer[1] = 'E';
buffer[2] = 'E';
buffer[3] = return(v_retid());
write(video_buff[4]);
return(v_retid());
} 

int v_v_setc()
{ int v_v_setc(al, a2)
int "al", "a2";
int v_v_retid()
buff[0] = 'E'
buff[1] = 'V'
buff[2] = 'T'

/* Case 1 */
buff[3] = 'Y'
buff[4] = 'L'
break
/* Case 2 */
buff[3] = 'Y'
buff[4] = 'L'
break
/* Case 3 */
buff[3] = 'Y'
buff[4] = 'L'
break
/* Case 4 */
buff[3] = 'V'
buff[4] = 'L'
break
/* Case 5 */
buff[3] = 'V'
buff[4] = 'L'
break

buff[3] = (al2 == 0) ? '0' : '1'
buff[5] = return;
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_L()
{ int v_L()

buff[0] = 'L'
buff[1] = 'L'
return;
/* if(video_buff,) */
write(video_buff[1]);
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_MC()
{ int v_MC()

buff[0] = 'M'
buff[1] = 'C'
buff[2] = return;
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_NT()
{ int v_NT()

buff[0] = 'N'
buff[1] = 'T'
return;
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_c_f()
{ int v_c_f()

buff[0] = 'C'
buff[1] = 'F'
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_q()
{ int v_q()

buff[0] = 'Q'
buff[1] = return;
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_C()
{ int v_C()

buff[0] = 'C'
buff[1] = return;
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_p_t()
{ int v_p_t()

buff[0] = 'P'
buff[1] = return;
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_p_tch)
{ int v_p_tch)

buff[0] = 'T'
buff[1] = return;
/* if(video_buff,) */
write(video_buff[1]);
return(v_retid());
}

int v_sum
{ int v_sum

char *carry.
/* if at all return is there, it would only be at the end */
if (carry == '0')
    carry = '0' RETURN;
/* least (video, GL), */
write (video, carry, 'sum');
}
/* v_getch command */
v_getch(sum, carry)
int *sum;
char *carry.
int i = 1,
char c;
typ;
pt = carry.
if (*sum == 0)
    while (read (video, c, 1) == 1)
        if (c == ' ') pt = 0;
    else return (0);
else for (i = 0, i < *sum, read (video, pt++, 1), i++)
    pt;
/* v_flush command */
v_flush()
char c;
while (read (video, c, 1) == 1).
**This is the tsp. driver for the LAMBDA LISP machine, started by loading this package. This package is COMMON LISP compatible.**

```lisp
(defvar '01230' OXJQ)
(defvar '01312' 0X3R 3)
(defvar '013-1K' 0X38)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
(defvar '013-1K' 0X54)
```

```lisp
;isp
(defun getcchar()optional &rest)
  (defun vp-getchar(char))
    ; write all the chars in the lis
    ( Pasadena lis) ((lisp (caddr a1)
      (lisp (caddr a1)
    )
    
    ; flush all the buffers and reset the port so the user can reset
    ; if in case something goes wrong.
    
    ; to get the return id and print the err msg if any
    (defun vp-retidoptional (ml n))
      (defun getcchar()optional &rest)
        (defun vp-getchar(char))
          (defun getcchar()optional &rest)
```
(defmacro vp-getchar (s) (length s))

(to change char to integer)

(do (cond ((null s) 0)
          ((< (char s) 32) 0)
          ((< (char s) 127) (char s))
          ((< (char s) 256) (char s))
          ((< (char s) 65536) (char s))
          (t (error "Character Too Large")))

... to send a number /
(apply vp-sendnum (list (char s)))

... fast forward /
(apply vp-fast-forward (list (char s)))

... fast reverse /
(apply vp-rewind (list (char s)))

... go to a frame number nl & al is on off flag /
(apply vp-go-to-frame (list (char s) (char s)))

... bold selectable /
(apply vp-bold-enable (list (char s)))

... bold selectable /
(apply vp-bold-disable (list (char s)))

... initialize /
(apply vp-initialize (list (char s)))

... play forward...
... nl is frame number to start, al is frame number to end
... 0 = starts at the current position, (default)
... 0 = start from beginning,
... 0 = stops at the end (default),
... 0 = stops at that frame number,
(apply vp-play-forward (list (char s) (char s) (char s)))

... reject
(apply vp-reject-tape)

... single step forward & frame /
(apply vp-single-frame)

... home /
(apply vp-home)

... environment read /
... argument nl can be video, audio, audio2, videoed, continuous /
(apply vp-environment-read (list (char s) (char s)))

... second (should be second)
(apply vp-second)
(defmacro vp-mode-computer)
  (prog)
    (vp-putchar (list 'CM 'UC 'ENTER))
    (vp-retid))

(defvar vp-mode-terminal)

(defvar vp-frame-number)

(defvar vp-quit)

(defvar vp-last-command-status)

(defvar vp-help)

(setq N 1) """"The following functions are available """"

(defvar vp-set-opts)

(defvar vp-getchar)

(defvar vp-getenv)

(defvar vp-setenv)

(defvar vp-environment)

(defvar vp-stack)

(defvar vp-stream)

(defvar vp-buffer)

(defvar vp-env)

(defvar vp-set)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)

(defvar vp-set-option)
VIDEO DISK PLAYER
Usage

The video disc can be operated from lisp. The routines are written in C. These routines should be loaded into lisp before using them. To use them do the following:

set the EXT switch to 'on' on the backside of the video player. switch on the video player.

On VAX

(load 'discfns)
(vd-open)
--- whatever ---
(vd-close)

On LMI

(load 'discfnslm)
(with-open-file (*video* "sdu:serial-b:")
  (send *video* ':set-baud-rate 1200)
  --- whatever ---
)

Some times this won't work due to the serial-port being opened by someone and forgot to close. In that case use the following:

(with-open-file (*video* (steal-port-b))
  (send *video* ':set-baud-rate 1200)
  ---- whatever ----
)

On Symbolics

(load 'discfnslm)
(with-open-stream (*video* (si:make-serial-stream :unit 2
  :baud 1200 :check-over-run-errors t :ascii-characters t
  :force-output t))
  ---- whatever ----)
Commands

Following is a description of each of these functions that can be called from lisp. All the commands return an integer code. A returned value of 0 always indicates success. For other values, it prints an error message and returns an integer between 1 and 9.

Name: vd-open
Arguments: none
Description: vd-open enables device communication (only on VAX).

Returns 0.

Name: vd-close
Arguments: none
Description: vd-close disables the device communication (only on VAX).

Returns 0.

Extended Commands

Extended commands are a combination of the basic commands used often. They are:

Name: vd-search-frame
Arguments: n1
Description: Searches for the frame n1 and stills there.

n1 should be 0 <= n1 <= 54000.

Returns 0.

Name: vd-search-segment
Arguments: n1
Description: Searches for the beginning of segment n1 and stills there.

n1 - should be 0 <= n1 <= 63

Returns 0.

Name: vd-search-frame-repeat
Arguments: n1 n2 n3 n4
Description: Searches for frame n1 and repeats n3 times, playing frames n1 to n2 at speeds specified by n4.

n1 - beginning frame number
n2 - ending frame number
n3 - number of times to repeat (lm:default 1)
n4 = 0 - normal speed (lm:default)
   = 1 - fast
   = -1 - slow

if n1 > n2 then direction is backward,
else direction is forward.
n1 and n2 should be within 0 and 54000.
n3 should be within 0 and 15.
Returns 0.

Name: vd-search-segment-repeat
Arguments: n1 n2 n3 n4
Description: Searches for segment n1 and repeats n3 times, playing segment n1 to n2 at speeds specified by n4.
   n1 - beginning segment number
   n2 - ending segment number
   n3 - number of times to repeat (lm:default 1)
   n4 = 0 - normal speed (lm:default)
       = 1 - fast
       = -1 - slow
if n1 > n2 then direction is backward,
else direction is forward.
n1 and n2 should be within 0 and 63.
n3 should be within 0 and 15.
Returns 0.

Name: vd-search-frmsseg-repeat
Arguments: n1 n2 n3 n4
Description: Searches for frame n1 and repeats n3 times, playing n1 to segment n2 at speeds specified by n4.
   n1 - beginning frame number
   n2 - ending segment number
   n3 - number of times to repeat (lm:default 1)
   n4 = 0 - normal speed (lm:default)
       = 1 - fast
       = -1 - slow
n1 should be within 0 and 54000.
n2 should be within 0 and 63.
n3 should be within 0 and 15.
Returns 0.

Name: vd-segsave
Arguments: n1 n2 n3
Description: Sets the segment number n1 to the frames n2 tp n3.
   n1 - segment number to be set.
   n2 - beginning frame number
   n3 - ending frame number.
Returns 0.
Basic Commands

Basic commands are lower-level commands. The user can program using these basic commands according to his needs. Each individual command returns 0 to indicate success. The user has to check the returned code of all the basic commands being used in the user program.

Name: `vd-sendnum`
Arguments: `arg1`
Description: `arg1` should be an integer and should be within some limits imposed by the context.
While referring to a frame number - 0 <= `arg1` <= 54000.
While referring to a segment - 0 <= `arg1` <= 63.
While referring to repetition factor - 0 <= `arg1` <= 15.
Returns 0.

Name: `vd-play`
Arguments: `n1`
Description: Plays in the specified direction until the next command.
Direction is indicated by `n1`.
`n1 = 0` => backward, and `n1 = 1` => forward. (lm:default)
Returns 0.
Stops at the end or beginning of the disc.

Name: `vd-fast`
Arguments: `n1`
Description: Fast forwards/backward the disc.
`n1 = 0` => fast backward.
`n1 = 1` => fast forward (lm:default).
Returns 0.
Stops at the end or beginning of the disc.

Name: `vd-slow`
Arguments: `n1`
Description: `vd-slow` plays slowly in the f/b direction.
The speed is 1/3 of the normal play speed.
`n1 = 0` => play backwards slowly.
`n1 = 1` => play forward slowly (lm:default).
Returns 0.
Stops at the end/beginning of the disc.

Name: `vd-step`
Arguments: `n1`
Description: `vd-fstep` steps one frame at a time in the f/b direction.
`n1 = 0` => step backward.
`n1 = 1` => step forward (lm:default).
Returns 0.
Stops at the end or beginning.

Name: **vd-scan**
Arguments: n1
Description: vd-fscan scans in the f/b direction at a speed of 30 times the normal play speed.
- n1 = 0 => scan backward.
- n1 = 1 => scan forward (lm:default).
Returns 0.
Stops at the end or beginning.

Name: **vd-stop**
Arguments: none
Description: vd-stop stops the active command that is executing.
Returns 0.

Name: **vd-enter**
Arguments: none
Description: vd-enter is used to separate individual commands.
Returns 0.

Name: **vd-ce**
Arguments: none
Description: vd-ce is used to cancel the last input while giving commands.
Returns 0.

Name: **vd-menu**
Arguments: none
Description: Gives a menu on the screen.
The disc should have menu choice recorded in it. (None of the current discs have this.)
Returns 0.

Bugs: Tries to find the menu and may not stop finding.
At times it stops at some frame number.
Use vd-cl to cancel this command, if it seems to be not stopping.

Name: **vd-search**
Arguments: none
Description: This is used to search a frame. Won't work as an individual command. Use vd-search-frame. This is used in combination with other commands to search a frame or a segment.
Example:
```
(vd-search) : search mode
(vd-sendnum n1) ; frame number to be searched
```
(vd-enter) : indicate the completion of the command
Finds frame number n1, and returns 0.

Name: vd-repeat
Arguments: none
Description: This is used in combination with other commands to repeat some frames or segments. Won't work individually.
Example:
(vd-search-frame 1000) ; go to frame number 1000
(vd-repeat) ; repeat mode
(vd-sendnum 2000) ; till 2000th frame
(vd-enter)
(vd-sendnum 2) ; repeat 2 times
(vd-enter)
plays the frames 1000 to 2000, 2 times.
Returns 0.

Name: vd-segment
Arguments: none
Description: Used in combination with other commands.
Example: (vd-segment) ; segment mode
(vd-sendnum 2) ; segment number
(vd-enter)
(vd-sendnum 200) ; beginning frame number
(vd-enter)
(vd-sendnum 300) ; ending frame number
(vd-enter)
Memorizes segment 2 to be from frames 200 to 300.
Returns 0.

Name: vd-ch1
Arguments: n1
Description: vd-ch1 changes the status of audio channel 1.
Depending on the value of n1, it toggles/on/off the status of audio channel 1.
n1 < 0 -- toggles (Im:default)
n1 = 0 -- turns off (regardless of previous status)
n1 > 0 -- turns on (regardless of previous status)
Returns 0.

Name: vd-ch2
Arguments: n1
Description: vd-ch2 changes the status of audio channel 2.
Depending on the value of n1, it toggles/on/off the status of audio channel 1.
n1 < 0 -- toggles (lm:default)
n1 = 0 -- turns off (regardless of previous status)
n1 > 0 -- turns on (regardless of previous status)
Returns 0.

Name: vd-still
Arguments: none
Description: Freezes at the current frame.
Returns 0.

Name: vd-index
Arguments: n1
Description: The index when turned on, displays the current command, status etc in a small rectangle in the upper left corner of the screen.
n1 = -1 => toggles index (lm:default).
n1 = 0 => turns off index.
n1 = 1 => turns on index.
Returns 0.

Name: vd-dumpin, vd-dumpout
Arguments: add1
Description: add1 is the address of a block containing 1024 bytes. These are used to send/receive a block of 1024 bytes. Won’t send any return code in acknowledgement. Have to send 1025 bytes and get error code as an acknowledgement (returned code).
Returns error code.

Name: vd-cl
Arguments: none
Description: vd-cl clears the previous command. This is used when the previous command has errors in it.
example: (vd-search-frame 1000)
       (vd-search-frame 2000)
The second command won’t be executed if enough time is not lapsed between the first and second search. To clear the second command use vd-cl.

Name: vd-pgm, vd-pgmend, vd-run, vd-end
Arguments: none
Description: Used to start, run and end a program.
Can do the programming in lisp.

Name: vd-memory
Arguments: none
Description: Memorizes the current frame, which can be called later by vd-msearch.
Returns 0.
example: (vd-play 1)

say we are at 1202th frame.
(vd-memory) memorizes this frame.
& the play continues and stops at the end.

Now (vd-msearch) would take back to 1202th frame.
vd-memory can remember only one number.
Always the current number replaces the previous number.

Name: vd-msearch
Arguments: none
Description: Used to find the previously memorized frame.
Returns 0.

Name: vd-skip
Arguments: none
Description: Used to skip a frame. Usually used after vd-still to look at the next frame.
Returns 0.

Name: vd-int
Arguments: none
Description: Initializes the player, puts the head at the beginning of the disc and turns on audio channels 1&2.
Returns 0.

Name: vd-review
Arguments: none
Description: Its a control instruction. Returns 0.

Name: vd-mode
Arguments: n1
Description: Depending on the value of n1 it sets the mode.
   n1 = -1 toggles between frame mode and segment mode (lm:default)
   n1 = 0 sets to frame mode.
   n1 = 1 sets to segment mode.
Returns 0.

Name: vd-continue
Arguments: none
Description: Used to continue the previous operation after a vd-still.
Returns 0.
Name: vd-motor
Arguments: n1
Description: Turns the motor on/off
n1 = 0 => off
n1 = 1 => on (in: default)
Returns 0.
This set of LISP calls reads in and binds LISP function names to the C code which actually does the work. Run once at startup time.

```lisp
(defun _open 'vd-open "integer-function")
;;;
(defun _close 'vd-close "integer-function")
(defun _begin 'vd-set "integer-function")
(defun _fastterm 'vd-reset "integer-function")
(defun _sendum 'vd-sendum "integer-function")
(defun _send "vd-send "integer-function")
;...
```
This set of C functions is loaded by the LISP code is "discofs.1" and does the actual work of the LISP functions. Compile with:

```
cc -O -c vdis.c
```

```
#include <stdio.h>
#define MEMPL 256
#define MEMPL 256

struct video {
  int width, height;
  int pitch;
  int xoffset, yoffset;
  int xorig, yorig;
  int numcolors;
  int numplanes;
  int depth;
  int bytesperpixel;
};

struct video Colombia {
  int width, height;
  int pitch;
  int xoffset, yoffset;
  int xorig, yorig;
  int numcolors;
  int numplanes;
  int depth;
  int bytesperpixel;
};

struct video Vicon {
  int width, height;
  int pitch;
  int xoffset, yoffset;
  int xorig, yorig;
  int numcolors;
  int numplanes;
  int depth;
  int bytesperpixel;
};

struct video Vicon {
  int width, height;
  int pitch;
  int xoffset, yoffset;
  int xorig, yorig;
  int numcolors;
  int numplanes;
  int depth;
  int bytesperpixel;
};
```

```c
/*
 * Struct video
 * 
 * Define the structure for video parameters.
 * 
 * Fields:
 * 
 * width, height: Dimensions of the video.
 * pitch: Number of pixels in a line.
 * xoffset, yoffset: Offsets from the origin.
 * xorig, yorig: Origins of the video.
 * numcolors: Number of colors.
 * numplanes: Number of planes.
 * depth: Depth of each pixel.
 * bytesperpixel: Number of bytes per pixel.
 */

struct video {
  int width, height;
  int pitch;
  int xoffset, yoffset;
  int xorig, yorig;
  int numcolors;
  int numplanes;
  int depth;
  int bytesperpixel;
};

/*
 * Struct video Colombia
 * 
 * Define the structure for video parameters.
 * 
 * Fields:
 * 
 * width, height: Dimensions of the video.
 * pitch: Number of pixels in a line.
 * xoffset, yoffset: Offsets from the origin.
 * xorig, yorig: Origins of the video.
 * numcolors: Number of colors.
 * numplanes: Number of planes.
 * depth: Depth of each pixel.
 * bytesperpixel: Number of bytes per pixel.
 */

struct video Colombia {
  int width, height;
  int pitch;
  int xoffset, yoffset;
  int xorig, yorig;
  int numcolors;
  int numplanes;
  int depth;
  int bytesperpixel;
};

/*
 * Struct video Vicon
 * 
 * Define the structure for video parameters.
 * 
 * Fields:
 * 
 * width, height: Dimensions of the video.
 * pitch: Number of pixels in a line.
 * xoffset, yoffset: Offsets from the origin.
 * xorig, yorig: Origins of the video.
 * numcolors: Number of colors.
 * numplanes: Number of planes.
 * depth: Depth of each pixel.
 * bytesperpixel: Number of bytes per pixel.
 */

struct video Vicon {
  int width, height;
  int pitch;
  int xoffset, yoffset;
  int xorig, yorig;
  int numcolors;
  int numplanes;
  int depth;
  int bytesperpixel;
};
```
case UNERR:
    case UACE:
    case UCMP:
    case UPOKE:
        return(0);
    case CNTAB:
        printf("number not found\n"),
            return(1);
    case UPOKE:
        printf("invalid command\n"),
            return(2);
    case UNERR:
        printf("command not in valid range\n"),
            return(1);
    default:
        printf("returning undefined code : 0\n", "buf");
            return(5);

let mputchar() {
    case x:
        let y = return(x);
        return(y);
    case y:
        let z = return(y);
        return(z);
    case z:
        let w = return(z);
        return(w);
    default:
        return(-1);
}

let t: commands:
let e:
let b:
let y:
let f:
let g:
let h:
let i:
let j:
let k:
let l:
let m:
let n:
let o:
let p:
let q:
let r:
let s:
let t:
let u:
let v:
let w:
let x:
let y:
let z:

let _main() {
    let main:
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
        return(main);
    let main()
return(rl);
}

int v eyeb() {
    return mputcher();
}

int v menu() {
    return mputcher(UMENU);
}

int v search() {
    return mputcher(USEAR);
}

int v repeat() {
    return mputcher(UREP);
}

int v segment() {
    return mputcher(USEGM);
}

int v chk(n)
int 'a' {
    return mputcher(UCCHL);
}

int v chk2(n)
int 'b' {
    return mputcher(UCCH2);
}

int v_index(n)
int 'c' {
    return mputcher(UCDIK);
}

int v dumpin() {
    return mputcher(UDUMP);
}

int v dumpout() {
    return mputcher(UDUMP);
}

int v cli() {
    return mputcher(UCLI);
}

int v pgn() {
    return mputcher(UPGN);
}

int v rna() {
    return mputcher(UKRN);
}

int v end() {
    return mputcher(USEN);
}

int v memory() {
    return mputcher(UMEM);
}

int v mesearch() {
    return mputcher(UMSEA);
}

int v skip() {
    return mputcher(USKIP);
}

int v list() {
    return mputcher(ULIST);
}
```c
int v_review()
{
    int v_putchar();
    return(v_putchar();)
}

int v_mode(sl)
int *sl,
{
    int v_putchar();
    return(v_putchar("al > 0") UMODE
            "al == 0") UFRAME : USMODE);
}

int v_controle()
{
    int v_putchar();
    return(v_putchar();)
}

int v_motor(sl)
int *sl,
{
    int v_putchar();
    return(v_putchar("al > 0") UMODE : USMODE);
}

int v_sleep(sl)
int *sl,
{
    "al = ("al < 0") > "al = "sl.
    sleep();
    return(0);
}

int v_search_frame(sum)
int *SUM,
{
    int rs_putchar();
    v�판dictionary();
    if (num < 0 : num > 54000) return(4).
    if (rs v_putchar(USEAK) > 0) return(rz).
    if (rs v_putchar(USEAK1) > 0) return(rz)
    1 = rs_putchar(VENTE);
    sleep();;
    return(rz);
}

int v_search_sequence(sum)
int *SUM,
{
    int rs_putchar();
    v�판dictionary();
    if (num < 0 : num > 63) return(4).
    if (rs v_putchar(USEAK) > 0) return(rz).
    if (rs v_putchar(USEAK1) > 0) return(rz).
    if (rs v_putchar(VENTE) > 0) return(rz).
    if (rs v_putchar(VENTE1) > 0) return(rz);
    sleep();;
    return(rz);
}

int v_search_frame_repeat(sum, sum2, sum3, sum4, sum5)
int *sum, *sum2, *sum3, *sum4, *sum5,
{
    int rs_putchar();
    v�판dictionary();
    if (sum < 0 : sum > 63) return(4).
    if (rs v_putchar(USEAK) > 0) return(rz).
    if (rs v_putchar(USEAK1) > 0) return(rz).
    if (rs v_putchar(USEAK) > 0) return(rz).
    if (rs v_putchar(USEAK1) > 0) return(rz).
    if (rs v_putchar(VENTE) > 0) return(rz).
    if (rs v_putchar(VENTE1) > 0) return(rz).
    if (rs v_putchar(VENTE2) > 0) return(rz).
    return(rz);
}
```

if (rst = mputchar(UNITE) > 0) return(rst).
if (rst = m_sequadum(states) > 0) return(rst).
return(mputchar(UNITE)).

int v_segsave(al, sl, a3)
int "al", "a3", "a3",

int st, v_sequadum(), mputchar().
if ("sl" > 5 || "sl" > 6)
{
    printf("Illegal segment number : %d\n", "sl").
    return(4)
}
if ("a2" > 54000 || "a3" > 54000)
{
    printf("Illegal frame number : 54000 \n").
    return(4)
}
if (rst = mputchar(USEG)) > 0) return(rst).
if (rst = v_sequadum(sl) > 0) return(rst).
if (rst = mputchar(UNITE) > 0) return(rst).
if (rst = v_sequadum(a3) > 0) return(rst).
}
This is the package of functions which interfaces to the disk player from the LAMBDA machine. It is COMMON LISP compatible.

(defvar *UNILL* (x00))
(defvar *XOIP* (x02))
(defvar *XIPI* (x04))
(defvar *XO5* (x05))
(defvar *XO6* (x06))
(defvar *XO7* (x07))
(defvar *XO8* (x08))
(defvar *XO9* (x09))

(defun (defunser -or-
 (dof (daYsIr-e-le-devices)
 a
 (cond ((string-equal "SOU-SERIAL-B", (send day :none))
 (send dey :allocate t)
 (rep eca (send dcv :lock)
 nil)
 (return dev))
 (defun vsdemli
 (video "sdv-serial-b:" (Send -video-':Sot-baud-rate 9600)
 (loop (print (oval (read)))))))
 (defun idi:cdemosym
 (vsit- open-strean (.,iao- (silmake-serlel-stream
 -unit 2
 :beud.9600
 :choc -oie:r-run-:rrors t
 .eSciiti-c ract rs t
 force-output t
 .t(3
 (loop (359
 (N 1)
 (eval (reeSd)((
 .~(defunmygetchar()
 (ca
 (ZL:QxPioden (read-cher)((
 (def:so sygetcher
 (en*idso'
 (dunep!utchar(etm 4OPtiOnal
 (0 O((;sleep
 time before
 reading again
 (megur!
 (L'
 (asii tm
 (N I"(
 )
 Or
 n;
 O id-sleep
 no)...
 (vd-retid((
 (dsfuD
 OyputCbAr(Chx &Optioneli(n
 Oflelee4p time before reading in again
 a~(ed) video'
 tyo chrO
 Or
 I)
 n o)(sepn
 C
 (id-rei)
 (defies id-reset))
 ...
 flueS all the
 buffers
 and reset
 the
 Port so the user
 can
 reset
 if
 is
 c
 esm
 thing
 goes
 Wrng.
 (dofus
 oyqetchst#)
(defun v-search-frame-nosequence (top)
  (if (or (< top 0) (> top 54000))
      (return top))
  (if (self top (myputchar "OCTE-")) 0 (return top))
  (if (self top (vd-sendnum top)) 0 (return top))
  (return (vd-enter 2))))

(defun v-search-segment-repeat (fnum same optional (attns 1) (speed 0))
  (prog (top)
    (if (< fnum 0) (setf fnum 0))
    (if (< fnum 63) (setf fnum 63))
    (if (< same 0) (setf same 0))
    (if (< same 63) (setf same 63))
    (if (self top (vd-search-segment fnum)) 0 (return top))
    (cond ((< speed 0)
      (if (self top (myputchar (if (> top same) "UFSLO" "OFSLO") 0)
        (return top)))
      (if (self top (myputchar (if (> top same) "UFSLO" "OFSLO") 0)
        (return top)))
      (if (self top (myputchar "UENTU") 0 (return top))
        (if (self top (vd-sendnum same)) 0 (return top))
        (return (vd-enter 2))))
  (defun v-enter (speed)
    (if (self top (myputchar (if (> top same) "UFSLO" "OFSLO") 0)
      (return top)))
    (if (self top (myputchar (if (> top same) "UFSLO" "OFSLO") 0)
      (return top)))
    (if (self top (myputchar (if (> top same) "UFSLO" "OFSLO") 0)
      (return top)))
    (if (self top (vd-sendnum same)) 0 (return top))
    (return (vd-enter 2))))

(defun v-search-segment (same)
  (prog (top)
    (if (or (< same 0) (> same 63))
      (max 1 "Invali segment number " same)
      (max 1 "should be with in 0 and 63 ")
      (return top)))
    (if (< same 0) (setf same 0))
    (if (< same 63) (setf same 63))
    (if (self top (vd-search-segment same)) 0 (return top))
    (if (self top (vd-sendnum same)) 0 (return top))
    (return (vd-enter 2))))

(defun v-help()
  (max (N L) " myputchar")
(msg (N l) - ep名 sue")
  (msg (N l) - vd"cat")
  (msg (N l) - vd"ch")
  (msg (N l) - vd"cl")
  (msg (N l) - vd"close")
  (msg (N l) - vd"continue")
  (msg (N l) - vd"dump")
  (msg (N l) - vd"dumpout")
  (msg (N l) - vd"end")
  (msg (N l) - vd"exit")
  (msg (N l) - vd"fast")
  (msg (N l) - vd"load")
  (msg (N l) - vd"last")
  (msg (N l) - vd"memory")
  (msg (N l) - vd"mode")
  (msg (N l) - vd"move")
  (msg (N l) - vd"message")
  (msg (N l) - vd"open")
  (msg (N l) - vd"ping")
  (msg (N l) - vd"play")
  (msg (N l) - vd"repeat")
  (msg (N l) - vd"recall")
  (msg (N l) - vd"review")
  (msg (N l) - vd"run")
  (msg (N l) - vd"scope")
  (msg (N l) - vd"search\{frame\}"
  (msg (N l) - vd"search\{frame-repeat\}"
  (msg (N l) - vd"search\{mang-repeat\}"
  (msg (N l) - vd"search\{segment\}"
  (msg (N l) - vd"search-segment-repeat")
  (msg (N l) - vd"save")
  (msg (N l) - vd"saveout")
  (msg (N l) - vd"sleep")
  (msg (N l) - vd"slow")
  (msg (N l) - vd"step")
  (msg (N l) - vd"still")
  (msg (N l) - vd"stop")
  (msg (N l) - vd"zero")
LA50 PRINTER
This package allows the user to print screen dumps to the 1x60 printer connected to SOU serial port B. Any rectangular portion of the screen may be dumped. The printer is not the fastest in the world.

(defun steal-port-bo)
(dolist :dev ais'all-shared-devices)
  (cond ((string-equal "SOU-serial-b" dev)
    (send dev 'allocate) nil))
  (return dev))))

(let* (,(swd (+ 6 (ceiling wd 6))))
  (nh wd)
  (awar-t (make-array (list nh wd) :type 'rt-lb :initial-value 80))
  (tcb swd nhl nhl)
  (copy-array-contents arhd awar-t)
  (with-open-file (ipSO "sdu-serial-b" :direction :io)
    (send ipSO 'set-baud-rate 4800)
    (send ipSO 'typool)
    .. escape
    (send ipSO 'typool)
    .. P
    (send ipSO 'typool)
    .. Q
    (send ipSO 'typool)
    .. form feed
    (global dotimes (i (ceiling nh 6))
      (setf nh t i 61))
    (global dotimes (j (ceiling wd 6))
      .. for each column of 6 pixels
      (setf swd (+ 1 j 6))
      . for each row of (= 712 pixels
      (global dotimes (k (min 500 (- wd nhl)))
        (setq awd2 (+ swd2 k))
        (tcb (get-num (aref awar-t swd2) nhl)
          (aref awar-t (+ 1 swd2) nhl)
          (aref awar-t (+ 2 swd2) nhl)
          (aref awar-t (+ 3 swd2) nhl)
          (aref awar-t (+ 4 swd2) nhl)
          (aref awar-t (+ 5 swd2) nhl))
        (send ipSO 'typool)
        (send ipSO 'typool))
        (twice because of the aspect ratio 1:2)
        (send ipSO 'typool)
        (send ipSO 'typool))
        (next line)
        (send (zerop (mod shil 131)))
        (send ipSO 'typool)
        (send ipSO 'typool)))
    (global dotimes (k (min 500 (- wd nhl)))
      (setq swd2 (+ swd2 k))
      (tcb (get-num (aref awar-t nhl swd2)
          (aref awar-t (+ 1 nhl) swd2)
          (aref awar-t (+ 2 nhl) swd2)
          (aref awar-t (+ 3 nhl) swd2)
          (aref awar-t (+ 4 nhl) swd2)
          (aref awar-t (+ 5 nhl) swd2))
      (send ipSO 'typool)
      (send ipSO 'typool))
      (twice because of the aspect ratio 1:2)
      (send ipSO 'typool)
      (send ipSO 'typool))
      (next line)
      (send (zerop (mod shil 131)))
      (send ipSO 'typool)
      (send ipSO 'typool)))
    (global dotimes (k (min 500 (- wd swd2)))
      (setq swd2 (+ swd2 k))
      (tcb (get-num (aref awar-t swd2 nhl)
          (aref awar-t (+ 1 nhl) swd2)
          (aref awar-t (+ 2 nhl) swd2)
          (aref awar-t (+ 3 nhl) swd2)
          (aref awar-t (+ 4 nhl) swd2)
          (aref awar-t (+ 5 nhl) swd2)))
      (send ipSO 'typool)
      (send ipSO 'typool))
      (twice because of the aspect ratio 1:2)
      (send ipSO 'typool)
      (send ipSO 'typool))
      (next line)
      (send (zerop (mod nh 131)))
      (send ipSO 'typool)
      (send ipSO 'typool)))
    (global dotimes (k (min 500 (- nh swd2)))
      (setq swd2 (+ swd2 k))
      (tcb (get-num (aref awar-t swd2 nhl)
          (aref awar-t (+ 1 nhl) swd2)
          (aref awar-t (+ 2 nhl) swd2)
          (aref awar-t (+ 3 nhl) swd2)
          (aref awar-t (+ 4 nhl) swd2)
          (aref awar-t (+ 5 nhl) swd2))))
      (send ipSO 'typool)
      (send ipSO 'typool))
      (twice because of the aspect ratio 1:2)
      (send ipSO 'typool)
      (send ipSO 'typool))
      (next line)
      (send (zerop (mod nh 131)))
      (send ipSO 'typool)
      (send ipSO 'typool))
This set of functions allows the user to interface to the RATES displays in a uniform manner. All functions are used directly in the user’s C programs by including this file at the top of the user’s source with:

```c
#include crtk.c
```

or by initially compiling with:

```c
cc -0 -c crtk.c
```
and including it on the compile call line as:

```c
cc yourprog.o crtk.o
```

```c
static char buff[4096];
static int ramtek;
```

```c
r_open(orientation)
set orientation.
```
buf[10] = length, length, 0xff. buf[11] = 0x00, if (vals = 22 * length: 1) val:++.
  ' number of bytes for write
  bptr = &buf[12];
  while (length-- * bptr) = *textptr;
  if (write ram text, buf, val = write) return (-1);
  return (0).
}

write_values, load_table, start, ovall, select_table
char *values;
int load_table, start, ovall, select_table;

register 1, j, k, l = 0;
if (val = { /* Then must load before selecting */
  'Load Auxiliary Memory command -- Device 3 --
  buf[10] = 0x00,
  buf[11] = 0x00,
  buffer start address -- 16bit start, must entry start --
  buf[2] = (start = 1),
  buf[3] = (l = 1) -- (load_table = 1) = 0xff.
  'Number of bytes to load -- 4 times number of entries --
  buf[4] = (l = ovall = 1) = 0xff.
  buf[5] = (l = 1) = 0x00.
  j = 4,
  for (l = 2: l = ovall ++ )
  buff[2] = 'values -- * Blue
  buff[3] = 'values -- * Green
  buff[4] = 'values -- * Red
  buff[5] = 'values -- * And any blank

  k = 0;
  'Either zero or (4 = (ovall ' 4)) *
  buff[2] = 0x00.
  'Select table
  buff[4] = 0x00.
  'Table number
  buff[5] = 0x00.
  select_table -- (select_table = 2) = 0x00.
  buff[6] = 0, 'Just selecting so no actual writing ...
  buff[7] = 0.
  if (write ram text, buff, k = k) return (-1).
  return (select_table).
}

r_image (image data, ext start, ystart, numpts, scan_style, operation)

char *image data;
int xstart, ystart, numpts, scan_style, operation;

register 1, j, numpts;
1 = 0;
buf[1] = 0x00.
buf[2] = 0x00.
'Write Image instruction /
buf[3] = 0x00.
'OP1 -- Scan...
buf[4] = 0x00.
'OP2 -- Image Mode...
buf[5] = 0x00.
'Nothing
buf[6] = 0x00.
'Image Mode
buf[7] = 0x00.
'Select scan & xoff
buf[8] = 0x00.
'Function
buf[9] = 0x00.
'Operation & xoff
buf[10] = 0x00.
COP at first
buf[11] = (xstart = xoff)
buf[12] = (ystart = xoff)
buf[13] = (ystart = yoff)
'Image Mode
buf[14] = 0x00.
buf[15] = 0x00.
buf[16] = numpts & numpts & 0xff.
buf[17] = (numpts >> 8) & 0x00.

/*.
while (numpts--)

if (write ram text, buff, 1 = 1) return (-1).
return (0).

r_rectangle, y)
int x, y;

{ r rectangle, x, y, x+175, y+70, 4, 1, 1, 4, "NAVY CENTER",
  r_text(x, y+48, y+15, 1, 4, "NAVY CENTER"),
  r_text(x, y+48, y+15, 1, 4, "FOR",)
  r_text(x, y+48, y+15, 1, 4, "APPLIED RESEARCH"),
  r_text(x, y+48, y+15, 1, 4, "IN")
  r_text(x,y+50, y+5, 1, 4, "ARTIFICIAL INTELLIGENCE");
}
static char buff[150];
static int ramtek;

r_open()
|
| if((ramtek = open("dev:rtk", O_RDONLY)) < 0) return -1);
| return(0);
|
|

r_close()
|
close(ramtek);
| return(0);
|
|

r_line(x0, y0, xl, yl, color)
sint *x0, *y0, *xl, *yl, color;
|
| buff[0] = 0x02, buff[1] = 0x09,
| buff[2] = 0x04, buff[3] = 0x00,
| buff[4] = 0x06, buff[5] = 0x00,
| buff[6] = 0x08, buff[7] = 0x00,
| buff[8] = 0x0A, buff[9] = 0x00,
| buff[13] = 0x0F, buff[14] = 0x00,
| buff[15] = 0x0F, buff[16] = 0x00,

if(write(ramtek, buff, 19) == 19) return -1);
| return(0);
|
|

r_erase()
|
| buff[0] = 0x02, buff[1] = 0x09,
| buff[2] = 0x04, buff[3] = 0x01,
| buff[4] = 0x06, buff[5] = 0x00,
| buff[6] = 0x08, buff[7] = 0x00,
| buff[8] = 0x0A, buff[9] = 0x00,
| buff[13] = 0x0F, buff[14] = 0x00,
| buff[15] = 0x0F, buff[16] = 0x00,
| if(write(ramtek, buff, 16) == 16) return -1);
| return(0);
|
|

erase()
|
| buff[0] = 0x02, buff[1] = 0x09,
| buff[2] = 0x04, buff[3] = 0x00,
| buff[4] = 0x06, buff[5] = 0x00,
| buff[6] = 0x08, buff[7] = 0x00,
| buff[8] = 0x0A, buff[9] = 0x00,
| buff[13] = 0x0F, buff[14] = 0x00,
| buff[15] = 0x0F, buff[16] = 0x00,
| if(write(ramtek, buff, 14) == 14) return -1);
| return(0);
|
|

r_reset()
|
| buff[0] = 0x00, buff[1] = 0x05,
| if(write(ramtek, buff, 2) == 2) return -1);
| return(0);
|
|

text(x0, y0, color, textptr)
sint *x0, *y0, *color;
| char *textptr;
|
| register int文字. length;
| register char *bptr;
| buff[0] = 0x08, buff[1] = 0x0A,
| buff[2] = 0x04, buff[3] = 0x00,
| buff[4] = 0x06, buff[5] = 0x00,
| buff[6] = 0x08, buff[7] = 0x00,
| buff[8] = 0x0A, buff[9] = 0x00,
| buff[10] = 0x00, buff[11] = 0x00,
| buff[12] = 0x0F, buff[13] = 0x01,
| buff[14] = 0x0F, buff[15] = 0x00,
| buff[16] = 0x0F, buff[17] = 0x00,
| if(length > 0) return -1);
| if(length == 0) length = 0;
| if(write(ramtek, buff, 19) == 19) return -1;
| return(0);
END

4 - 1 - 87

DTTC