Lion Machine Software

7956 Rio Vista Dr. Goleta, California 93117

LION MACHINE

HP-IL DEVELOPERS TOOL KIT

OWNERS REFERENCE MANUAL

LION MACHINE 7956 Rio Vista Dr. Goleta Ca. 93117 (805) 685-2296

Getting started...

You may be wondering just what you have bought here and how to use it. These are the questions that we are going to try to answer. First here are the requirements necessary to use this software package.

1) An IBM PC/XT/AT or close compatible to run the software on.

2) This computer must have a HP-IL interface adapter card installed in it. Either Hewlett Packard's or Interloop's adapter card will be fine. Installation will load the device driver interrupt $54 in the CONFIG.SYS file. This driver is necessary for the loop routines to function. To install the adapter you should consult the owners manual of the adapter card you have.

3) Turbo Pascal 4.0 this package will not work with Turbo Pascal 3.0. Well.. you can modify it to work with 3.0 but you might as well get a copy of 4.0 since it is sooo.. much nicer!

4) Ability to write programs in Pascal. If you don't think you have the ability, then a desire to learn Pascal will be fine.

A quick word on Turbo Pascal 4.0 and pre-compiled units.. Turbo Pascal 4.0 supports libraries of pre-compiled procedures. These libraries are called units. The units are stored on disk and are available to the programmer when she writes programs. If you find that you use a group of procedures in most of your programs then you can group them into a unit and compile it. Now you can use them in your programs by just adding them in your "uses" statement. For example if you have some string handling routines CentJust, RightJust, LeftJust & StripBlank and they are compiled in the unit StrInc. You can use these routines in any program you write by adding the line..

```
uses
   StrInc;
```

at the top of the file. Now these procedures are available to use. The Turbo Pascal system keeps track of which unit has been modified and re-compiles them as necessary. This relieves the programmer of worrying about the code being up to date.

Units are divided into two sections, interface and implementation. The interface section has all of the constants, type defs, variables functions and procedures that calling programs can access. The implementation section contains the code that is hidden from the calling programs. Section (1) & (2) of this manual contain information about the interface part of the Lion Machine HP-IL tool kit. This information is necessary to be able to use this code properly. Section (3) contains information about the implementation section. This section is here for the interested user. Not all of the routines contained here are used by the Tool Kit but the source code is there in case you would like to modify it. Section (2) contains information for the Interloop IO port. This can be ignored if you have the Hewlett-Packard interface card since that card has no IO port.

The Lion Machine HP-IL Developers Tool Kit is a pre-compiled unit called HPILInc. It consists of all of the procedures, functions, variables and constants necessary for controlling HP-IL devices. It is

intended to be copied into your programming library. Once copied into your library the loop controlling routines can be used in the programs that you develop. Here is an example of a loop control program. This program called Plist will output a text file to the first HP-IL printer on the loop. This makes a handy utility program.

```
program Plist;

{
This is a simple demo program that will prompt for a text file then
print it out to the first HP-IL printer it can find on the loop.
if no printer is found (the printer must be able to respond to an
Accessory search), then the program is terminated with an error
message.
}

   uses
      HPILInc;

   var
      InFile : text;
      Line : String;
      PrinterAddr : integer;


   function GetFile(var InFile:text) : boolean;

   {
   GetFile will open up an input text file if possible. GetFile
   returns true if successful or false if not.
   }

      var
         FileName : string;
         FileOk : boolean;

      begin
         FileOk := false;                { Assume failure.       }
         writeln;                        { Prompt for file name. }
         write('File name : ');
         readln(FileName);
         assign(InFile,FileName);
         {$I-}
         reset(Infile);                  { Try a reset...        }
         {$I+}
         if IOresult=0 then              { Test file...          }
            FileOk := true               { Have valid file.      }
         else begin                      { Test Failed.          }
            writeln;
            write('Can not open ',FileName);
         end;
         GetFile := FileOk;
      end;
```

```
function SetUpLoop(var PrinterAddr:integer) : boolean;

{
SetUpLoop will configure the Loop and find a printer. SetUpLoop
returns true if successful or false if not.
}

   var
      LoopOk : boolean;

   begin
      LoopOk := false;                        { Assume failure.      }
      Restoreio;                              { Set up loop.         }
      if LoopError<>0 then
         writeln('Loop error')                { Loop did not work.   }
      else begin
         Clear;                        {  Loop worked, Reset devices. }
         PrinterAddr := AccAddr(Printer);  {  Look for printer.   }
         if (PrinterAddr=NullAddr) then
            writeln('Can''t find printer on loop.') { No printer!}
         else
            LoopOk := true;          { Loop's up and found printer. }
      end;
      SetUpLoop := LoopOk;
   end;


begin
   if GetFile(InFile) then begin    { Have valid file, Set up loop.}
      if SetUpLoop(PrinterAddr) then begin {Loop's up, Print file.}
         while ((not EOF(InFile))and(LoopError=0)) do begin
            readln(InFile,Line);
            Output(PrinterAddr,Line);
         end;
         if LoopError<>0 then
            writeln('Loop broken...');  { Finished with an error. }
      end;
      close(Infile);
   end;
end.
```

Section (1) Program accessible loop control interface.

To use the HP-IL Tool Kit to its fullest extent you should read and understand this section. This section explains all of the procedures, constants, and variables that can be used by your HP-IL controlling programs.

Variables & Constants

Loop Error flag :

```
Possible Loop Errors
   1 : Output Error
   2 : Enter Error
   4 : Configuration error

LoopError : byte;
```

Optional string terminator :

This is the string variable that is added to the end of output strings as an end of data message. Default value is <CR><LF>.

```
EndStr : String;
```

Loop Time Out Value :

This is the number of seconds before setting loop time out error. This value can be changed at compile time to suite the system.

```
TimeOut  = 15;
```

Null address value :

Any function that returns an address will return a -1 or "NullAddr" if unsuccessful. This will not cause "LoopError" to be set because there is no error, just a device can not be found. Procedures that use address as input will not execute if Address = "NullAddr".

```
NullAddr =  -1;
```

Accessory ID pre-defined values :

This block of constants are used for finding devices on the loop by accessory ID. For example to find a printer on the loop the statement Addr := AccAddr(Printer); will return the address of the first printer on the loop to the integer variable "Addr".

```
Controller  =   15
MassMem     =   31
Printer     =   47
Display     =   63
IntfceDev   =   79
ElectInst   =   95
GraphDev     =  111
AnalitInst  =   127
```

RestoreIO procedure :

RestoreIO should be run once to configure the loop before attempting any loop operations. This will assign addresses to devices on the loop, set the loop time out, and put the loop in a known state with the PC as loop controller. If successful, the variable "LoopError" will be cleared. If not successful, "LoopError" will be set to 4.

*** NOTES ON ADDRESSING ***

Address 1 - 7 are reserved for HP-IL to HP-IB interfaces. This was decided by Hewlett-Packard when they designed the HP-IL interface card, what this means to you is...

1. HP-IL loop address will start at 8.

2. HP-IL to HP-IB interfaces must have their address set in the range of 1 - 7.

3. This does not affect devices that do not support auto addressing. If a device does not have auto addressing, it will be addressed by its built in default address.

```
procedure RestoreIO;

   var
      OldTime : integer;

   begin
      OldTime := time(TimeOut);
      if not Config(TimeOut) then
         LoopError := 4
      else
         LoopError := 0;
   end;
```

Clear procedure :

Clear will cause the Device clear message to be sent out to every device on the loop. Device clear does not affect the way a device communicates on the loop, it sends out a pre-defined reset signal. This signal will reset printers, clear buffers, set devices to initial states and things like that. The response that a device has to the clear message is entirely up to the device's designer. This means that it can vary widely from device to device. You should consult the owners manuals for your loop devices to see what if any response your devices will have to this command.

```
procedure Clear;

  var
     Error : boolean;

  begin
    Error := SendFrame(1044,0);
  end;
```

DevAddr, NextDevAddr functions :

DevAddr returns the address of a device on the loop given its I.D. name "DevName". If the device is not found, an address of -1 or "NullAddr" is returned. NextDevAddr will find the next device on the loop corresponding to the Last "DevName". Device ID's are an optional function of loop devices. This means that they might or might not support this function. Device names are usually the device model number or company name.

```
function DevAddr(DevName:String) : integer;

  var
     Addr : integer;

  begin
     LastDevName := DevName; { Setup to find next Device on loop. }
     LastDevNum  := 2;
     Addr := FindID(DevName,1);
     if Addr = 0 then
        Addr := NullAddr;
     DevAddr := Addr;
  end;
```

```
function NextDevAddr : integer;

   var
      Addr : integer;

   begin
      Addr := NullAddr;
      if (LastDevNum>1)and(LastDevNum<=30) then begin
         Addr := FindID(LastDevName,LastDevNum);
         if Addr = 0 then
            Addr := NullAddr
         else
            LastDevNum := LastDevNum + 1;
      end;
      NextDevAddr := Addr;
   end;
```

AccAddr, NextAccAddr functions :

AccAddr will search the loop for a class match of accessory ID "AccNum" and return the address of the first occurrence of that accessory. If no match is made, the address of -1 or "NullAddr" is returned. This function is handy for use when writing code for printers or video displays. This can tell you the address of your printer and display so you can send output to them automatically. NextAccAddr will give back the next device on the loop that corresponds to the AccNum last used.

HP-IL accessory types as defined by Hewlett-Packard are...

```
0   - 15   Controller
16  - 31   Mass storage device
32  - 47   Printer
48  - 63   Display
64  - 79   Interface Device
80  - 95   Electronic instrument
96  - 111  Graphics Device
112 - 127  Analytical instrument
```

If you want just a class match send the highest number then only a class match will take place i.e. for the first occurrence of any printer, send 47 and you will get the address of the first printer.

```
function AccAddr(AccNum:integer) : integer;

   var
      Addr : integer;

   begin
      Addr := Find(1,AccNum);
      if Addr = $1F then
         Addr := NullAddr;
      LastAccAddr := Addr;  { Set up for finding next device. }
      LastAccNum := AccNum;
      AccAddr := Addr;
   end;
```

```
function NextAccAddr : integer;

   var
      Addr : integer;

   begin
      Addr := NullAddr;
      if LastAccAddr<>NullAddr then begin
         Addr := Find(LastAccAddr+1,LastAccNum);
         if Addr = $1F then
            Addr := NullAddr;
         LastAccAddr := Addr;
      end;
      NextAccAddr := Addr;
   end;
```

DevID function :

DevID returns the device name at the address "Addr". An address value of "NullAddr" is ignored. Any error will return a null string. If a loop error occurs then "LoopError" of 2 will be set.

```
function DevID(Addr:integer):String;

   var
      OutStr : string;

   begin
      OutStr := ''; { Assume failure. }
      if (Addr<>NullAddr) then begin
         OutStr[0] := chr(lo(input(Addr,OutStr[1],MaxData,$562)));
         if length(OutStr)=0 then
            LoopError := 2;
      end;
      DevID := OutStr;
   end;
```

OutPut procedure :

OutPut sends data to a device on the loop. "Addr" is the Device address on the loop and "DataStr" is the data to be sent to that address. An address value of "NullAddr" is ignored. If a loop error occurs the "LoopError" will be set to 1.

```
procedure Output(Addr:integer;DataStr:String);

   var
      NumBytes : byte;

   begin
      if (Addr<>NullAddr) then begin
         DataStr := DataStr + EndStr;
         NumBytes := LoopOut(Addr,length(DataStr),DataStr[1],true);
         if NumBytes<>Length(DataStr) then
            LoopError := 1;  {data transmission error}
      end;
   end;
```

Enter function :

Enter returns data from a device on the loop. "Addr" is the address of the device on the loop to receive data from. An address value of "NullAddr" is ignored. Any error will return a null string. If a loop error occurs then "LoopError" of 2 will be set.

```
function Enter(Addr:integer):String;

   var
      OutStr : string;

   begin
      OutStr := ''; { Assume failure. }
      if (Addr<>NullAddr) then begin
         OutStr[0] := chr(lo(input(Addr,OutStr[1],MaxData,$560)));
         if length(OutStr)=0 then
            LoopError := 2;
      end;
      Enter := OutStr;
   end;
```

SendStatus function :

SendStatus returns the Status of the device at address "Addr". An address value of "NullAddr" is ignored. Any error will return a null string. If a loop error occurs then "LoopError" of 2 will be set.

```
function  SendStatus(Addr:integer) : String;

   var
      OutStr : string;

   begin
      OutStr := ''; { Assume failure. }
      if (Addr<>NullAddr) then begin
         OutStr[0] := chr(lo(input(Addr,OutStr[1],MaxData,$561)));
         if length(OutStr)=0 then
            LoopError := 2;
      end;
      SendStatus := OutStr;
   end;
```

Section (2) The Interloop IO port interface.

To use the Interloop IO port you should read and understand this section. This section explains all of the procedures, constants, and variables that can be used by your programs to control The Interloop IO Port. If you have an Hewlett-Packard HP-IL adapter card instead of the Interloop HP-IL adapter card you can safely skip this section. This is because the Hewlett-Packard HP-IL adapter card has no IO Port.

Variables & Constants

Setting the port vector :

To use the Interloop IO port the address that Model# 150 HP-IL interface card is installed at must be known and stored in hexadecimal form in the variable "HPILCardAddr". Interloop defaults this address at 300H. This address may be changed because of address conflicts when the card was installed. If your system has a HP-IL card installed and running you can find out the address by checking the CONFIG.SYS file where its HPIL.SYS file is installed. If the address is different than the default address, it will be shown as a parameter /Axxx where xxx is the address in hex. An example for setting the Interloop default address of 300 would be the statement: HPILInc.HPILCardAddr := $300;

```
HPILCardAddr : LongInt;
```

Reading & Writing to the port.

Writing to the output port is accomplished by setting the variables Out4, Out5, Out6 to either true or false as you want them, then by calling the WritePort procedure. Reading the input port is accomplished by first calling the procedure ReadPort then reading the values from In6 and In7.

```
Out4, Out5, Out6, In6, In7 : boolean;
```

Procedures & Functions :

ReadPort Procedure :

ReadPort uses HPILCardAddr to load the boolean variables In6 and In7 with the corresponding values from In6 and In7 of the IO Port.

```
procedure ReadPort;

   var
      InNum : word;

   begin
      if HPILCardAddr <> nullAddr then begin
         InNum := Port[HPILCardAddr+7];
         if (128 and InNum)=0 then
            In7 := true
         else
            In7 := false;
         if (64 and InNum)=0 then
            In6 := true
         else
            In6 := false;
      end;
   end;
```

WritePort Procedure :

WritePort uses HPILCardAddr to set the IO port outputs to equal the boolean variables Out4, Out5, Out6.

```
  procedure WritePort;

     var
        OutNum : word;

     begin
        if (HPILCardAddr<>NullAddr) then begin
           OutNum := 0;
           if Out6 then
              OutNum := OutNum or 64;
           if Out5 then
              OutNum := OutNum or 32;
           if Out4 then
              OutNum := OutNum or 16;
           Port[HPILCardAddr+7] := OutNum;
        end;
     end;
```

Section (3) Non accessible low level code.

This section is purely optional. All of the Type definitions, constants, variables, procedures, and functions listed here are "invisible" to programs using the HP-IL Tool Kit. This code is only "visible" to the procedures and functions contained in the HP-IL Tool Kit. This section can be most useful to programmers that need greater control over HP-IL operation that the tool Kit offers. It is included for those that would like to know more about the inner workings of the HP-IL Developers Tool Kit so they can modify it.

Used for the input buffer size.

```
MaxData     = 255;
```

Calling DOS interrupt procedures :

The HPIL.SYS drivers that come with the Hewlett-Packard and Interloop HP-IL interface cards are invoked by DOS interrupt calls. Interrupt procedures access data through the CPU registers. Turbo Pascal's method for accessing the CPU registers is through a variable of type DOS.registers. To invoke an interrupt procedure load a variable of type DOS.registers with your input data then call the necessary interrupt procedure. After the interrupt procedure has finished the ending state of the processor's registers will be found in the DOS.registers type variable. These registers hold the output of the interrupt procedure. "Regs" is the variable used to pass data to the CPU registers.

```
Regs : DOS.registers;
```

Variables used for multiple device searches :

Variables declared in a Turbo Pascal unit's implementation section can only be accessed by the unit itself and are protected from being overwritten by other code from the calling program. These features make it possible to save the state of a unit from call to call. That is what these variables below are for: saving the state of the unit between calls to allow for multiple searches from one data input.

```
LastDevNum, LastAccAddr, LastAccNum : LongInt;
LastDevName : string;
```

Initialization Code

Along with the power to store information from call to call is the ability to initialize values before execution of the main program. Here is the code use by the Lion Machine HP-IL Tool Kit for initialization. This procedure sets up all of the default values for the unit's variables.

```
begin
   EndStr := Char(13) + Char(10); { <CR><LF> Sequence          }

   LastDevNum := 1;               { Set up for multiple        }
   LastAccAddr := NullAddr;       { device and accessory       }
                                  { searches.                  }

   LoopError := 0;                { Assume no errors yet.      }

   HPILCardAddr := NullAddr;      { Set up for using IO port.  }
end.
```

Note that the comments sections end with a number such as 5-73 which refers to the appropriate page in the Technical Reference Manual for the HP Portable Plus.

Time function :

```
*************************************************************************
This function sets the time_out to a new value and returns the old
value

Secs is an integer that is the number of seconds to wait before
declaring a time out error.  The value returned is old time out if
loop ok or 0 if not.

Set Time out 5-76
*************************************************************************

function Time(secs : integer) : integer;

   begin
      if secs <=0 then
         secs:=1
      else if secs > 4095 then
         secs:=4095;
      Regs.ax:=$0A00;
      Regs.bx:=secs shl 4; { intr $54 uses time in 1/16ths of a sec. }
      intr($54,Regs);
      if (Regs.flags and 1) = 1 then
         Time:=0
      else
         Time:=Regs.bx shr 4;
   end;
```

ConFig function :

```
*************************************************************************
 This function places the loop in a known state and assigns addresses.

 Secs is an integer that is the number of seconds to wait before
 declaring a time out error.  Config will be set to true if the loop
is
 ok, else false.

 Configure Loop 5-69
*************************************************************************

function Config(secs : integer) : boolean;

   begin
      config:=false;     { assuming failure }
      Regs.ax:=$00;      { command to configure the loop }
      intr($54,regs);    { call hpil primitives }
      if (Regs.flags and 1) = 0 then
         if time(secs) <> 0 then
            config:=true;
```

```
   end;
```

Find function :

```
*************************************************************************
This function searches the loop starting at the address STARTADDR
looking for a device with an accessory id of ID.  If the ID is $xF,
only a class match is performed (only the top four bits are compared
with the device accessory IDs).  If a matching device is found, its
address is returned else 1F is returned.

Find Device 5-70
*************************************************************************

function Find(StartAddr, id : integer) : integer;

   begin
      Find := $1F;          { assume failure }
      if id<>254 then begin
        Regs.ax:=$0100;  { command for find device }
         Regs.bx:=(StartAddr shl 8) or id;
         intr($54,Regs);
         if (Regs.flags and 1)=0 then
          Find:=lo(Regs.bx);  { return address }
      end;
   end;
```

GetID function :

```
*************************************************************************
This function returns the accessory ID of the HP-IL device at the
address specified.  Several conditions exist that will cause a value
of $FE (254) to be returned:
   1) Addressed device does not support accessory ID.
   2) Addressed device does not exist.
   3) Addressed device has an accessory ID of $FE.

$FE is the id for an extended class, general device.  A program that
needs to control a device with this ID will require some other means
to determine if the device is on the loop. valid address is (00
through 30).  If id not found $FE returned.

Get Accessory ID 5-71
*************************************************************************

function GetID(addr : integer) : integer;

   begin
      GetID:=$FE;     { assume failure }
      if addr in [0..30] then
         begin
            Regs.ax:=$0200;
            Regs.bx:=addr;
            intr($54,Regs);
            if (Regs.flags and 1)=0 then
               GetID:=lo(Regs.bx);
         end;
   end;
```

Input function :

```
**************************************************************************
```
This function receives characters from a device on the loop.

Addr is the address of the device to receive data from. Buff is a buffer area to put the data. MaxBytes is the maximum number of bytes to receive. Option can have the following values:

```
     $560      -- send data
     $561      -- send status
     $562      -- send device id.
     $563      -- Send accessory id.
     0         -- No SOT frame is sent.
```

The functional return is the actual number of bytes transferred, or 0 if an error.

Note that MaxBytes is an integer that is from 0-255.

Note that "buff" can be any type variable (integer, string, etc. and that the bytes received will start at the first byte of buff. If you have the data going into a string variable, it will start at position 0, not 1 unless you define the variable to be the first position. For example, if "line" is a string, data could fill line by specifying "line[1]" as the variable "buff". Note that in this case you should set line[0] as the return from this function.

Example - `line[0]:=chr(lo(input(addr,line[1],MaxBytes,option)));`

This takes care of the string and the length.

```
Address          5-72
Input Data Block 5-74
**************************************************************************
```

```
function Input(addr:integer;var buff;MaxBytes,option:integer):integer;

   begin
      Input:=0;   { assume failure }
      Regs.ax:=$0300;
      Regs.bx:=(addr shl 8) or 31;{ talk addr and my listen addr(31) }
      intr($54,Regs);
      if (Regs.flags and 1)=0 then
         begin
            Regs.cx:=MaxBytes;
            if (option>=$560) and (option<=$563) then
               Regs.bx:=option
            else
               Regs.bx:=0;
            Regs.es:=seg(buff);     { segment that buff is in }
            Regs.di:=ofs(buff);     { offset into segment of buff }
            Regs.ax:=$0500;
            intr($54,Regs);
            if (Regs.flags and 1)=0 then
               Input:=Regs.cx;  { bytes received }
         end;
   end;
```

LoopOut function :

```
**********************************************************************
```
This function sends characters to a device on the loop.

Addr is the address of the device to send data to. Buff is a buffer area that has the data. len is the number of bytes to send (see note in function above. The functional return is the actual number of bytes transferred, or 0 if an error.

See notes for "buff" above in "input" function

Address 5-72
Output Data Block 5-73

```
**********************************************************************

function LoopOut(addr,len:integer;var buff;EndOpt:boolean): integer;

   begin
      LoopOut:=0;   { assume failure }
      Regs.ax:=$0300;
      Regs.bx:=(31 shl 8) or addr;{ my talk addr (31) and addr listen}
      intr($54,Regs);
      if (Regs.flags and 1)=0 then
         begin                        { no error }
            Regs.cx:=len;
            Regs.dx:=ord(EndOpt); { finish with END frame if desired }
            Regs.es:=seg(buff);   { segment that buff is in }
            Regs.di:=ofs(buff);   { offset into segment of buff }
            Regs.ax:=$0400;
            intr($54,Regs);
            if (Regs.flags and 1)=0 then
               LoopOut:=Regs.cx;    { bytes sent }
         end;
   end;
```

GetFrame function :

```
**********************************************************************
```
This function waits for a frame to be received. If no frame is available from the HP-IL interface then 00 is returned. If an error occurs -1 is returned.

Get Frame 5-75

```
**********************************************************************

function Getframe : integer;

  begin
     Regs.ax:=$0700;
     intr($54,Regs);
     if (Regs.flags and 1)=0 then
        GetFrame:=Regs.bx and 2047
     else
        GetFrame:=-1;
  end;
```

SendFrame function :

```
*************************************************************************
This function will send the specified frame over the loop and an error
code is returned.  Frame is the frame value (0..2047) to be sent.  The
values and their usage are defined in the HP-IL interface
specification manual.  the options supported are:
       0 : wait for loop ready before transmit.
       1 : No wait before transmit.

Send Frame  5-75
*************************************************************************

function SendFrame(Frame,Option:integer) : boolean;

   begin
      Regs.ax:=$0600;
      Regs.bx:=frame and 2047;
      Regs.dx:=option and 1;
      intr($54,Regs);
      SendFrame:=((Regs.flags and 1)=0);
   end;
```

Status function :

```
*************************************************************************
This function returns the current loop status.  If the loop is ready
for a frame to be sent, bit 0 will be set.  If a frame is available,
then bit 1 will be set.  Note that any operation that sends a frame
will effectively erase the frame available bit.  The time out is also
tested if the loop is not ready for a frame, and a time out error is
returned if necessary.

    status bit 15-7 : not used.
           bit 6    : controller active.
           bit 5    : talker active.
           bit 4    : listener active.
           bit 3    : service request received.
           bit 2    : (not used).
           bit 1    : frame available.
           bit 0    : Loop ready for frame.

if an error occurs a -1 is returned.

Status  5-76
*************************************************************************

function Status : integer;

   begin
      Regs.ax:=$0800;
      intr($54,Regs);
      if (Regs.flags and 1)=1 then
         Status:=-1
      else
         Status:=Regs.bx;
   end;
```

FindID function :

```
*****************************************************************
 FindID will find a non-HP-IB device on the loop if it exists by it's
 device ID (i.e. Name) rather than by it's accessory ID. The parameter
 "LoopNum" is the number of the device in the loop. For example, if
you wanted to find the second one of a given device in the loop, then
LoopNum=2. If the device is not found, 0 is returned.
*****************************************************************

function FindID(Name:String;LoopNum:integer) : integer;

   var
    Addr, Num  : byte;
    Line : string;

   begin
     FindID:=0;   {assume failure}
     Num:=0;
     if ((LoopNum<1)and(LoopNum>22)) then
        LoopNum:=1;   {check limit}
     for Addr:=8 to 30 do begin
        line[0]:=chr(lo(input(Addr,line[1],9,$562)));
        if (Regs.flags and 1)<>0 then
           exit;
        if copy(line,1,length(name))=Name then begin
           Num:=succ(Num);
           if num=LoopNum then begin
              FindID:=Addr;
              exit;
           end;
        end;
     end;
  end;
```

Appendix

Where do I find out more about HP-IL ?

A good introduction to HP-IL can be found in the book:

THE HP-IL SYSTEM:
An Introductory Guide to the Hewlett-Packard Interface Loop
by
Gerry Kane, Steve Harper & David Ushijima

Another source of information is the Hewlett-Packard HP-IL technical reference manual available from Hewlett-Packard.

Where can I buy HP-IL devices ?

Hewlett-Packard supplies a complete line of scientific and engineering devices that communicate with HP-IL. To order Hewlett-Packard and other computer equipment you can contact:

Educalc
27953 Cabot Road
Laguna Niguel, Ca. 92677
(714) 582-2637

Lewis & Lewis
1600 Callens Road
Ventura, Ca. 93003
(800) 324-3607

or
Hewlett-Packard sails in your area.

Interloop carries an entire line of Step motor drivers, I/O interfaces, HP-IL repeaters, and other HP-IL products.

Interloop
706 Charcot Avenue,
San Jose Ca. 95131
(408) 922-0520

Semifusion Co. Carries a line of high performance robotic servo motors that communicate with HP-IL.

Semifusion Co.
2352 Walsh Avenue,
Santa Clara Ca. 95051
(408) 748-8416

Lion Machine Software

7956 Rio Vista Dr. Goleta, California 93117

THE

LION MACHINE

HP-IL DEVELOPERS TOOL KIT

WHAT IS IT?

HP-IL DEVELOPER'S TOOL KIT

What is it ?

The HP-IL Tool Kit is a software tool for developing Pascal programs to control HP-IL devices with an IBM PC/XT/AT or close compatible. The Tool Kit consists of several input/output and searching functions that can be added directly into Turbo Pascal 4.0 programs to control HP-IL devices. These functions and procedures should be sufficient for building most HP-IL systems. The source code and complete documentation is supplied for users that would like a greater knowledge of HP-IL or need greater control of HP-IL.

What is its purpose ?

The Tool Kit was designed as a software tool to relieve the system developer from the intricacy of HP-IL protocol. The Tool Kit used as is gives the programmer the power to auto configure the HP-IL communication loop, find devices on the loop by I.D. or device type, input and output data, input device status and reset devices. This allows the programmer to quickly build systems using HP-IL devices.

How does it work ?

The Tool Kit is a set of Pascal routines that control the HPIL.SYS system supplied with the HP-IL interface card from Hewlett Packard. The Tool Kit is written in the form of a pre-compiled Turbo Pascal 4.0 unit library file. To use the Tool Kit you must first have an HP-IL interface card and its HPIL.SYS system properly installed. To create programs that operate HP-IL all that is done is to include HPILInc in your "uses" statement. This is done by typing...

```
uses
   HPILInc;
```

at the top of any Pascal code file that needs to use the HP-IL routines. This is all explained in your Turbo Pascal 4.0 owners manual.

The Tool kit offers these pre-compiled HP-IL routines..

* RestoreIO	Auto configures HP-IL.
* Clear	Resets loop devices.
* DevAddr & NextDevAddr	Locates devices by name.
* AccAddr & NextAccAddr	Locates devices by type.
* DeviceID	Reads device names.
* Output	Sends Data to devices.
* Enter	Reads Data from Devices.
* SendStatus	Reads status from devices.

Also included are the driver procedures for the Interloop model # 150 IL interface IO port.

* WritePort	Toggles IO port bits.
* ReadPort	Reads IO port bits.

All source code and documentation is included for easy modifications and changes.

Lion Machine Software

7956 Rio Vista Dr. Goleta, California 93117

THE

LION MACHINE

HP-IL DEVELOPER TOOL KIT

WHY BUY IT ?

A lot of talk goes around about robotics, computer automated machine control, etc. Talk like this is exciting. It's fun to dream of all of the wild things that can be made by hooking computers up to machines. But when the talk turns to how these things are going to be accomplished, you will only see a lot of arm waving and hear generalizations like, "Well.. we'll just hook a computer to it". The fact of the matter is that for the most part, unless you had a large expense account, or you loved assembly language and were an electronic wizard, these wonderful ideas just weren't possible.

This is no longer the case. The Lion Machine HP-IL Developers Tool Kit allows programmers to control any HP-IL device from Borland Turbo Pascal 4.0. programs. Using the Tool Kit you can develop programs that will...

* Auto configure HP-IL loop system.

* Reset peripheral loop devices.

* Find devices on loop according to their type, printer, display, electronic instrument, etc.

* Find devices by their individual I.D. names.

* Enter data from sensing devices.

* Enter status from devices.

* Output data to devices.

* Output instructions to machine control devices.

* Tie into other software and Turbo Pascal tool kits for calculations and data formatting.

Other features of the Lion Machine HP-IL Tool Kit...

* Compatible with both the Hewlett Packard and the Interloop HP-IL interface adapter cards.

* Runs on all of the IBM PC/XT/AT and close compatible computers.

* Complete source code. Perfect for users that would like a greater knowledge of HP-IL or need greater control over HP-IL systems.

* No royalties charged for programs developed with the Lion Machine HP-IL Tool Kit.

SCIENTIFIC APPLICATIONS :

HP-IL was designed with the scientist in mind. Hewlett Packard supports a entire range of scientific test equipment that communicates with HP-IL. Using Turbo Pascal 4.0's powerful DOS calling routines linked with the control of peripheral equipment, not only can you control scientific test equipment, you can tie into other programs to output and format your data. This type of computing power will give you professional results previously unavailable with calculators and hand held computers.

* Control existing scientific test equipment easily through Turbo Pascal 4.0.

* Quickly construct custom scientific test systems by using off-the-shelf equipment and Turbo Pascal 4.0.

* Link "in the field" data gathering equipment to the more powerful IBM PC computer for data interpretation.

* Link scientific data with existing IBM PC graphics and numerical analysis software for better presentations and reports.

COMPUTER AUTOMATED MACHINE CONTROL :

Until now most computer automated machine control was tied down to assembly language and hard wiring. This is no longer the case. There is a wide variety of machine control devices presently on the market that communicate with HP-IL. Machines built around devices like these can have their function changed quickly and easily by altering Pascal programs instead of re-coding in assembly language or electrical redesign.

* Fast and inexpensive development of complex mechanical systems through use of off-the-shelf components.

* Simplified machine function modifications and additions through alterations in high level Turbo Pascal 4.0.

* Simplified mechanical design and longer life through use of computer logic instead of mechanical linkages.

* Tie into existing software packages for extended machine applications, e.g. Statistical analysis during operation.

* Tight machine control with 2K+ bits per sec. information flow.

FOR THE HOME :

The home user benefits because HP-IL was designed for low cost systems. For example : A system could be built that would control the temperature of your home as well as log and graph energy usage. After the initial system is up and running, hooking it up to control your pool's heating system would consist of a little more Pascal code. Getting it to water the plants would cost the price of some electric valves. Using HP-IL and off-the-shelf electronic devices, one could control everything in the home at a very reasonable cost. If you are the type of hobbyist that likes to tinker with electronics and computers, this is the product for you. With this product, you can let your imagination run wild.

* Easy to operate. Devices just plug onto the loop, no configuration to deal with.

* Low cost. Complex systems can be easily built up on a household budget.

* Flexibility. With some electronic know how, anything you can dream up can be built from off-the-shelf equipment and a little software.

Lion Machine Software
7956 Rio Vista Dr. Goleta, California 93117

WHAT IS HP-IL ?

Hewlett Packard supports a wide array of computer peripherals from printers and displays to electronic and scientific test equipment. They needed an inexpensive way to control all of these devices from their hand held computers and calculators. This is why HP-IL was developed. HP-IL is a system for interfacing electronic devices to a controlling computer by a 2 conductor wire up to 100 meters. The communication line sources at the controlling computer and goes to device #1 then to device #2 and so on until it returns to the controlling computer. This links all of the peripheral devices together in a loop.

BENIFITS OF HP-IL :

* Only one protocol. Unlike RS-232 that has many different ways to communicate, HP-IL only has one protocol. This means that if a device is HP-IL then all you have to do is plug it in. There is no configuration to deal with.

* Multiple devices. Also unlike RS-232 that only allows one device per communication line, HP-IL will allow multiple devices on one communication loop.

* Fast data flow. HP-IL can easily handle 2,000 baud data rate. This is a device dependent value. Theoretical baud rate is 20,000.

* Low cost. HP-IL was designed for low cost computer systems. Multi function digital/analog IO devices cost in the neighborhood of $500-$600, high performance robotic servo controllers price at less than $1,000 per axis.

* Availability of peripheral devices. Any scientific, mechanical or home system can be either bought or easily created from off-the-shelf items.

* Electrical isolation. Devices communicating with HP-IL are electrically isolated from each other to eliminate the risk of power surges from one device harming other devices.