6-1-1984

An Image Processing Software Package for the Laser Scanning Phase Modulation Microscope

William J. Murray
Dartmouth College

M.S.

An Image Processing Software Package for the Laser Scanning Phase Modulation Microscope

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the degree of

Master of Science

by

William J. Murray

Thayer School of Engineering
Dartmouth College
Hanover, New Hampshire 03755

June 1984

Examining Committee:

John W Strohbehn
Chairman

Eric W Hansen

Mark S Sherman

Dwight Zahn
Dean of Graduate Studies

 William J Murray

AN IMAGE PROCESSING SOFTWARE PACKAGE FOR THE LASER SCANNING PHASE MODULATION MICROSCOPE

William J. Murray

Technical Report PCS-TR86-128

Thayer School of Engineering
Dartmouth College

An Image Processing Software Package for the Laser Scanning Phase Modulation Microscope

William J. Murray

M.S. June 1984

ABSTRACT

This thesis documents the most recent effort to develop a user-friendly image processing software package for the Laser Scanning Phase Modulation Microscope (LSPMM). The LSPMM is composed of three integrated subsystems, the Laser Scanning (LS) system, the Phase Modulation (PM) system, and Digital Image Aquisition (DIA) system. Under the control of the image processing software, the DIA system can receive and store the digital image data, display the image on a monochrome monitor, and process the image to provide the microscopist with quantitative information regarding the image. The implementation of this image processing software package required the specification of a four level software hierarchy to serve as an organizational framework, with the highest level interacting with the LSPM microscopist, and the lowest level performing hardware control. This framework should prove useful for the development and implementation of additional software in the future. The programs that were developed accept command line arguments; however, most will interactively query the user if the command line arguments are not known. This software provides the microscopist with the capability to scan, save, and display a 512 by 512 pixel image. The image may be scanned to, saved from, or displayed in either of the two DeAnza image display memory planes. Considerable effort has been made to incorporate all of the devices useful for image processing into a single operating system kernel. This alleviates the problem of taking down one operating system and bringing up another version in order to dump image files on magnetic tape.

ACKNOWLEDGEMENTS

It is hard to recognize all the people who assisted me with the work involved in this thesis, and those who are not noted here are deeply appreciated.

Special thanks is in order for Professor Strohbehn and Professor Hansen for their support and guidance, and Professor Sherman is thanked for reading the thesis on short notice. The financial support of the National Institutes of Health grant number RO1 GM28313 is gratefully acknowledged.

Finally, I would like to thank some of the special people in my life, who offered support and expert advice. My close friends and cohorts Dave Farrington, Jon Panek, and Larry Bohs were always willing to lend a helping hand or encourage extracurricular activities. Kevin Ryan provided the opportunity for trips to Boston when the fresh air became too fresh. I am also grateful to my Aunt Virginia Sheehan and Uncle John MacDonald for their support and encouragement towards achieving my goals. Last, I thank my parents and sisters for their support and patience.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The goal of this thesis was to develop a user-friendly image processing software package for the Laser Scanning Phase Modulation Microscope (LSPMM). Allen, Brault, and Moore introduced phase modulation microscopy as an improvement to conventional image contrast microscopy in 1963 [1]. A phase modulation microscope measures several optical properties including birefringent phase retardation, where retardation is the birefringence times the optical path length through the microscope specimen. Given an incident beam of plane polarized light composed of two orthogonal components, the birefringent phase retardation is the phase delay induced between these components when the light passes through the specimen. The current configuration of the LSPMM is the result of three years of joint effort between research groups at the Thayer School of Engineering and the Dartmouth College Biology Department.

The LSPMM provides the microscopist with the means for accurate measurement of birefringent phase retardation in living specimens, along with the ability to apply digital image processing techniques to images composed of mappings of these measurements to a 512 by 512 pixel area. Each pixel is a gray level between 0 (black) and 255 (white), where the gray level is a discrete representation of the optical property currently being measured by the microscopist.

The LSPMM is composed of three integrated subsystems: the Laser Scanning (LS) system, the Phase Modulation (PM) system, and Digital Image Aquisition (DIA) system. The LS system, described in Pillsbury's thesis, provides the scanning laser beam for specimen illumination, and the timing signals necessary for correct digitization of the image data [15]. The PM system, completed by Farrington in 1984, is an electrooptic system that acquires the quantitative specimen measurement [9]. The DIA system receives and stores the digital image data, displays the image on a monochrome monitor, and processes the image to provide the microscopist with quantitative information regarding the image. The first work on the DIA system was performed by M. Riley [16]. Riley specified the hardware components to be used in this system, designed a hardware double buffer critical to the aquisition of an image, and provided the first programs for aquisition and display of an image. The reader is referred to Riley's thesis for further information [16].

This thesis documents the development of a user-friendly image processing software package for use within the DIA system. The hardware components selected for the DIA system imposed some constraints on the image processing software:

1. The software must operate on a PDP-11/60 minicomputer that serves as a host computer for a DeAnza image display system.
2. The software must be written in the C programming language and run under version 7 of the UNIX operating system.
3. The software must provide the capability to acquire and store a 512 by 512 pixel image in at most one second.

Constraint 1 was met by making minor hardware modifications that are described in Chapter 2. The second constraint was met by writing all of the program code in either the C programming language or the "shell" command language. Both of these languages are supported by our version of the UNIX operating system. The third constraint was first met by M. Riley; thus any modified version of the original "scan" program must also meet the specifications. The modified version of "scan" described in Chapter 6 does meet the third constraint.

Several criteria were considered in the development of "user-friendly" image processing software:

1. Novice microscopists should only need to be familiar with a limited set of commands for use with the LSPMM. Expert knowledge of the UNIX operating system should not be required.

2. The commands should provide an interactive mode that prompts the user for necessary input information.
3. The commands should accept command-line arguments and have reasonable default parameters to minimize the typing necessary to execute a command.
4. Functions and programs should be developed in a structured fashion with each program module performing a logically distinct operation or set of operations. This will provide a set of software tools useful for future program development.
5. Tools should be provided for maintenance of the system software in the event of modification or expansion.

Thirteen commands proposed for implementation are described in Chapter 3. These commands are outlined in order to satisfy the first criterion; however, all of these commands are not implemented in this thesis, because the amount of programming would be beyond the scope of a master's thesis. The second and third criteria were satisfied by providing both command-line and interactive user interfaces in the programs. A four level hierarchy was established to create an organizational framework for the image processing software. The highest level contains the commands used by the microscopist. The second and third contain C programs and C function subprograms respectively. Device driver code and operating system software resides in the lowest level. This hierarchy provided a framework that

was useful in satisfying the fourth criterion. The fifth criterion was satisfied by establishing versions of a "makefile" that are used by the UNIX program maintainer "make" [10]. The "make" command is used to recompile general programs and to regenerate the operating system kernel. If hardware is added to the system, the make files will be useful for regenerating the software. As new software is developed in the future, the make files can be altered with a minimum of difficulty by using the current versions as a model.

Once the four level framework was established, programs were developed at all levels. This required modification of existing programs to fit into the hierarchy in some cases and the creation of new programs in other cases. Specific goals were set to provide the microscopist with certain basic image processing capabilities; they are listed below:

1. The ability to scan an image into either of the two memory planes available within the DeAnza image display system.
2. The ability to save an image from either of the DeAnza image display memory planes in a disk file on the PDP-11/60.
3. The ability to display an image from a disk file in either of the DeAnza memory planes.
4. The ability to use the trackball as a simple image reviewing mechanism for the microscopist,

incorporating the hardware scroll and zoom features of the DeAnza image display system.

These goals were met by developing the programs "scan," "save," "disp," and "tbzoom" contained in the second level of the hierarchy which are incorporated into the commands "scan," "save," "display," and "zoom" in the top level. The programs accept command line arguments in a format resembling standard UNIX commands; however, most will interactively query the user if the command line arguments are not known. The commands created are simple yet versatile; for example, the "tbzoom" program provides the microscopist with either a simple trackball zoom display combination, or trackball zoom display with annotations for cursor position and gray value. Generally, these commands default to the most commonly used capability of the program, with options for other possibilities provided by more complex statements.

The following chapters and appendices describe the implementation of the image processing software. Chapter 2 discusses the modifications made to existing hardware so that the software would run correctly. The software hierarchy is outlined in Chapter 3, and the techniques used in the implementation of this hierarchy are described in Chapter 4. The programs incorporated into hierarchy Level 1 are described in Chapter 5; these are the "commands" currently available to the microscopist. Chapter 6 provides

a description of the Level 2 programs that actually perform the work selected in a Level 1 command. The Level 3 function subprograms are documented in Chapter 7. A discussion of the operating system interface and system regeneration software can be found in Chapter 8. Source code listings and UNIX manual pages are reproduced in the appendices.

Chapter 2: System Hardware and Modifications

2.1 Background Information

This chapter describes the hardware modifications made to the DIA system since it was first described by M. Riley [16]. Modifications were made to both the PDP-11/60 and the DeAnza image display system to allow the trackball device driver software to function correctly. The trackball is a peripheral device that communicates with the PDP-11/60 through the DeAnza image display system via a standard interrupt transaction. A short summary of the hardware components in the DIA system is presented below. This summary is followed by an outline of a standard interrupt transaction, emphasizing the signals that must be present for successful completion.

The input to the DIA system is the analog output from the Phase Modulation subsystem. This signal serves as the input to the analog-to-digital (A/D) converter. Referring to Figure 2.1, the output from the A/D converter is transmitted in parallel (EIA RS-422 standard) to the hardware double buffer inside the PDP-11/60 host computer. The bytes of data are transferred to the DeAnza image display memory through the DR-11W DMA controller. The dual-port, random-access memory within the DeAnza image display system is scanned in a raster pattern and the output is converted to an analog video signal. This video signal

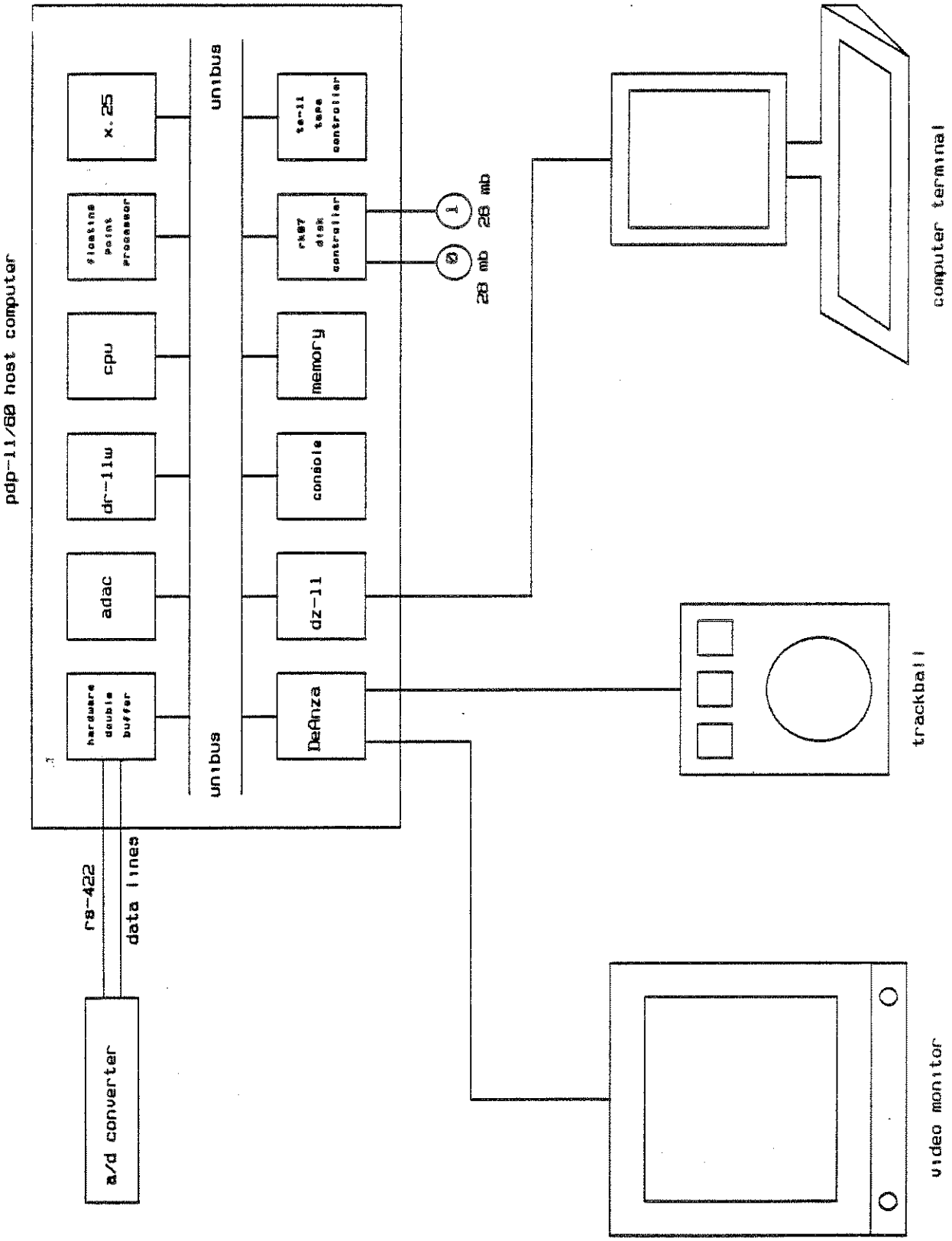


FIGURE 2.1 The Digital Image Acquisition System

serves as the input to a video monitor where the image is viewed by the microscopist. The microscopist can interact with the PDP-11/60 host computer via the terminal to manipulate the image. The trackball provides the microscopist with an easy image reviewing mechanism, allowing interactive scroll and zoom of displayed images. For more detailed information concerning the A/D converter or the Phase Modulation system, refer to D. Farrington's thesis [9]. Further information concerning the hardware double buffer, DR-11W, and the UNIX device driver software may be found in M. Riley's thesis [16].

The PDP-11/60 host computer communicates with peripheral devices through a high-speed bus architecture known as the UNIBUS [6]. Data, address, and control information are transmitted through the 56 lines of the UNIBUS. A peripheral device may interrupt the Central Processing Unit (CPU) when it requires service. This method of servicing the device allows the processor to perform other tasks until the peripheral device needs attention.

The PDP-11/60 provides a two dimensional interrupt priority structure to allow peripheral devices to communicate with the host computer. The interrupt structure consists of 5 vertical priority levels: NPR(non-processor request) the highest level, BR7 (bus request 7), BR6, BR5, and BR4, the lowest level. A horizontal level of priority exists at each vertical level of priority with the device

electrically closest to the processor on any horizontal level having the highest priority.

A typical UNIBUS interrupt transaction follows the pattern below. This pattern is valid for the trackball, which issues a bus request 5 (BR5) and receives a bus grant 5 (BG5). The interrupt vector for the trackball is 0234 octal. SACK (Slave ACKnowledge), BBSY (Bus BuSY), SSYN (Slave SYNchronize), and INTR (INTerrupt Request) are electrical signals on the UNIBUS.

1. A requesting device asserts BR5 which is received by the UNIBUS arbitrator.
2. Given that SACK negation occurs from the previous priority arbitration and that the interrupt fielding processor is ready to accept an interrupt vector at this level, the arbitrator asserts BG5 and the arbitration stops.
3. BG5 is received by the device. The device asserts SACK which is received by the arbitrator.
4. The arbitrator negates BG5 which is received by the requesting device. Once the requesting device receives the negation of BBSY and SSYN, it asserts BBSY becoming bus master.
5. The bus master places the interrupt vector (0234) on the D(data) lines, asserts INTR, then negates SACK.
6. The bus arbitrator and interrupt fielding processor receive the asserted INTR. The interrupt fielding

processor strobes the interrupt vector from the data lines, then negates SSYN.

7. The bus master receives the asserted SSYN, removes the interrupt vector from the D lines, and proceeds to negate INTR and BBSY.
8. The arbitrator and interrupt fielding processor receive the negation of INTR. The interrupt fielding processor then negates SSYN.
9. After receiving SACK negation (step 5), the arbitrator waits 75ns, then resumes issuing NPG's (non-processor grant) but not BG5's. This is followed by the interrupt fielding processor informing the arbitrator that it may start issuing BG5's.

The important thing to note here is that 6 control lines must be present: BR5, BG5, SACK, BBSY, SSYN, and INTR.

2.2 PDP-11/60 Alterations

The software device driver for the trackball was written to communicate with the trackball device from programs running on the PDP-11/60 host computer. The DeAnza image display contains the trackball controller board. Through the process of debugging the trackball device driver, it was determined by the use of a logic analyzer that some of the control lines necessary to perform an interrupt transaction were not physically connected to the controller board. The results of the use of the logic

analyzer were that BR5, BG5, SACK, and INTR were not present in the DeAnza image display.

The DeAnza UNIBUS cable is inserted in the section of the PDP-11/60 backplane where the memory for the PDP-11/60 host computer resides. The memory does not request the bus, because data is written in and read out by the CPU; hence, interrupt signals are not routed to the memory portion of the backplane.

The process of getting the signals to the DeAnza required two steps. The first step involved placing wirewrap jumpers from the UNIBUS to the DeAnza and memory backplane. BR5, BG5, SACK, and INTR were wirewrapped from a portion of the backplane where these signals exist to the DeAnza/memory portion of the backplane (Figure 2.2). This resulted in the signals reaching the DeAnza; however, the trackball controller was still unable to issue a bus request and receive a bus grant properly. Further investigation with the logic analyzer revealed that the UNIBUS terminator card inside the DeAnza image display card cage required jumpers for proper termination of the BG5 line (Figure 2.3). This jumper was inserted and resulted in the proper operation of trackball interrupts.

2.3 DeAnza Modifications

The DeAnza ID5400 monochrome image display system has a resolution of 512 rows by 512 columns. Individual picture elements (pixels) are coded with 8 bits of gray scale

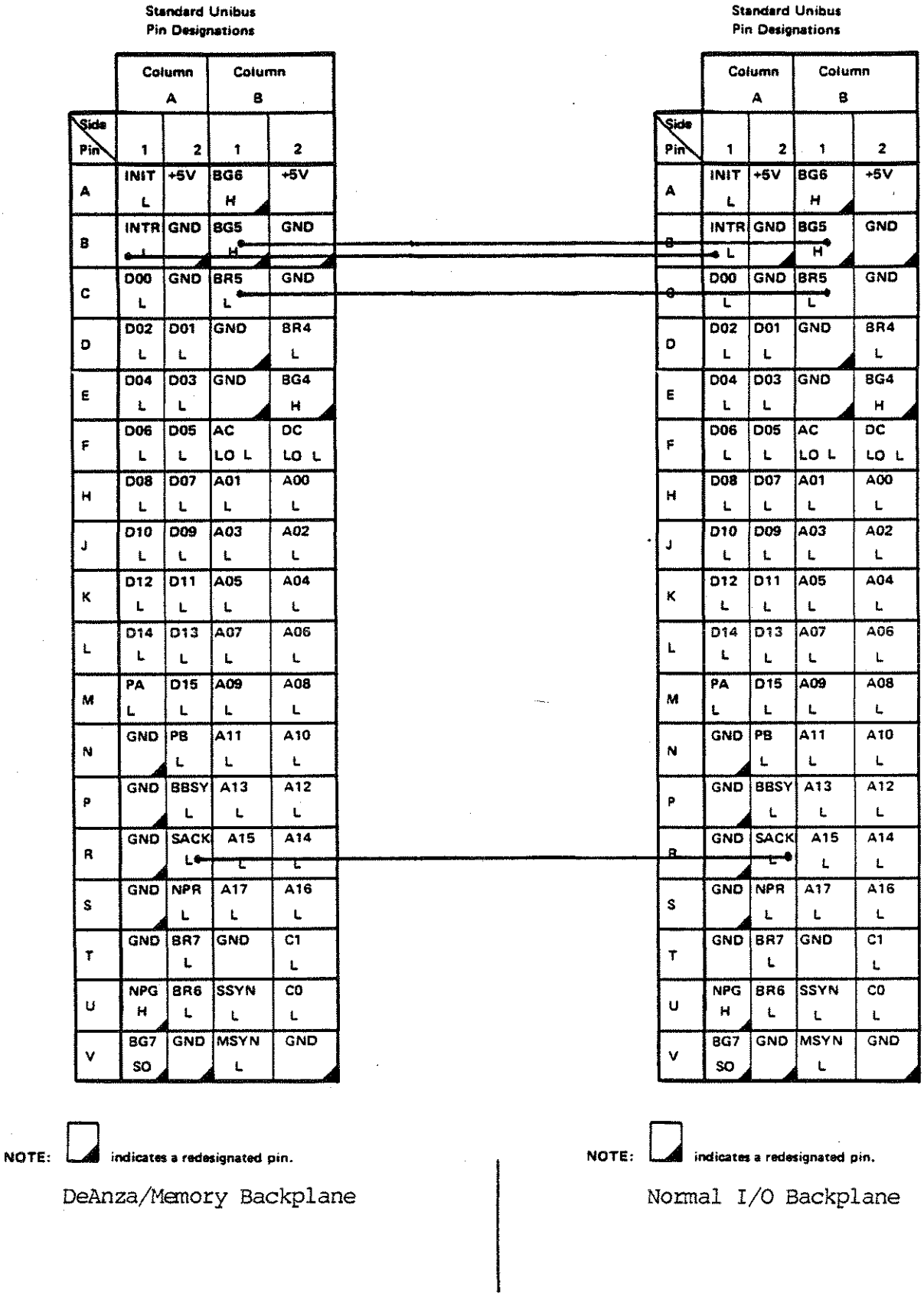


FIGURE 2.2 PDP-11/60 Backplane Connections

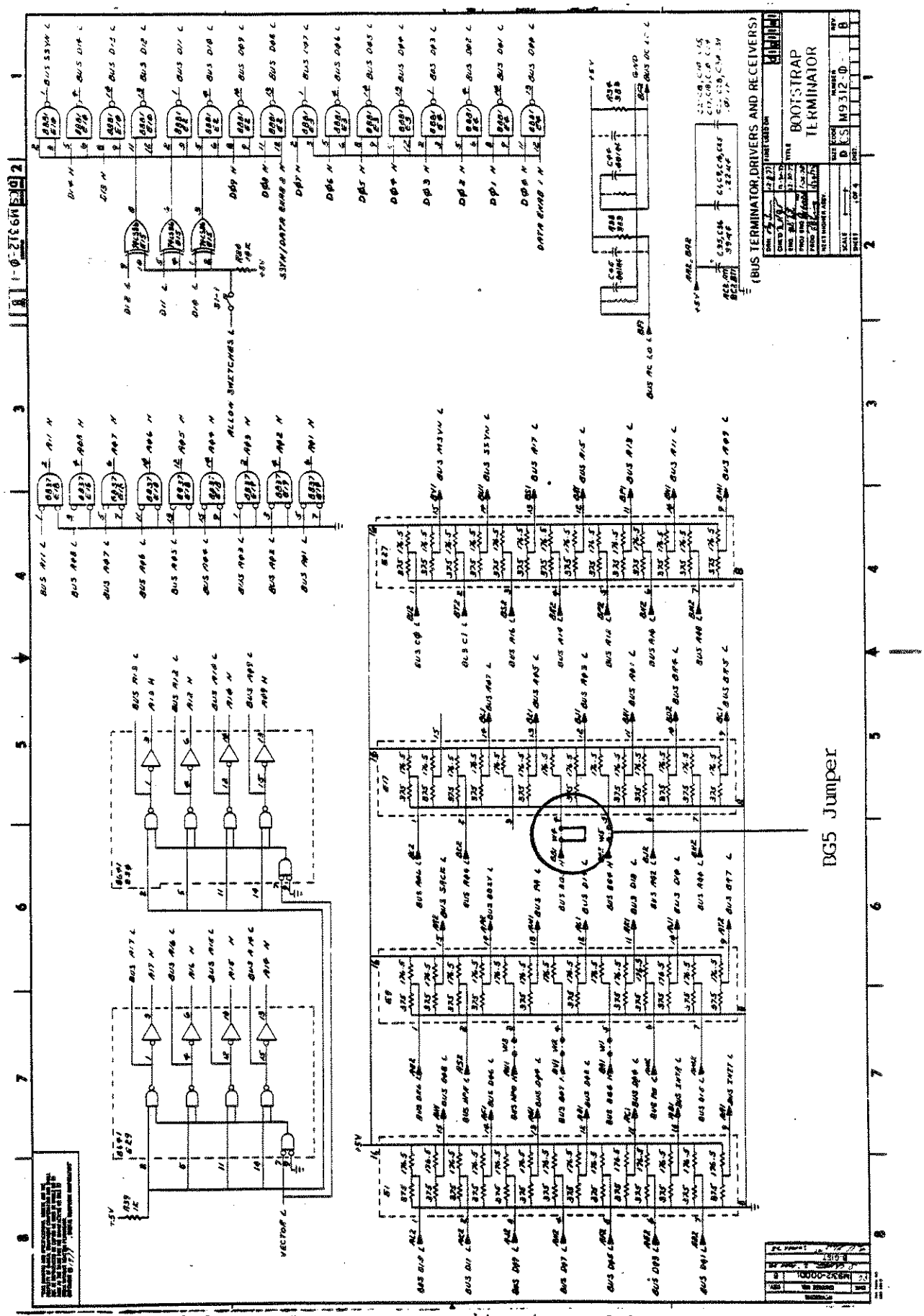


FIGURE 2.3 UNIBUS Bootstrap Termination Module

information, resulting in a level of gray between 0 (black) and 255 (white). The ID5400 is capable of working with three full 8 bit frames of image memory plus an overlay plane with 4 bits of gray scale information. This image display system is also capable of supporting a pipeline array processor, a trackball, a joystick, and a lightpen.

The image and overlay memory planes are constructed of high-speed, dual-port, random-access memory (RAM). The dual-port nature of the memory allows the PDP-11/60 host computer to write to and read from one port of the RAM, while the other port is scanned in a raster pattern and the output is put through the intensity transformation table, then through a digital-to-analog converter to provide the video signal output to the display monitor.

The current configuration contains 2 full frames of image memory, a 4 bit overlay plane, and a trackball with its controller. The second frame of memory was installed in the DeAnza image display, and the trackball was rendered operational after this system was first described by M. Riley [16].

Chapter 3: The Image Processing Software System Hierarchy

3.1 Introduction

The DeAnza ID5400 monochrome image display system, PDP-11/60 host computer, and the associated image processing software are integral parts of the LSPMM. In order to provide an organized design structure, top-down design, in combination with structured programming, was used to implement the image processing software. These programs were written in the C programming language, running under the UNIX version 7 operating system [13]. The image software is organized in the form of a hierarchy, with the highest level interacting with the LSPM microscopist, and the lowest level performing hardware control. The goals are to provide the novice microscopist with a friendly environment in which to operate the LSPMM, and to provide the future programmer with a structured environment in which to develop programs.

The image software hierarchy consists of four distinct levels. The top level, Level 1, contains all the commands that the microscopist will need to know to operate the microscope and process microscope images. Level 2 contains the programs that actually implement the Level 1 commands. Level 3 consists of functions and subroutines used by Level 2 programs. These routines are available for the experienced user to create custom programs, if the selection available in Levels 1 and 2 does not suit the user's

particular need. The lowest level, Level 4, contains the routines that actually perform hardware control. This permits the software programmer to control the display system without actually being concerned with the register bits necessary to perform a particular function.

This system hierarchy was utilized as a framework in which to assemble programs considered basic to forming a user friendly image processing software package. The following sections describe the different levels in more detail. It is important to note here that all the capabilities described below have not been developed. These sections simply outline the capabilities that may be available now or in the future at each level. The next chapter will discuss what was actually implemented within the framework described below.

3.2 Hierarchy Level 1

Level 1 contains a series of "shell" programs that allow the user to control display functions and file management, along with other "shell" programs that perform a broad spectrum of image processing operations [4]. Alternatively, an experienced user can create customized command files on his own. A menu or listing of the Level 1 programs available to the microscope user can be displayed on the DeAnza display and/or the user terminal simply by typing the "help" command.

The following is a list of Level 1 programs/commands that are available for the biologist's use, along with a short description of the types of operations they control. Seven of the commands have been implemented to date, the other six commands may be implemented in the future. In order to minimize typing by the user, each of these commands has a preset default mode of operation; thus the beginning user does not need to comprehend all the capabilities of the system in order to make use of it. Wherever functionally feasible, the selection of command options is possible via command line arguments. If further information is necessary but not supplied, the terminal prompts the user for the appropriate information.

1. <scan> -- This command allows user control of the interface between the microscope and the DeAnza image display system. Using this command, the biologist can control the different parameters of image scanning, including input format, memory frame destination, and number of images. For the beginning user, the only input required is the number of images to scan. The program defaults to a particular input format and memory frame destination.
2. <display> -- This command allows the user to display images on the DeAnza display that have been saved in a computer file on the system disks. Each image stored

on the system disk has an associated image file name. Optionally, the microscopist can create and display a caption on the image display. The user can also control whether or not the intensity reference scale is displayed.

3. <save> -- This command controls the various operations involved in actually saving an image as a computer file on the system disks. These operations include naming the image file, the creation of the proper image file header, and writing the image from the DeAnza display to the disk file. Options may exist in the future to allow the biologist to place a permanent caption in the file header along with a box or other cursor form to denote a particular area of interest on the image. This command may also provide the option of storing an image on magnetic tape.

The preceding commands are basic to the operation of the LSPM microscope. The rest of the commands available in Level 1 will have a much broader scope.

4. <remove> -- Image files created through the use of the "save" command may be deleted using this command. Histograms and other overlays displayed on the monitor may be removed using this command.
5. <zoom> -- Two types of zoom may be available to the microscopist. Fast zoom utilizes the hardware capabilities of the DeAnza display system with zoom

factors of 1X, 2X, 4X, and 8X. This fast zoom capability is combined with the scrolling function and trackball of the DeAnza display to provide the biologist with a useful image reviewing mechanism. Slow zoom will provide the biologist with non-standard zoom factors. Through the use of trackball and cursors, the user will be able to specify the area to be zoomed. A new zoomed image will be generated via interpolation techniques. This method will be significantly slower than the hardware method. The trackball cursor shape may optionally be controlled from this command.

6. <contrast> -- This command will allow the microscopist to perform point operations on an image. The point operations will involve different types of gray scale modification. Applications of point operations include histogram flattening, histogram matching, photometric decalibration, and display decalibration [5].
7. <smooth> -- This command will allow the microscopist to employ discrete convolution techniques to smooth a noisy image. A choice of several different smoothing algorithms will be available.
8. <filter> -- This command should be highly interactive when implemented. It should describe the filtering schemes available and allow the microscopist to select

one of the schemes present in the system. These schemes will include both spatial domain and frequency domain filtering. The types of operations performed in this program may overlap operations performed by the "smooth" and "restore" commands. The "filter" command will include general low-pass, high-pass, band-pass, and notch filtering programs, along with feature enhancement programs, and programs for the removal of periodic noise.

9. <restore> -- Through this command, the microscopist can select from among various programs that remove degradations in the image. Sources of this degradation include the optical system imperfections, image motion, and noise. The available restoration methods will include Wiener deconvolution, power spectrum equalization, and geometric mean filters [5].
10. <datanal> -- Operations available via this command allow the microscopist to obtain measurements and statistical information pertaining to an image or part of an image. Routines are available to output a specific pixel value, and to output an image histogram. Programs may exist in the future to calculate the area and perimeter of regions specified with the trackball cursors, along with mean pixel values for an outlined area.

11. <arithops> -- This command will allow the microscopist to perform simple arithmetic operations on all or part of a specified number of images. Simple arithmetic operations available under this command will include addition, subtraction, multiplication, division, and logarithms. The trackball and cursors may be used here to specify desired areas of operation.
12. <graphics> -- Through the use of this command, the microscope user may display several types of overlay graphics. Operations available under this command may overlap some of the options present in other Level 1 commands. There will be programs available to display histograms, to outline desired areas, and to display arrows at desired points in the image.
13. <help> -- This command provides assistance for the Level 1 users. Executing "help" with no arguments will display a general listing of the commands available and their capabilities. Detailed information on each command will be displayed by typing "help" followed by a Level 1 command name.

In the future, a typical user session for a beginner might proceed in the following manner. First the microscopist would log in to the computer, and then type 'help.' At this point, a menu with the selection of the Level 1 commands available would appear on the DeAnza

display. The user might select "contrast" by either typing the command into the terminal or by positioning the trackball cursor at the command on the display and pushing a switch on the trackball controller. A new menu would appear on the DeAnza display that lists the options available under the "contrast" command. An option might be selected in the same manner as the original Level 1 command, and the terminal would prompt the user for any necessary additional information, then the desired task would be executed. In the case of an experienced LSPM microscope user, this entire operation could be specified at the terminal by typing on a single line:

contrast [option] [information] <carriage return>

3.3 Hierarchy Level 2

Level 2 consists of task-oriented, interactive programs that are called by Level 1 programs. The beginning user need not be concerned with the programs available at Level 2; however, the experienced user may optionally wish to invoke a Level 2 program to conserve time and typing. Each Level 2 program may be considered an offspring from one or more parent Level 1 commands. The tight coupling of the 4 level hierarchy permits easy system expansion. As new Level 2 task routines are written, they can be incorporated into the appropriate Level 1 command with a minimum of difficulty.

A number of Level 2 programs are available for use, including scan, disp, save, etc. Further information on these and other Level 2 programs presently in use, along with the necessary Level 3 and Level 4 functions, may be found in Appendix 2. The Level 2 commands necessary to implement the first three Level 1 commands "scan," "display," and "save" have been developed and tested.

3.4 Hierarchy Level 3

Level 3 is presently organized as a single library of image functions. There may be more libraries at this level in the future, containing commonly used functions, such as matrix operations, file manipulation, and transform generators. The functions in this level will be the routines responsible for the majority of the actual work done in the processing of an image. These additional libraries may include, for example, addition, subtraction, multiplication, and division routines with selectable input, that is all or a specific part of an image. This level will also contain the functions responsible for interprocess communication between the host computer and the DeAnza display system. The functions at this level are hidden from the microscopist and used only by the programmer. These image libraries will be useful for program development, providing a source of high level software tools. Examples of the type of functions included in this level may be found

in Appendix 2, e.g. i_open(), i_close(), i_da_to_dsk(), and i_dsk_to_da().

3.5 Hierarchy Level 4

The fourth and lowest level, Level 4, contains the routines that provide the interface between the software program in the host computer and the hardware capabilities of the DeAnza display system. Hardware capabilities presently include:

1. The ability to store and display two 512 by 512 pixel images.
2. An alphanumeric character overlay generator to display annotations.
3. An overlay memory plane to display graphics.
4. A cursor generator and trackball controller to provide a simple user friendly system interface.
5. Scroll and zoom with a zoom factor choice of 1, 2, 4, or 8 to allow a fast method of zooming an image.
6. An intensity transformation unit exists to allow the image data to be routed through a look-up table en route to the digital-to-analog converter and video generator where pixel values may be altered.

This combination of features provides for the display of menus and histograms, along with providing the microscopist with a simple image reviewing mechanism. This level contains the driver software for the DR-11W DMA controller

and the trackball. Several sections of the operating system generation code were modified; this code also resides in Level 4.

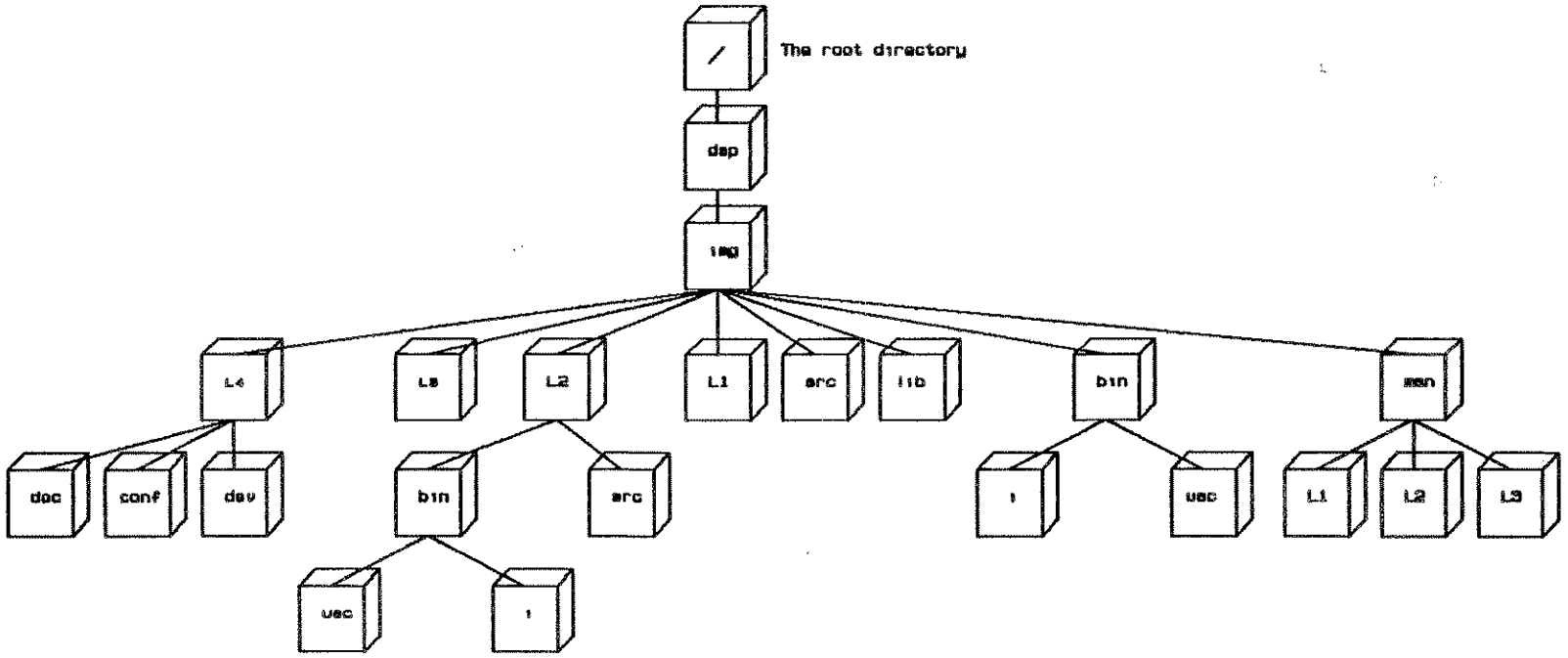
Chapter 4: System Implementation

4.1 Background Information

The UNIX operating system provides a number of convenient features useful in the creation of the four level system hierarchy. The ability to create a tree structured system of directories and subdirectories served as the method for organizing the framework to contain the four distinct levels of program code [17]. Level 1 programs were implemented through the use of the shell command language [4]. Level 2 programs were written in the C programming language utilizing many of the functional tools implemented in Level 3 [13]. Level 4 program code serves as the operating system level interface for the DeAnza image display. The following sections describe in more detail the tree structured directories, the shell command language, the object code library, and the system-generation "make" files used in the implementation [10].

4.2 Tree Structured Directories

The UNIX operating system's tree structured system of directories and subdirectories provided the means for implementing the four distinct levels of programming code. The UNIX file system is organized in the form of an upside down tree (Fig 4.1), with the 'root' directory of the filesystem at the 'top' of the tree. Several subdirectories exist as branches of the root directory. If one's current



This diagram shows the subdirectories under the directory /dsp/img

FIGURE 4.1 UNIX Filesystem Organization

directory is the root, then these branches are referred to as subdirectories of the root. However, one's current directory may also be a subdirectory of the root directory. This subdirectory may also contain further subdirectories, the limit being a function of disk storage and machine dependent operating system parameters.

Referring to Figure 4.1, the image processing software resides in the subdirectory of the root called "/dsp/img." A number of subdirectories exist within the subdirectory "/dsp/img," which may be categorized as follows: The microscopist operates within the subdirectories "L1," and "L2," and the programmer operates within the subdirectories "L1", "L2," "L3," and "L4." "L1," "L2," "L3," and "L4" contain the program code for Level 1, Level 2, Level 3, and Level 4 respectively. Programs may be developed or modified in other subdirectories of "/dsp/img" to provide a margin of safety; thus working programs are not replaced with non-functional experimental versions. The "src" subdirectory contains the source code for Level 2 programs which are available in executable form in the subdirectory "bin." Level 3 function subprograms are available in the subdirectory "lib," along with the appropriate archiving shell command. The subdirectory "man" contains subdirectories L1, L2, and L3, containing the text files for the manual pages, and the text files for the help command. Level 4 program code may be found in the appropriate operating system

directories "/usr/sys/dev" or "/usr/sys/conf." The subdirectories "src,", "bin," and "lib," provide a margin of safety for both the programmer and the microscopist; allowing the programmer to develop and run programs separate from where the microscopist is working. The result is that programs being tested cannot alter those needed by the microscopist until these programs are completely debugged and ready to install in the four level system hierarchy.

4.3 The Shell

Level 1 programs were implemented through the use of the UNIX shell [4]. The shell refers to both the command line interpreter for the UNIX operating system, and the programming language used to direct the operations of the command interpreter. This command interpreter serves as the user interface to the UNIX operating system. The shell programming language provides a number of high level algorithmic language constructs such as numerical variables, string variables, control-flow structures, and argument passing. "While-done," "if-then-else," and "case" are examples of the control-flow mechanisms available in the shell programming language.

The shell programming language permitted a number of Level 2 programs to be assembled together under a single Level 1 command. A good example of this is shown below in the Level 1 command "display."

```
if test "$1" = "catalog"
```

```
then
    cd ../L2/bin;ls -s $2
    exit 1
elif test "$1" = "channel"
then
    cd ../L2/bin;show -$2
    exit 2
elif test "$1" = "caption"
then
    cd ../L2/bin;an 25 c=9 $2 $3 $4 $5 $6 $7 $8 $9
    exit 3
elif test "$1" = "scale"
then
    cd ../L2/bin;inrefs $2 $3 $4
    exit 4
else
    cd ../L2/bin;disp $1 $2
    exit 5
fi
```

This command controls a number of display capabilities available within the image processing software. Five separate Level 2 programs are integrated into "display." These Level 2 programs individually provide for the display of saved images, the display of captions on the image, the display of an intensity reference scale, the ability to list image directories, and the ability to change memory frames displayed. It is a simple matter to add further capabilities to this command through the use of the shell programming language. The programmer starts with the development of a Level 2 program that performs or controls certain operations. This Level 2 program is then incorporated into the appropriate Level 1 program through the addition of a small amount of code to the existing Level 1 program.

Level 1 programs may be created in a simple straight-forward manner. A file, referred to as a shell file, is created containing the desired shell programming language operations. Since the shell is a form of command interpreter, there is no need to compile the shell program. There are two methods available to run the shell program. The first method requires the person executing the shell program to precede the shell program filename with "sh" as indicated below:

```
sh <shell filename>
```

The second method requires the programmer creating the program to change the permissions on the shell file to make the file executable. This can be accomplished via the UNIX command (qv):

```
chmod 755 <shell filename>
```

The Level 1 program may now be executed by simply typing the shell filename followed by a carriage return.

4.4 Object Code Library

Level 3 of the system hierarchy contains a group of function subprograms that are used extensively in most Level 2 programs. These function subprograms are called from the main program in Level 2; however, it is not necessary to include the source of the function subprogram with the source of the main program. These Level 3 function

subprograms are available in an object code library called "/usr/lib/libimg.a." The programmer need only be informed of the capability of a particular function subprogram to make use of the function subprogram within a Level 2 program or another Level 3 function subprogram. Level 3 function subprograms can be used much like standard C language instructions, once they are incorporated into the object code library.

In order to appreciate the usefulness of this object code library, one must be aware of the procedure followed to create an executable Level 2 program. The first step is the creation of a file containing the Level 2 program source code. The program below is a simple example of Level 2 source code.

```
FUNCTION:  This program serves as a software toggle switch
for the channel annotation.  If the channel annotation is
not displayed, this program will display it.  If the channel
annotation is displayed, this program will remove it from
the display.
#include        <Regdef.h>
main()
{
/* local declarations */
int     page;
int     *reg;
int     chan;
int     memcont;
int     csr_flag;
/* map to the DeAnza display */
page = first_page();
reg = virtual(page);
setreg (page,IDREGPG);
/* read the channel currently displayed */
chan = reg[IXSPLT] & 03;
/* read the contents of the annotation memory and the
annotation enable bit in the control status register */
memcont = (reg[ANOTTN + 35] >> 8) & 0377;
```

```
csr_flag = reg[CSR] & ANOTENBL;
/* perform the switching function */
if ((memcont != 0) && (csr_flag != 0))  reg[ANOTTN + 35] &= 0377;
else if ((memcont != 0) && (csr_flag == 0)) reg[CSR] |= ANOTENBL;
else  a_printf(1, 72, 'w', "%d", chan);
}
```

The source code can contain standard C programming language instructions, Level 3 function subprograms, and functions available in any of the standard C libraries of object code. This source code is compiled, resulting in the creation of an object code file. The next step in the process involves linking this object code with the object code needed from the libraries. This is accomplished via the addition of an option flag such as "-limg" in the compile string. This option flag is passed to the loader at load time. The loader performs the link operation and creates an executable file. This program may now be run by simply typing the executable filename followed by a carriage return.

An object code library is created by using the UNIX command "ar" [11]. This command is an archive and library maintainer. The process of creating the image object code library can be broken down into three parts. The first part requires the programmer to be in the L3 subdirectory. This directory contains the source code for the image function subprograms. Part two requires the programmer to compile the function subprograms down to object code. This can be accomplished via the execution of a shell file as follows:

```
compall <carriage return>
```

The final part assumes the compile stage was performed with no errors returned. The programmer now proceeds to create the image library with the execution of a shell file as follows:

```
mklib <carriage return>
```

This shell file contains the "ar" command string in the proper form to create the library "/usr/lib/libimg.a." This process is standard UNIX protocol for creating an object code library [3]. To add a function subprogram to the image library is relatively simple. The programmer moves the source code for the newly created function subprogram into the subdirectory "L3." Next, the "mklib" shell file is modified to include the new object code filename. Finally, the "compall" and "mklib" sequence is executed as described above. This will result in the creation of a one pass object code library "/usr/lib/libimg.a" that includes the new function subprogram. It is possible to simply add the Level 3 function subprogram to the object code library with the "ar" command; however, the library may no longer function as a one pass object code library. In this case, the -limg flag must be specified more than once in the compile string.

4.5 Operating System Interface

The operating system interface code resides in two file system sub-trees: "/dsp/img/L4" and "/usr/sys." Two types

of programming code are grouped within Level 4. The first group contains the device driver code "dr.c" and "tb.c" for the DR-11W DMA controller and the trackball respectively. This group also contains a version of DeAnza device driver called da.c that was purchased from PAR Technology Inc. [14]. The second group contains several system files that were modified. This includes "/usr/sys/sys/machdep.c" which was originally modified by M. Riley [16]. Another file that required extensive modifications was "/usr/sys/conf/mkconf.c." The most useful file in this group, however, is a "make" file called "/usr/sys/conf/makefile."

The device drivers "dr.c" and "tb.c" contain the actual routines that are used by the operating system. When a running program requests any type of interaction with the DR-11W DMA controller or the trackball, it must be through the functions contained within the device driver, otherwise the operating system will not recognize the device. A generic character device driver will contain functions that allow the programmer to open and close the device in the same manner as a standard file. There will also be functions that enable the programmer to write information to a device and read information from the device. It is also possible that there will be a function called "ioctl()" (input-output control). This function allows the programmer to control functions within a specific device. The

peripheral device appears as a special file created in the subdirectory "/dev."

The file "mkconf.c" provides the physical link between the special file created in "/dev" and the operating system [18]. The special file created allocates two special numbers, known as the major and minor device numbers, to a unique peripheral device. The result of executing the program "mkconf" is the creation of two files called "c.c" and "l.s." The file "c.c" has the major device numbers incorporated into two arrays referred to as the character and block device tables. The assembly language file "l.s" contains the code that provides the jump instruction to the appropriate interrupt handler on receiving a unique interrupt vector. These two files play an integral part in the creation of a new operating system kernel. The system kernel is regenerated in order to accommodate the addition of new peripheral devices.

The creation of an operating system kernel is a complicated process that is greatly simplified by the "make" program supported under the UNIX operating system [10]. The "make" program provides an organized framework for maintaining computer programs. The operating system kernel may be modified to include different peripheral devices with a few simple steps. The UNIX "ar" (archive) command is used to place the device driver for the peripheral in an object code library "/usr/sys/dev/LIB2.40" along with the object

code for the other known device drivers. The programmer must then modify the character and/or block device table arrays within "mkconf.c," and recompile this using the following command:

```
make mkconf <carriage return>
```

The next step requires the programmer to create or modify a configuration file in the subdirectory "/usr/sys/conf/conf.tbl." The "makefile" within the subdirectory "/usr/sys/conf" is modified such that the particular system kernel one wants to generate has the desired configuration table input to the "mkconf" command. The system kernel "unixlspm" may be generated with the following command:

```
make unixlspm <carriage return>
```

To illustrate how one can modify the operating system using the program "make," consider the following example. Suppose you wish to generate an operating system kernel that contains the trackball device driver, but does not contain the DR-11W device driver. The simplest method to perform this function involves two steps. The first step requires the programmer to delete the following line from "/usr/sys/conf/conf.tbl/lspmconf":

```
dr
```

The programmer executes the following command as a second step:

make unixlspm <carriage return>

This will result in the creation of a new system kernel "unixlspm" that contains the trackball device, but not the DR-11W DMA controller device.

4.6 Current Image Processing Software Overview

Table 4.1 summarizes the image processing software that is currently available in each of the four system levels. To give credit where it is due, indicators have been placed beside programs where contributions were made by more than one person. Some programs required only slight modification prior to incorporating them into the four level system hierarchy, others required extensive revision. For further information regarding the modifications, the reader is referred to the program listings in Appendix 2. Each program and function subprogram source listing contains a program header section labeled "MODIFICATION HISTORY."

The following chapters contain detailed descriptions of the capabilities of each of the programs listed here. These chapters will not contain a description of how they work. The program header that precedes each program and function subprogram includes a detailed program description. The source code for each program and function subprogram is heavily commented. The reader is referred to the program

listings in Appendix 2 for detailed information concerning how the programs work.

Current Image Processing Software Overview

LEVEL 1	scan save display zoom datanal remove help		contrast * restore * graphics * arithops * filter * smooth *
LEVEL 2	adisable.c $ adisp.c $ ainput.c $ an.c chan.c	disp.c $ hist.c & inrefs.c intl.c makefile	save.c scan.c $ show.c tbzoom.c unhist.c &
LEVEL 3	a_clran.c a_disable.c $ a_display.c $ a_doprnt.s a_printf.c compall h_display.c & h_print.c &	i_close.c $ i_creat.c $ i_curs.c i_da_to_dsk.c $ i_dsk_to_da.c $ i_flag.c $ i_getpix.c i_map.c	i_open.c $ i_refs.c i_scan.c $ i_scroll.c i_show.c i_zoom.c mklib tb_adb.c
LEVEL 4	dr.c $ tb.c da.c		makefile mkconf.c

* indicates the Level 1 command has not been implemented.

$ indicates that the original version of the program was written by M. Riley.

& indicates that the original version of the program was written by K. West.

TABLE 4.1

Chapter 5: Level 1 Program Descriptions

5.1 Overview

Seven Level 1 commands were implemented for use with the LSPMM. Scan, display, and save were the first Level 1 commands to be created, since these commands are critical to the operation of the microscope. Versions of the Level 1 commands "remove," "datanal," "zoom," and "help" were also implemented. The strategy followed in development of these Level 1 commands began with the development of Level 2 program modules. These Level 2 program modules were written as main programs with function subprograms utilized wherever possible. Once the Level 2 program was completely debugged, the function subprograms were removed from the Level 2 program source file and placed in an object code library "/usr/lib/libimg.a." This proved to be an efficient method for developing useful software tools. Function subprograms developed in previous Level 2 programs proved to be very useful for further program development. The most efficient way to write these programs is to build on the material already present, making use of the software tools. If the function subprograms cannot be used in their present form, then a programmer may modify them to suit the purpose. This method proved to be much faster than trying to write the programs from scratch. When a Level 2 program was complete, it was incorporated into the Level 1 command for which it was developed.

Each of the Level 1 programs implemented here are fairly simple in operation. The Level 1 commands "scan," "save," and "zoom" change the directory to the subdirectory where the Level 2 commands reside, and execute the appropriate Level 2 command. The remaining Level 1 programs make use of the program "test" in order to perform string comparing operations. By combining the if-then flow control mechanism with the "test" program, Level 1 programs may selectively execute Level 2 program modules given a particular key word as an argument to the Level 1 command. The Level 1 program "datanal" has been reproduced below as an example. In this shell program, two key word sequences are accepted "gray value" and "histogram." The variables "$1" and "$2" contain the first and second argument strings supplied when the shell program is executed.

```
if test "$1" = "gray" -a "$2" = "value"
then
    cd ../L2/bin;tbzoom -czg
    exit 1
elif test "$1" = "histogram"
then
    cd ../L2/bin;hist $2 $3
    exit 2
fi
```

When Level 1 commands are executed, the user is prompted in an interactive format for information necessary to run the program. Some programs will simply execute with a particular default format. In the few cases where the commands require command line arguments for execution, the

argument string is passed through the Level 1 command to the appropriate Level 2 program module.

The Level 1 programs are somewhat restricted in the operations they perform. Each command will execute in a particular default format; however, there may be options available within the Level 2 program module that is not normally used within Level 1. This situation may be remedied in the future through the development of a general program to serve as a menu interface to the user. Through this program, the Level 1 user could be given a complete description of all the options available in a given Level 1 command. The user would then select an option by typing a command string or by placing a cursor over the command string with the trackball and pushing a button.

The following sections describe the function and capabilities of the seven commands implemented in Level 1.

5.2 Level 1 Command - scan

This command executes a Level 2 program module called scan. The ground work for this program was done by M. Riley [16]. The main program at Level 2 was completely rewritten for reasons which will be outlined in the next chapter. This program will require modification in two cases:

1. A third DeAnza memory plane is added to the system.
2. The hardware capability to scan an image of a size other than 512 by 512 pixels is added to the system.

The microscopist makes use of this program once the specimen, microscope, and associated hardware are set to generate an image. The easiest way to use the program is to type the following:

scan 1 <carriage return>

The second argument "1" refers to the number of images to scan. It is important to note here that while scan does not store more than one image at a time, it is possible to scan more than 1 image at a time. The program currently allows the user to scan to either of the two DeAnza memory channels, defaulting to channel 0. The user is currently restricted to scanning 512 pixel by 512 pixel images.

5.3 Level 1 Command - save

This command executes a Level 2 program called "save." The microscopist uses this command to store a scanned image in a disk file. The program "save" is interactive, prompting the user for the required information. The user may store an image in row or column format from either memory channel 0 or 1, defaulting to a row format saved from channel 0. The simplest way to use this program is to type the following command:

save i/image1 <carriage return>

The characters "i/" in the command line argument refer to a special subdirectory created to serve as an image catalog.

The argument string "image1" is the image file name supplied by the user. This command would save the image in memory channel 0 in row format under the filename "image1," in the image catalog "i."

5.4 Level 1 Command - display

Four separate Level 2 program modules are incorporated into this command. The Level 2 program "disp" does most of the work required in this command. The microscopist makes use of "display" in order to display images stored in a disk file on the video monitor. The command provides the option to display the image in row or column format, and allows the user to select memory channel 0 or 1. Assuming an image named "image1" has been stored in the image subdirectory "i" using the "save" command as outlined in the previous section, the image may be displayed using the following command:

```
display i/image1 <carriage return>
```

The above command string would cause the image "image1" stored in a disk file in the image catalog "i" to be displayed in row format on memory channel 0.

Suppose the user wanted to list the contents of the image catalog "i." An option exists within this Level 1 command to accomplish this by typing:

```
display catalog i <carriage return>
```

The "display" command allows the user to select the memory channel to be displayed through the use of a Level 2 program "show." If the user was currently displaying channel 0, and wished to change the channel displayed to 1, then the user would type:

display channel 1 <carriage return>

The DeAnza ID5400 contains a 2000 character annotation memory which provides the mechanism for displaying alphanumerics on the video monitor. The "display" command provides for the display of a caption on an image by typing the following command:

display caption hello, world <carriage return>

Execution of the above command would result in the display of the caption "hello, world" in the bottom left corner of the image.

Each pixel of an image contains 8 bits of information, resulting in a gray value between 0 and 255. It is possible to display an intensity reference scale on the bottom of the image with the "display" command by typing:

display scale on

The intensity reference scale displayed is a sequential scale of gray values from 0 to 255 in steps of four; thus 64 levels are displayed. The intensity reference scale display

may be turned off by substituting "off" for "on" in the above command string.

5.5 Level 1 Command - remove

This command uses two Level 2 programs "unhist" and "an." The microscopist uses this command to delete an image stored in a disk file. Suppose the microscopist wanted to delete the file "image1" from the catalog "i." This is accomplished by typing:

remove image i/image1 <carriage return>

This command invokes the UNIX command "rm" to delete "image1" from the image catalog "i." A future release may provide a "softer" deletion command, i.e., one that merely marks the image file for later deletion when the user logs out. If the user changes his mind, he can "un-remove" the file before leaving the system.

The "remove" command serves two other purposes. If the microscopist wants to remove a caption from the image on the monitor, the following command string should be typed:

remove caption <carriage return>

To remove a histogram that has been displayed over the image, the microscopist types:

remove histogram <carriage return>

These two commands do not have any options after the keyword. The keyword "caption" causes the shell to execute the Level 2 program "an." The keyword "histogram" directs the shell to execute the Level 2 program "unhist."

5.6 Level 1 Command - datanal

The "datanal" command incorporates two Level 2 programs "hist" and "tbzoom." Suppose the microscopist has scanned an image and wants to determine the gray value at a particular point in the image. The microscopist types:

datanal gray value <carriage return>

This command enables the microscopist to roll a cursor to this point on the image using a trackball and obtain the gray value at any point on the image. The keywords "gray value" direct the shell to execute the Level 2 program "tbzoom" with a special set of options. The gray value for a pixel at a given coordinate is displayed on the monitor next to an annotation that reads "gray value =."

The histogram is a useful tool in determining the quality of an image. It is a bar graph of the number of pixels of given gray value for each gray value from 0 to 255. The microscopist can generate a histogram for an image by issuing the command:

datanal histogram <carriage return>

The above command will generate a histogram of the image displayed in memory channel 0 and display the histogram overlayed on the image. The histogram display includes an annotation indicating the maximum number of pixels occurring at a gray value and that gray value.

5.7 Level 1 Command - zoom

The "zoom" command invokes the Level 2 program "tbzoom." This program provides the microscopist with flexible image reviewing mechanism. The command is issued by simply typing the following command string:

zoom <carriage return>

Through the use of the trackball, and the scroll and zoom capabilities of the DeAnza display system, this program allows the microscopist to zoom in on localized sections of an image. This zoom is a form of hardware zoom that is very fast, providing 1X, 2X, 4X, and 8X magnification factors. The switches on the trackball control several features. The left switch controls magnification, and the right controls the memory channel displayed. The magnifications are cycled through 1X, 2X, 4X, and 8X by repeatedly pressing the left button. The center switch functions as a store mechanism. The microscopist may store a particular position, magnification, and memory channel by pressing in this switch, then toggle between the present state and the stored state using this switch.

5.8 Level 1 command - help

This command provides on-line assistance to the microscopist. The "help" command does not use any Level 2 commands. It is invoked simply by typing:

help <carriage return>

or

help keyword <carriage return>

The command simply prints out particular document files for a given keyword supplied. If no keyword is supplied, then the command prints out a general help file with list of the commands implemented and what they do, along with a list of the valid keywords to type for more detailed information on a particular topic. Future development of this command could integrate the UNIX command "man" with a menu selection mechanism allowing the microscopist to select commands with the trackball cursor from a command list displayed on the screen.

Chapter 6: Level 2 Program Descriptions

6.1 Overview

Level 2 programs and their associated options are described in this chapter. Each section begins with a "usage" statement block containing the user command string along with the options available under this program module. The command string notation resembles the notation found in the Unix manual, Volume I [2]. Briefly, any string within square brackets "[ ]" is optional, and any string within angle brackets "< >" must be supplied by the user. Standard Unix protocol uses a "-" as a flag to indicate an option selection. Some programs require an image filename if a certain option flag is selected. The image filename is typed after the option flag, separated from the flag by a space. If one of the Level 2 programs is executed without supplying any options or command line arguments, there are three possible outcomes. The first and most probable outcome is that the program will interactively query the user for the information necessary to execute the desired function. The second possible outcome results in the Level 2 program executing in a default format without questioning the user for more information. The third outcome results in the output of a "usage" statement block similar to those found in the beginning of each section.

A typical Level 2 program begins with the C language statement "main(argc,argv)." This statement indicates the

start of the main program, into which is passed a command line argument count and a pointer to the argument strings. The "main" statement is followed by variable declaration statements. The variable declarations are followed by a section that performs the command line interpretation. This section will interpret the command line arguments supplied by the user, or will handle the interactive inquiry necessary to execute the program successfully. The command interpretation section is followed by a section which performs calculation, then calls one or more Level 3 function subprograms. The main program is usually completed on return from the last Level 3 program to be called. The sections below will indicate some of the important Level 3 function subprograms called in the Level 2 program. The Level 1 programs into which the Level 2 program was incorporated will also be specified.

Some of the Level 2 programs below are modified versions of programs that were developed by others. The reader is referred to the program listings in Appendix 2 for further information. Each previously existing program contains a section describing the modifications made to Level 2 program. The reader is also referred to M. Riley's thesis (Riley, 1982) and C. West's B.E. project report (West, 1982). The Level 2 programs below were either developed for this thesis, or required extensive modification, therefore they are documented here.

There were two major reasons for not using the software available without modification: minimal upward compatibility and incorrect host computer to DeAnza interface. The software that existed was based on a starter package of software tools purchased from PAR Technology Inc. This software was designed to run on a PDP-11/34 host computer interfaced to a DeAnza display system with a color video monitor, neither of which we had. The specifications require the software to be capable of interaction with all available monochrome memory channels; this was not possible with the existing software. There was a considerable amount of frustration with the documentation supplied by the DeAnza manufacturer until the program "intl" was incorporated into the software package. The existing software contained statements that had to be modified once it was realized that the DeAnza had to be initialized in a certain way before it would function as indicated in the documentation.

6.2 Level 2 Program - intl

```
Usage:  intl [-0123zh]
    -0  initialize channel 0
    -1  initialize channel 1
    -2  initialize channel 2
    -3  initialize the overlay memory plane
    -z  initialize the zoom and scroll registers only
    -h  prints the help message
    defaults to initializing all channels.
```

As indicated above, "intl" initializes the DeAnza display system. The initialization procedure clears numerous locations inside the DeAnza register page,

including the control status register and annotation memory. The scrolling registers for each memory channel are set for (0, 511) resulting in the top left corner of the image being the display point (0, 511). If the Level 2 program "intl" is not run on power up of the system, the scrolling registers may contain (0, 0). The DeAnza will not function as documented in this case.

This program is a modified version of an initialization program that was supplied by PAR Technology Inc. It is not incorporated into any Level 1 command; thus it remains transparent to the microscopist. The program is executed through the UNIX .profile mechanism each time the microscopist logs on to the computer.

6.3 Level 2 Program - scan

```
Usage:  scan [ [-012ynrch] <integer number of frames> ]
    scan alone is interactive
    -012 selects the channel scan destination
    -yn enables or disables mirror image option
    -rc selects row or column format
    -h prints out the help message
```

The original version of the "scan" program was written by M. Riley [16]. Version 1 was modified in order to incorporate several different features. The modified version uses the standard protocol for the user interface portion of the program with logical default modes set. The new version also supports the capability to scan to any of the available memory frames. A number of Level 3 function subprograms are called from this Level 2 program, including

i_refs(), i_show(), and i_scan(). The function subprogram i_scan() performs the scan operation given the function arguments supplied in the main program. The Level 1 program "scan" simply changes directory to "/dsp/img/L2/bin" and executes this Level 2 program. At present, all of the functional capabilities available through the Level 1 "scan" program are performed by this Level 2 program.

6.4 Level 2 Program - save

```
Usage:  save [ [-012dfrcsh] [nrow ncol] <filename> ]
    save alone is interactive
    -012 selects the display channel
    -df selects save from DeAnza or from a file
    -rc selects row or column format
    -s indicates the next 2 arguments contain
       the number of rows and columns in the image
    -h prints out the help message
```

The microscopist makes use of this program through the Level 1 program "save" in order to store an image in a disk file. The program accepts user input through command line arguments in the standard protocol format as described above. Default modes are set such that if the program is supplied with a filename only, then an image will be saved in row format from memory channel 0. The program interactively questions the user for information if it is not supplied in the command line.

This program provides a number of options that make it useful as a general image processing tool. Besides allowing format and memory channel selection, "save" is capable of storing an image from another disk file; thus should the

header on an image file be destroyed, the image file may be repaired using this command. The user may also specify a non-standard image size, such as 256 by 256 pixels.

Several Level 3 function subprograms are called from this Level 2 program, including i_creat(), i_open(), i_close, and i_da_to_dsk(). The function subprogram i_da_to_dsk() performs the save operation given the function arguments supplied in the main program. The Level 1 program "save" simply changes directory to "/dsp/img/L2/bin" and executes this Level 2 program. At present, all of the functional capabilities available through the Level 1 "save" program are performed by this Level 2 program

6.5 Level 2 Program - disp

```
Usage:  disp [ [-012rch] <filename> ]
    disp alone is interactive
    -012 selects the display channel
    -rc selects row or column format
    -h prints out the help message
```

The microscopist makes use of this program through the Level 1 program "display" in order to display an image stored in a disk file. The program accepts user input through command line arguments in the standard protocol format as described above. Default modes are set such that if the program is supplied with a filename only, then an image will be displayed in row format in memory channel 0. The program interactively questions the user for information if it is not supplied in the command line.

This program provides a number of options that make it useful as a general image processing tool. Besides allowing format and memory channel selection, "disp" is capable of displaying any image stored with the Level 2 "save" program; thus a 256 by 256 pixel image may be displayed provided it is saved correctly.

Several Level 3 function subprograms are called from this Level 2 program, including i_open(), i_close, i_refs(), i_show(), and i_dsk_to_da(). The function subprogram i_dsk_to_da() performs the display operation given the function arguments supplied in the main program. The Level 1 program "display" defaults to changing directory to "/dsp/img/L2/bin" and executes this Level 2 program.

6.6 Level 2 Program - an

```
Usage:  an <row> [<c=column>] [-lsdwbh] [<string>]
    -l  erase characters previously on line
    -s  erase all characters previously on screen
    -d  disable the annotation display, but leave
         the characters in memory.
    -w  white characters on a black background (default)
    -b  black characters on a white background
    -h  prints the help message
```

The microscopist makes use of this program through the Level 1 program "display" invoked with the "caption" option in order to display a caption or annotation at the bottom left corner of the image. The program accepts user input through command line arguments in the standard protocol format as described above. The default mode prints out the "usage" statement block when no command line arguments are

specified. This program is also executed by the Level 1 "remove" command to delete a caption from the display.

This program provides a number of capabilities that are not available through a Level 1 command. The program allows the user to specify the row and column where the caption will be displayed, along with black or white character selection.

Level 3 function subprograms are called from this Level 2 program, in order to map or connect main program variables to actual physical address locations in the PDP-11/60 host computer memory. The Level 2 program "an" was a precursor used in the development of a general function subprogram in Level 3 called "a_printf()."

6.7 Level 2 Program - inrefs

```
Usage:  inrefs [<off>, <on>, <64>, <128>, <256>, -0123h]
    defaults to channel 0 reference scale
    off - turns off intensity reference scale
    on  - turns on intensity reference scale
          defaults to 64 levels in the scale
    64  - indicates 64 levels desired scale
    128 - indicates 128 levels desired scale
    256 - indicates 256 levels desired scale
    -0  - specifies channel 0 reference scale
    -1  - specifies channel 1 reference scale
    -2  - specifies channel 2 reference scale
    -3  - specifies channel 3 reference scale
    -h  - prints out the help message
```

The microscopist makes use of this program through the Level 1 program "display" invoked with the "scale" option in order to display an intensity reference scale at the bottom of the image. The program accepts user input through

command line arguments in the standard protocol format as described above. The default mode turns on a 64 level gray scale at the bottom of the image in memory channel 0.

This program provides a number of optional capabilities that are available through a Level 1 command, provided that the correct options are known to the user. The program allows the user to specify the memory channel where the scale is displayed, along with the number of levels in the scale.

One Level 3 function subprogram is called from this Level 2 program, in order to control the intensity reference scale. This program operates by simply decoding what the user desires done, and calling the Level 3 function i_refs() with the correct function arguments.

6.8 Level 2 Program - show

```
Usage:  show [ -0123h ]
    -0  channel zero
    -1  channel one
    -2  channel two (not yet purchased)
    -3  overlay channel
    -h  prints out the help message
```

The microscopist makes use of this program through the Level 1 program "display" invoked with the "channel" option in order to display the desired memory channel on the video monitor. The program accepts user input through command line arguments in the standard protocol format as described above. The program interactively questions the user for information if it is not supplied in the command line.

One Level 3 function subprogram is called from this Level 2 program, in order to control the memory channel displayed. This program operates simply by determining what the user wants done, and calling the Level 3 function i_show() with the correct function arguments. The function i_show() routes the video signal generated from the desired memory channel to the video monitor.

6.9 Level 2 Program - hist

```
Usage:  hist [1, 2, 3, p, t, f <image filename>, c, l]
    -1  channel 1 display
    -2  channel 2 display
    -3  overlay display
    -p  point display
    -t  terminal display
    -f <filename>  computes histogram of <filename>
        stores the results in .hs file
    -c  coerces new histogram computation of <filename>
    -l  list out the histogram values
    -h  prints out the help message
    defaults to bar format histogram of 512 by 512 image
    in the DeAnza memory plane 0
```

The original version of this program was written by C. West [20]. The modified version uses the standard protocol for the user interface portion of the program with logical default modes set. Default modes are set such that if the program is executed with no command line arguments, a histogram will be generated from a 512 by 512 pixel image in memory channel 0.

This program provides a number of optional capabilities that make it useful as a general image processing tool. Besides allowing display format and memory channel

selection, "hist" is capable of generating a histogram from an image file. The "hist" program will also work with a non-standard image size, such as 256 by 256 pixels, provided that it is generated from the image file. The new version supports the capability to generate a histogram directly from any memory channel, along with a bar or point display format. The histogram display is removed via a simple Level 2 command "unhist," that disables the overlay memory display. A variety of Level 3 function subprograms are called from this Level 2 program, including i_refs(), i_show(), h_print(), and h_display(). The function subprogram h_display() is passed a histogram array pointer from the main program in order to perform the histogram display operation. When the Level 1 program "datanal" is selected with the "histogram" option, the shell program changes directory to "/dsp/img/L2/bin" and executes the Level 2 program "hist."

6.10 Level 2 Program - tbzoom

```
Usage:  tbzoom [-hczgs012] [0, 1, 2, 3, 4, 5]
    -h  prints out the command description
    -c  enables running coordinate annotation
    -z  enables running zoom power annotation
    -g  enables running gray value annotation
    -s  allows selection of cursor shape
        the shape is specified by an integer 0-5
        in the next argument
    -0  default condition to channel 0 display
    -1  optional channel 1 display
    -2  optional channel 2 display

Cursor shapes are as follows:

0 = full screen crosshair        3 = dashed crosshair
```

1 = single element
2 = crosshair(alphanumeric)
4 = solid crosshair
5 = solid crosshair

There exists a method of executing system commands while this program is running. It involves using the exclamation point in the same manner as in the system file editor <ed>. Consult the section on the editor in the Unix manual Vol. 2 for further details.

The microscopist makes use of "tbzoom" through the Level 1 program "zoom" in order to review images displayed in either memory channel of the DeAnza display. It is also used in the Level 1 program "datanal" with the "gray value" option selected. The "gray value" option causes "tbzoom" to be executed with options selected for a running coordinate, zoom, and gray value annotation display. The program accepts user input through command line arguments in the standard protocol format as described above. Default modes are set such that if the program is executed with no command line arguments, the program will enable the trackball with scroll, zoom, and channel selection capabilities; however, there will be no annotations displayed.

This program provides a number of capabilities that make it useful as a general image processing tool. Besides providing scroll and zoom capabilities integrated with a trackball, it is possible to execute commands while inside the "tbzoom" program by preceding the command string with an exclamation point. The microscopist may also select from six possible cursor shapes.

This Level 2 program calls a number of Level 3 function subprograms, including i_show(), i_curs(), i_scroll(), i_zoom, and a_printf(). The function subprograms i_scroll() and i_zoom() perform the operations necessary to control the hardware scroll and zoom capabilities of the DeAnza. The Level 1 program "zoom" simply changes directory to "/dsp/img/L2/bin" and executes this Level 2 program.

Chapter 7: Level 3 Function Subprograms

7.1 Overview

Chapter 7 describes the Level 3 function subprograms implemented to provide the programmer with a useful set of software tools for the development of image processing programs. The tools developed are grouped into 5 categories:

1. file management
2. data transfer
3. hardware register control
4. annotation display
5. histogram display

The software tools are made available to programmers through the use of an object code library called "/usr/lib/libimg.a." The function call for one of these tools can be inside a main program or another function subprogram. Once the source code for the main program has been completed, the routine is compiled with a compile string of the form shown below:

cc main.c (main program) -limg -o main (output file)

This compile string instructs the compiler to compile main.c and pass the object code generated to the UNIX loader (ld(1)) to create an executable program. The flag "-limg", also passed to the loader, instructs the loader to search through the object code library "/usr/lib/libimg.a" for unknown function references.

Problems may arise as a result of the single pass nature of the search. If there is a function in the object code library that references another function in the library, the referenced function must appear after the referencing function in the object code library. When this is not the case, the loader will fail to include the earlier function, causing the loader to terminate and produce an unexecutable program.

The shell commands "compall" and "mklib" were created to ensure that the object code library "/usr/lib/libimg.a" is a working one pass library. These commands reside in the directory "/dsp/img/L3." Both commands are used to create the library as follows:

compall;mklib <carriage return>

The source code from all the Level 3 functions is compiled into object code by the "compall" command. Once "compall" has successfully completed, the object code library is created with the "mklib" command. The "mklib" command incorporates three UNIX commands into a shell command: "ar(1)," "lorder(1)," and "tsort(1)." The "lorder" command finds an ordering relation for the object code library which is piped to "tsort." The output of "lorder" is processed by "tsort" to order the library such that it is suitable for one pass access by "ld(1)." Finally, the library is created with the "ar" command. The shell command "mklib" is listed below.

```
echo re-creating IMAGE system library:
rm /usr/lib/libimg.a
echo LORDER *.o piped to tsort and archived:
ar cr /usr/lib/libimg.a `lorder *.o | tsort`
rm *.o
chown bin /usr/lib/libimg.a
echo IMAGE library complete
```

The sections below give short descriptions of the Level 3 function subprograms in each of the five categories. The ENTRY section of the function subprogram header is included to indicate the proper function call, along with the function arguments.

7.2 Level 3 File Management Routines

Image files contain the pixel data for an image, along with a 512 byte image header that precedes the image data. The functions below manage image files, allowing the programmer to create, open, and close files. The manipulation of image header flags is also dealt with by functions in this category. The first versions of these Level 3 routines were written by M. Riley [16]. The first two entries below refer to routines created by M. Riley that have remained unmodified since they were last documented in his thesis, and are included here for completeness.

```
ENTRY:  i_creat (filename, nr, nc)
filename -- name of image file
nr -- number of rows in image
nc -- number of columns in image
```

```
ENTRY:  Both of the Level 3 functions below are contained in
the source file "i_flag.c"
i_setf(ip,flag)
ip - struct image (IMAGE) pointer
```

```
flag - flag to be set (see macros in image.h)
i_rsetf(ip,flag)
ip - image pointer
flag to be RESET (macros in image.h)
```

The function subprograms "i_open" and "i_close" have been modified since they were last documented in M. Riley's thesis [16]. To open and close standard files, a user program should use "i_open" and "i_close" functions respectively. The modification made to "i_open" and "i_close" results in these routines more closely resembling the UNIX "open" and "close" functions. Permission access modes were added to the "i_open" routine, and "i_close" was altered to compensate for this addition. Before the protection modes were added, the files were always opened with both read and write permission. The "i_open" function now opens the file first with both read and write permission to check and set flags in the image header. After the header check is successfully completed, the file is closed and opened again with the proper permission mode. The "i_close" compensates for this change by first closing the file, then opening the file with both read and write permission. The image file header is checked, flags are reset, and the image file is closed at the exit of the "i_close" function. The entries for the "i_open" and "i_close" functions are listed below.

```
ENTRY:  i_open(filename, rwmde)
filename - image file name
rwmde - access modes
```

0 - read only
1 - write only
2 - read/write

ENTRY: i_close(ip)
ip - image pointer defined in the include file "image.h"

7.3 Level 3 Data Transfer Routines

Three routines in this category are modified versions of function subprograms originally written by M. Riley [16]. The function subprograms "i_scan," "i_da_to_dsk," and "i_dsk_to_da" were modified to correct the DeAnza control register initialization, and to enable the functions to handle more than one memory frame. The "i_da_to_dsk" routine corresponds to a routine called "i_read" and "i_dsk_to_da" corresponds to "i_display," both documented in M. Riley's thesis.

A program "intl," originally supplied by PAR Technology Inc., was rewritten to properly initialize our DeAnza display system [14]. This proved to be a valuable asset, because a number of discrepancies showed up on the display screen that led programmers to believe that the DeAnza was not performing as documented in both the DeAnza programming manual and the PAR software documentation. The key factor leading to the discrepancies on the display turned out to be the fact that the scrolling registers in the DeAnza must be loaded initially with (0,511), the upper left corner of the display, and not (0,0). Each time the "image" user logs in to the system, the "intl" program is executed to initialize

the DeAnza display system through the use of the ".profile" mechanism provided by the UNIX shell. Once this modification was implemented, the programs were corrected and the DeAnza functions as stated in the documentation. The corrections to the function subprograms correct the starting indicies loaded into the Control Base Coordinate Registers (CBCR).

Without the modifications made to these three function subprograms, each memory frame would require a separate group of functions dealing with the register level interactions between host computer and DeAnza display system. This problem was rectified by specifying a structure, or template of variables external to the Level 3 function thus making the function channel independent. The structure, _deanza, consists of variables representing important channel dependent parameters which may in turn be selectively initialized in source programs through the use of appropriate pseudo-functions. These pseudo-functions act as a movable window, allowing the programmer to selectively initialize a global structure in a source program. The _deanza structure resides in the include file "/usr/include/pick_disp.h." The include file "pick_disp.h" is a modified version of the original file supplied by PAR Technology Inc. [14]. These changes will make the system upward compatible. When a new memory frame or other DeAnza device is purchased for the system, the only changes

necessary, if any, will be to add the additional variable parameters to the structure and recompile the new or existing routines.

The entries for the three routines discussed above are listed below.

```
ENTRY:  i_scan (nr, nc, nf, format, r_flush)
nr -- # rows in image
nc -- # columns in image
nf -- # frames to read in
format -- display in row or column format:
'r' for row
'c' for column
r_flush -- reverse contents of data buffers
upon flushing from dr/buffer hardware:
'n' - forward data
' - reversed data

ENTRY:  i_da_to_dsk(ip, read_flag)
ip -- valid image pointer, returned from
i_open.c found in the image library.
read_flag -- 'r' = read image from display in row format
'c' = read image from display in column format

ENTRY:  i_dsk_to_da(ip, disp_flag)
ip -- valid image pointer
disp_flag -- 'r' display disk file in row format
'c' display disk file in column format
```

The "i_getpix" function was created as a mimic of the UNIX "getchar" function. Given the x and y coordinates of a pixel in a selected memory plane as function arguments, "i_getpix" returns the gray value of that pixel. The "i_getpix" function maps to the DeAnza register and data window, loads the x and y coordinates into the CBCRs, and returns the pixel value from the data page. The entry for the "i_getpix" function is listed below.

```
ENTRY:  i_getpix(x_point, y_point)
```

x_point - x position coordinate on the image display
y_point - y position coordinate on the image display

7.4 Level 3 Hardware Register Control Routines

PAR Technology Inc. supplied the functions "first_page" and "virtual" in their software package, and they remain unmodified. These functions allow the programmer to map variables to physical memory addresses. Entries for both functions are listed below; they reside in the same file "i_map.c"

ENTRY: Both of the functions below are contained in the source file "i_map.c"
first_page() - returns an integer referring to the first available memory mapping register.
virtual(page) - where page is the page number to be mapped to a virtual address.

The function "i_show" is the first in a series of routines written to control the hardware capabilities of the DeAnza image display system. The DeAnza can support 3 full 512 by 512 pixel frames of image memory, along with 1 overlay frame. Channel display selection is controlled through the use of the "i_show" function. The function "i_show" is supplied the channel the programmer wishes to display as an argument. The "i_show" function maps to the DeAnza register page and loads the correct location (denoted IXSPLT in Regdef.h) with the appropriate bit pattern such that the video signal from the channel selected is routed to the monitor. The entry for "i_show" is listed below.

ENTRY: i_show(channel)

where channel is 0, 1, 2, or 3

The "i_refs" function subprogram controls the state of the intensity reference scale for each memory channel. The intensity reference scale may be turned on with 64, 128, or 256 levels of gray. Channel selection is controlled within the main program by using the "sel_chn" pseudo-functions contained in the include file "pick_disp.h." The "i_refs" function is called after the channel selection to select the state and number of levels for that channel's intensity scale. The "i_refs" function maps to the DeAnza register page and loads the correct location (denoted IREF0, IREF1, IREF2, or IREF3 in Regdef.h) with the appropriate bit pattern such that the intensity reference scale for the channel selected is altered as desired. The entry for "i_refs" is listed below.

```
ENTRY:  i_refs(state, level)
where state is off (0) or on (1)
and level is (64), (128), (256).
```

The "i_scroll" function subprogram controls the scrolling state for each memory channel. Scrolling refers to a position shift of an image pixel to the upper left hand corner of the image. Normally the coordinates of the upper left corner are (0,511). The "i_scroll" function permits the programmer to specify the coordinates of the pixel to be placed in the upper left corner. Wrap around refers to the sections of the image that are scrolled off screen. These

are "wrapped around" in the x and y direction as specified by the programmer. The wrap around may also be blanked where indicated by the programmer, blackening the portion of the image that is wrapped. For further discussion of the scrolling capabilities of the DeAnza display system, the reader is referred to the DeAnza Programming Manual [12]. Channel selection is controlled within the main program by using the "sel_chn" pseudo-functions contained in the include file "pick_disp.h." The "i_scroll" function is called after the channel selection to scroll the image to the desired position. The "i_scroll" function maps to the DeAnza register page and loads the scroll control registers with the bit pattern containing the x and y coordinates, wrap, and blank information requested by the programmer. The entry for "i_scroll" is listed below.

```
ENTRY:  i_scroll(x, y, xmode, ymode)
x - starting(left) x coordinate
y - starting(left) y coordinate
xmode - 0 - 7  defined as follows:
0 - no blank on vertical wrap or horizontal wrap
blanking after horizontal wrap around
1 - no blank on vertical wrap or horizontal wrap
blanking before horizontal wrap around
2 - no blank on vertical wrap, blank on horizontal wrap
blanking after horizontal wrap around
3 - no blank on vertical wrap, blank on horizontal wrap
blanking before horizontal wrap around
4 - blank on vertical wrap, no blank on horizontal wrap
blanking after horizontal wrap around
5 - blank on vertical wrap, no blank on horizontal wrap
blanking before horizontal wrap around
6 - blank on vertical wrap, and blank on horizontal wrap
blanking after horizontal wrap around
7 - blank on vertical wrap, and blank on horizontal wrap
blanking before horizontal wrap around
```

ymode - 0 or 1 defined as follows:
0 - blank after vertical wrap around
1 - blank before vertical wrap around

The function subprogram "i_zoom" controls the zoom state for each memory channel. Four hardware zoom factors are available: 1X, 2X, 4X, and 8X. The hardware zoom is accomplished through a direct mapping of pixels:

```
1X = a 1 pixel to 1 pixel mapping (No zoom)
2X = a 1 pixel to 4 (2 by 2) pixel mapping
3X = a 1 pixel to 16 (4 by 4) pixel mapping
4X = a 1 pixel to 64 (8 by 8) pixel mapping
```

Channel selection is controlled within the main program by using the "sel_chn" pseudo-functions contained in the include file "pick_disp.h." The "i_zoom" function is called after the channel selection to zoom the image to the desired magnification. The function "i_zoom" maps to the DeAnza register page and loads the scroll control registers with the bit pattern containing the zoom factor requested by the programmer. The entry for "i_zoom" is listed below.

```
ENTRY:  i_zoom(power)  power refers to the valid hardware
zoom factors.  The valid parameter choices are:
1 --- 1 to 1   (no zoom)
2 --- 2 to 1     zoom
4 --- 4 to 1     zoom
8 --- 8 to 1     zoom
```

The function subprogram "i_curs" controls the cursor display state in the DeAnza display system. One or two cursors can be displayed in one of eight formats described in the DeAnza Programming Manual [12]. Cursor state is controlled within the main program by using the

pseudo-functions "cur_enbl" and "cur_dsbl" contained in the include file "pick_disp.h." The "i_curs" function is called after the cursor state selection to actually load the correct registers with the patterns selected in the main program. The function "i_curs" maps to the DeAnza register page and loads the five cursor control registers with the bit pattern chosen by the programmer in the main program. The entry for "i_curs" is listed below.

```
ENTRY:  i_curs() - this function operates on the external
structure
_cursor defined in /usr/include/Regdef.h
      struct _cursor
      {
      int cs1x;
      int cs1y;
      int cs2x;
      int cs2y;
      int csctl;
      };
```

7.5 Level 3 Annotation Display Routines

The "a_printf" function subprogram is similar to the C language function call "printf" used to format output to a file or terminal. It is useful to print formatted character strings on the monitor (versus printf() for a terminal output) in any position on the screen. The user must specify the row (1-25) on which the line is to begin, and specify the column (1-80) where the string should begin. The user has the option of black or white characters. The character string can be passed as a string pointer or as a string enclosed in double quotes, along with any of the

standard conversion characters available with the "printf" function. The arguments for conversion are appended to the argument list in the same manner as "printf." The function "a_printf" maps to the DeAnza register page, converts the arguments supplied to a valid character string, writes the string to the annotation memory in the DeAnza, and enables display of the character string on the monitor.

The "a_clran" function subprogram is used to erase either a line of characters, or the entire monitor screen. The user must specify the row (1-25) for a line erase, and can optionally erase the entire screen. It is also possible to disable the annotation display while the character string remains in the annotation memory. The entries for "a_printf" and "a_clran" are listed below.

```
ENTRY:  a_printf(row, column, option, fmt, args)
row - integer 1 to 25
column - integer 1 to 80
option - w or b: white or black characters
fmt - a string and/or argument format like printf()
args - arguments

ENTRY:  a_clran(row, options)
row - an integer 1 thru 25
options - e: erase the specified line
a: erase the entire screen
d: disable the annotation,
but leave the characters in memory
```

7.6 Level 3 Histogram Display Routines

The function subprograms "h_display" and "h_print" are second versions of routines originally written by C. West [20]. The "h_display" function is used to generate a

histogram display on the overlay memory plane, while the "h_print" generates a compressed histogram display on a terminal. The reader is referred to C. West's B.E. project report for detailed information on version 1 of both these routines. Two display formats, "bar" and "point", are now available through the "h_display" function. An annotation was added to the "h_display" function, indicating the maximum histogram value and associated gray value. Modifications made to the "h_print" function were essentially cosmetic in nature; an include file was eliminated. The entries for both "h_display" and "h_print" are given below.

ENTRY: h_display(hist), hist is pointer to array of long integers

ENTRY: h_print(hist) where hist is a pointer to an array of long integers

Chapter 8: Level 4 Operating System Interface Code

8.1 The Include Files

Three files are used extensively throughout all of the program code in Levels 2, 3, and 4:

1. /usr/include/Regdef.h
2. /usr/include/pick_disp.h
3. /usr/include/image.h

These files are incorporated into program source code via the "include" mechanism supplied by the C preprocessor portion of the C compiler. A section labeled "INCLUDE FILES" exists in the program header for each routine in Levels 2, 3, and 4. The include files are "included" in the section under this heading using a statement with the following syntax

```
#include  <Regdef.h>
```

The angle brackets are required, since they instruct the C preprocessor to search the directory "/usr/include" for the file "Regdef.h." The include file "image.h" remains unchanged since last documented in Riley's thesis [16].

Two of the include files, "Regdef.h" and "pick_disp.h," are modified versions of files of the same name supplied by PAR Technology Inc. [14]. The include file "Regdef.h" provided definitions for parameters used on an IP5324 color display or an ID1124 1024 by 1024 pixel monochrome display. The file "pick_disp.h" provided a means of selecting between an IP5324 color display or an ID1124 1024 by 1024 monochrome

display. Neither of these files were useful in their original form, since we use a DeAnza ID5400 512 by 512 monochrome display. Extensive modifications were made to accommodate our particular display system.

The include file "Regdef.h" symbolically defines the DeAnza(ID 5400) display registers as integer offsets relative to the base of the address window and also defines several commonly used constants associated with the displays. Symbols are also defined for accessing the trackball via its special device file. Structures are defined for initializing the trackball mode, for reading the trackball, for loading intensity transformation tables, for flickering images, and for using registers and parameters that vary from one display to another. These "definitions" are made by using the "define" macro substitution mechanism supplied the C Preprocessor in the following manner:

```
#define  IDREGPG  07000
```

This statement defines the starting address of the DeAnza register page in core clicks, where one core click is 32 words or 64 bytes; thus the starting address is 112K words or 224K bytes.

The include file "pick_disp.h" contains a structure array definition for "_da" and several pseudo-function definitions for selecting a portion of that structure array such that the structure pointer "da" points to the desired subsection. The structure array is pre-initialized with register

addresses and display parameters that vary from one display channel to another. The pseudo-functions simplify selection of the appropriate element of the structure array. The simplest analogy for what is created in this include file is a table of variables defined by the structure definition "_deanza" with several columns of data beside the variables. Each data column contains register specific information for each memory channel. By calling a pseudo-function such as "sel_chn0()," the programmer is filling the contents of the variables within the structure with the data contained in one particular column. The pseudo-function is called in the same manner as a normal C language function:

```
sel_chn0();
```

8.2 DR-11W Device Driver Modification

The following modification was made to the "drioctl" function within the device driver that was written by M. Riley called "dr.c." The driver was modified to provide a 4 millisecond signal pulse in order to compensate for mirror flyback time.

```
case START:
DR->i_odr = START_SYNC;
temp = 10000;                /* modification */
while(temp != 0)  temp--; /* modification */
DR->i_odr = 0;
break;
```

The while loop increases the pulse duration of the START signal supplied to the laser scanning subsystem of the LSPM microscope.

8.3 The Trackball Device Driver

UNIX treats all peripheral devices as special files. The special file for the trackball is called /dev/tb and it is created by the following command:

```
/etc/mknod tb c 8 0 where

tb - is the name of the special device file,
c - indicates the special file is for a character
device,
8 - is the major device number,
0 - is the minor device number.
```

The special file entry appears as follows in the directory "/dev":

```
4 crw-rw-rw- 1 adm        8,  0 Jul 26  1983 /dev/tb
```

The major device number for the trackball provides the critical link between the programmer who opens the special file, and the operating system kernel that processes all requests for use of the trackball. The file "/usr/sys/conf/c.c" contains a structure pointer "cdevsw" that is directly incorporated into the operating system kernel at load time via the object code file "c40.o." The initialization of the "cdevsw" pointer contains several lines of character strings that refer directly to the device driver function names for a given major device number. The position of the character string is directly related to the number associated with that string; thus provided that all character strings are in the correct format, the major device number associated with a character string is set to the string position within the initialization sequence. The

"cdevsw" structure pointer initialization is given below to clarify the above description.

```
struct  cdevsw  cdevsw[] =
{
klopen, klclose, klread, klwrite, klioctl, nulldev, 0,
/* console = 0 */
nodev, nodev, nodev, nodev, nodev, nulldev, 0,
/* pc = 1 */
nulldev, nulldev, mmread, mmwrite, nodev, nulldev, 0,
/* mem = 2 */
nulldev, nulldev, r611read, r611write, nodev, nulldev, 0,
/* r7 = 3 */
dzopen, dzclose, dzread, dzwrite, dzioctl, nulldev, dz11,
/* dz = 4 */
syopen, nulldev, syread, sywrite, sysioctl, nulldev, 0,
/* tty = 5 */
tmopen, tmclose, tmread, tmwrite, nodev, nulldev, 0,
/* tm = 6 */
dropen, drclose, nodev, nodev, drioctl, nulldev, 0,
/* dr = 7 */
tbopen, tbclose, tbread, nodev, tbioctl, nulldev, 0,
/* tb = 8 */
nodev, nodev, nodev, nodev, nodev, nulldev, 0,
/* adac = 9 */
0
};
```

The link between the operating system kernel and the trackball device is the device driver "tb.c." This device driver has five separate function subprograms:

tbopen() - used when "/dev/tb" is opened

tbclose() - used when "/dev/tb" is closed

tbread() - used when "/dev/tb" is read from

tbioctl() - used for input-output control on "/dev/tb"

tbintr() - the interrupt service routine

When the trackball requires attention, the current task being performed by the PDP-11/60 CPU is interrupted, and the interrupt service routine "tbintr"is initiated. Once the

interrupt service routine is entered, a buffer structure is filled with the current position of the trackball cursor, along with switch status. This buffer is common to both "tbintr" and "tbread" functions; hence a read call to the trackball device supplies the programmer with cursor position and switch status information.

The sequence of operations required to service a trackball interrupt are outlined below.

1. The CPU gives up control of the UNIBUS, priorities permitting.
2. When the trackball gains control of the UNIBUS, it sends the CPU an interrupt command and the memory address 0234 (octal) which contains the address of the interrupt service routine "tbintr." The memory address 0234 is called the interrupt vector address. The location (vector address + 2) is used as a new Processor Status Word (PSW).
3. The CPU stores the current Processor Status (PS) and Program Counter (PC) in temporary registers.
4. The interrupt vector provides the address for the new PC and PS. The old PS and PC are then pushed onto the current stack, and the service routine "tbintr" is entered.
5. When the interrupt service routine "tbintr" has finished, it causes the CPU to resume the interrupted process. This is done by executing the Return from

Interrupt instruction, which pops two words from the current stack and uses them to load the PC and PS registers.

This sequence is the hardware/software interface between the trackball device and the user software. The interrupt vector information is incorporated into the operating system kernel at load time (ld(1)) through the object code file "140.o," which is assembled from the file "1.s."

The last link in the chain is the device driver library called "/usr/sys/dev/LIB2.40," where the object code "tb.o" is archived. The trackball device driver source code file is "/usr/sys/dev/tb.c," which is compiled with the following command string:

```
cc -c -O tb.c
```

The resulting file "tb.o" is archived with following command:

```
ar cr /usr/sys/dev/LIB2.40 tb.o
```

The device driver is incorporated into the last load (ld(1)) command in the sequence of events that generate an operating system kernel.

The source code for the device driver "tb.c" may be found in Appendix 2. The driver was modelled after the device driver "da.c" supplied by PAR Technology Inc. [14]. The driver "da.c" was not implemented into the image processing package because it required too much space in the operating system kernel. The size limit for the operating

system kernel is 49,152 bytes (6 pages X 8K bytes/page). In order to meet this constraint, another device such as the tape drive would have to be eliminated from the operating system kernel; this was judged to be unacceptable. The next section addresses the implementation of the "unixlspm" version of the operating system kernel that handles all the required devices.

8.4 Operating System Kernel Modification and Regeneration

The operating system required modification to incorporate the trackball, DR-11W, tape drive, and the DZ-11 into a single system kernel. Normally, the new device would simply be added to the current system configuration; however the system kernel is constrained to fit into 6 pages of memory or 49,152 bytes. Extensive modification was made to the file "mkconf.c" in order to reduce the code that is actually incorporated into the system kernel. This modification made it necessary to recreate the entire directory "/dev." Modifications were also made to files in the directory "/usr/sys/sys." The code concerning the "mpx" (multiplex) filesystem was eliminated from all system code through the use of the conditional compile statement "#ifdef." All these modifications led to the creation of an operating system kernel called "unixlspm" with a size of 49,136 bytes (16 bytes to spare!).

The source code listing for the file "mkconf.c" may be found in Appendix 2. Essentially, the modifications to this

file were limited to the structure pointer initializations for "cdevsw" and "bdevsw." The executable version of "mkconf" accepts input in the form of a "configuration table," such as "/usr/sys/conf/conf.tbl/lspmconf" listed below.

```
r7
root r7 1
swap r7 1
swplo 6000
nswap 2724
dr
tm
tb
dz 8
```

The configuration table contains a list of the devices to be incorporated into an operating system. The resulting output from "mkconf" is the files "c.c" and "l.s." Before the modification was made to "mkconf.c," it generated a great deal of unnecessary code in "c.c" and "l.s" concerning devices that we would not (and could not in some cases!) ever use. These modifications resulted in the elimination of 1021 bytes of code from "c.c." This reduction had a direct effect on the size of the system kernel.

The directory "/dev" contains all the special files for devices that the system recognizes. The special files indicate whether the device is a character or a block device, along with the major and minor device numbers. Since the structures "cdevsw" and "bdevsw" were modified in the file "mkconf.c," the special file entries containing the major and minor device numbers had to be changed. The major

device number is directly related to the position of the device in the "cdevsw" and "bdevsw" structures. The special files in the directory "/dev" were altered such that the major and minor device numbers corresponded to the correct device file.

The operating system is recreated by entering the directory "/usr/sys/conf" and typing the command:

```
make unixlspm
```

This command will only work correctly for the "root" user, otherwise any user could write over the existing system kernel. It uses the UNIX utility "make" which is a smart program maintainer [10]. The listing below is an edited portion of the "makefile" that is concerned with the system kernel regeneration.

```
unixlspm:
        make dr - installs the DR-11W device driver
        make machdep_DEANZA - installs DeAnza dependent code
        echo making unix40 ...
        mkconf < conf.tbl/lspmconf - creates "c.c" and "l.s"
        make unix40 - actually generates "/unixlspm"
        size unix40 - sizes the system kernel
        chmod 755 unix40
        echo moving unix40 to "/unixlspm" ...
        mv unix40 /unixlspm
        echo restoring the baseline unix form of machdep.c
        make machdep_base
        sh -c 'sync ; exit 0'
        echo /unixlspm complete:
```

This listing is included to reinforce the fact that system regeneration is made quite simple by the "make" utility; however, system regeneration is much more complicated than this one command indicates. If modification are made to

this kernel in the future, an experimental version of the kernel should be created and tested before writing over the "/unixlspm" version of the operating system kernel.

Chapter 9: Conclusions and Recommendations

An image processing software package has been developed for the LSPMM. This software provides the microscopist with the capability to scan, save, and display a 512 by 512 pixel image. The image may be scanned to, saved from, or displayed in either of the two DeAnza image display memory planes. The microscopist has also been supplied with a versatile image reviewing mechanism combining the trackball, scroll, and zoom capabilities of the DeAnza display into one command called "zoom." Considerable effort has been made to incorporate all of the devices useful for image processing into a single operating system kernel. This alleviates the problem of taking down one operating system and bringing up another version in order to copy image files onto magnetic tape. A four level hierarchy provided the organizational framework for the program development. This framework should prove useful for the development and implementation of additional software in the future. All of the software bugs known to this programmer have been eliminated; however, bugs not known may crop up in the future as the image processing software is used more often.

Regarding additional uses for this software, two different applications have already surfaced. The first application concerns the use of the DeAnza for a large data buffer to hold electrocardiogram signal data from an analog-to-digital converter. The Level 3 routine

"i_dsk_to_da.c" was modified and incorporated into another high level program to accomplish this task. The other application demonstrates that this software is useful for general image processing. A magnetic tape containing images of various sizes was obtained from University of Southern California. These images were read in and stored in a separate image catalog called "usc." This demonstrated that the software is not limited to the 512 by 512 pixel images generated by the "scan" command.

Having gained some experience in writing image processing software, several recommendations are made regarding future work:

1. If the Level 1 command "arithops" described in this thesis is implemented in the future, a pipeline array processor should be purchased to provide the capability of performing high speed arithmetic operations.
2. The trackball provides the user and programmer with an elegant tool for interaction with the DeAnza image display. Further development of this tool should be pursued in areas such as combining the trackball device with overlay plane graphics for feature outlining.
3. Once the pipeline array processor has been obtained, numerous image processing techniques can be implemented that will operate at rates approaching

real-time (1 frame time = 1/30 sec). This includes the development of time domain filtering under the Level 1 command "filter," and area statistics operations under the Level 1 command "datanal."

4. The last recommendation refers to moving the image processing software and DeAnza display system to a VAX-11/750 running Berkley UNIX version 4.2. This problem is addressed because Thayer School has purchased a VAX-11/750 to be installed in the summer of 1984 that will run under Berkley UNIX Version 4.2. The DeAnza display system and the image processing software will be transferred to this machine, along with the other peripherals currently available on the PDP-11/60. Once the DeAnza display system is integrated into the hardware via the UNIBUS controller, a base memory address will be established. Modification will have to be made to the include file "Regdef.h" to provide all the image processing programs with the correct base address for the DeAnza register and data pages, and all of the programs in Levels 2, 3, and 4 will have to be regenerated.

APPENDIX 1

A1.1 Software Demonstration

This section contains a demonstration session with the image processing software. The commmands typed by the user and the computer output are offset from comments by an arrow (. ->). The first thing done by the user is to turn on the computer terminal and login to the computer as "image."

```
->  login: image
->  ********************************
->  *****  SYSTEM: unixlspm    *****
->  ********************************
->  Welcome to Unix
->  Mon May 21 03:48:34 EDT 1984
->  Let a fool hold his tongue and he will pass for a sage
->  $ ls
->  arithops
->  contrast
->  datanal
->  display
->  filter
->  graphics
->  help
->  remove
->  restore
->  save
->  scan
->  smooth
->  zoom
->  $
```

This sequence logs the user into the system and lists the Level 1 commands. The monitor display might look as shown in Figure A1.1. Suppose the user wanted to scan an image from the LSPMM. The following sequence would perform this task.

```
->  $ scan
->  Input the number of frames to scan:  1
->  Scan image in row or column format ('r' or 'c'):  c
->  Do you want the mirror image of the input data (y or n):  y
```

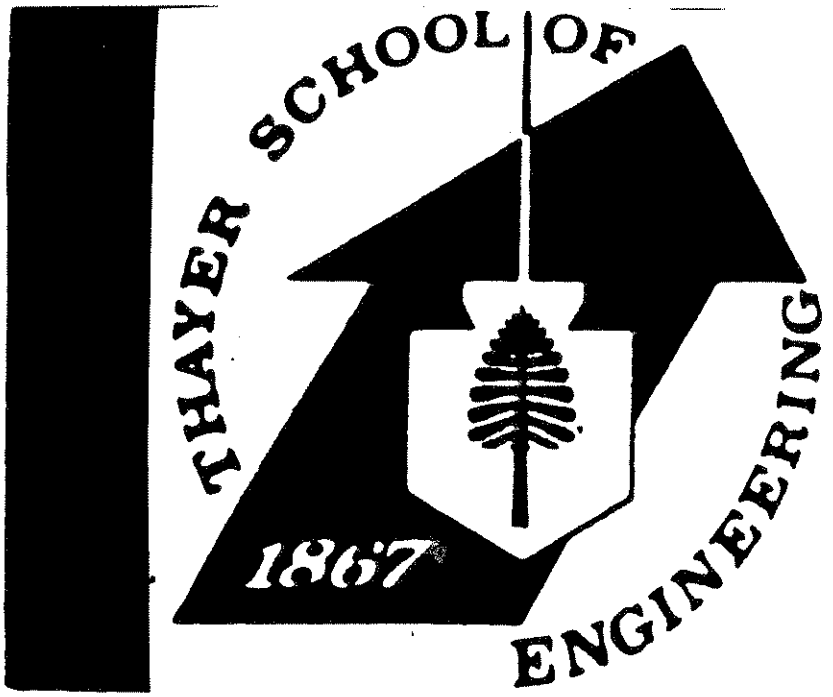


FIGURE A1.1 The Thayer School Logo

-> Select the memory channel scan destination (0, 1, 2): 0

-> Broadcast Message ...

-> Image scanning in 5 seconds.....

-> Image scanning completed.

-> $

This sequence could also be performed with the single command:

-> $ scan 1

To display a caption on the image, the following command is executed.

-> display caption bucal epi pm

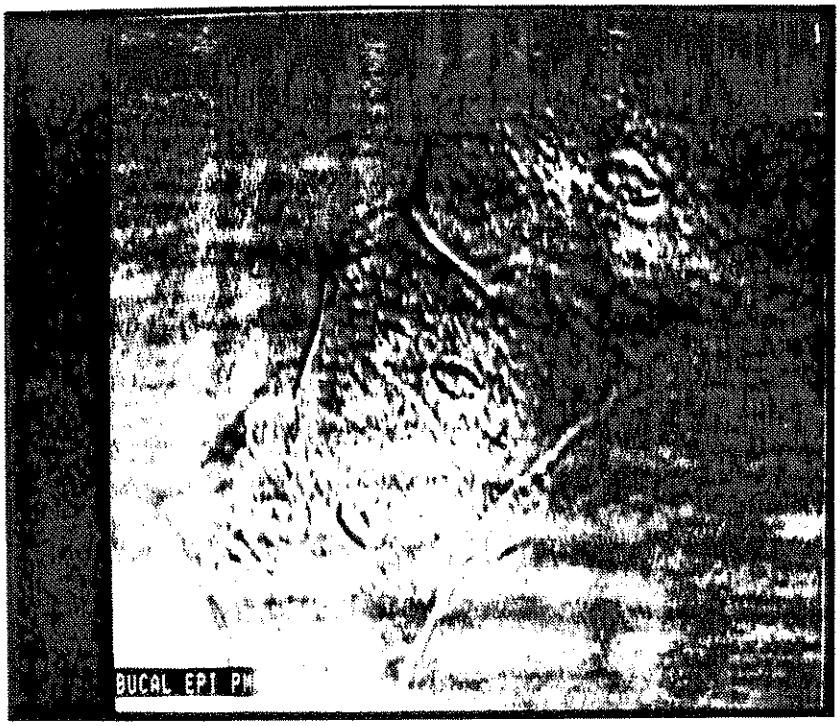


FIGURE A1.2 A Scanned Image

The monitor display after the above commands are executed is shown in Figure A1.2. The image that was scanned may be stored in a disk file by using the save command as outlined below.

```
->  $ save
->  Input the image filename:  i/bucal.pm
->  Input number of rows & columns:  512 512
->  Read from DeAnza display or previous image file (d or f):  d
->  DeAnza read in row or column format (r or c):  r
->  Select the memory channel to read from (0, 1, 2):  0
->  $
```

The image can also be saved with the single command:

```
->  $ save i/bucal.pm
```

The image "i/bucal.pm" can be removed from the image catalog "i" with the following command:

```
->  $ remove image i/bucal.pm
```

Suppose the microscopist wants to list the files in the image catalog "i." The following command is executed:

```
->  $ display catalog i
->  total 10260
->    513 100x70amp
->    513 16x450amp
->    513 16x450brf
->    513 16x450pmon
->    513 davef.1
->    513 grayscale
->    513 ground
->    513 logo
->    513 ohbaby
->    513 on.1.pm
->    513 on.2.bfd
->    513 on.2a.pm
->    513 on.2a.pol
->    513 on.2b.pm
->    513 on.2b.pol
->    513 on.2c.pm
->    513 on.2d.pm
```

To display an image saved in a disk file, the following command is used:

```
-> $ display
-> Input the image file name:  i/on.2c.pm
-> Display the image in row or column format ('r' or 'c'):  r
-> Select the memory channel you wish displayed (0, 1, 2):  0
-> $
```

The same operation can be accomplished simply by typing:

```
-> display i/on.2c.pm
```

A caption can be displayed by typing:

-> display caption on.2c.pm

Figure A1.3 shows what is now displayed on the video monitor.

The "zoom" command provides the microscopist with a simple image reviewing mechanism. The command is executed as shown below.

```
-> $ zoom -cz
-> The trackball module has a ball and three switches.
-> Pressing the white switch on the left will cause the
-> zoom magnification to toggle -- 1, 2, 4, 8 powers.
-> Pressing the white switch on the right will cause the
-> display channels to toggle -- channel 0 or 1.
-> Pressing the red switch in the center will allow 2 images
-> to be reviewed independently.  You may toggle between
-> reviewing sessions using the red switch.
-> There exists a method of executing system commands while
-> this program is running.  It involves using the exclamation
-> point in the same manner as in the system file editor <ed>.
-> Consult the section on the editor in the Unix manual Vol. 2
```

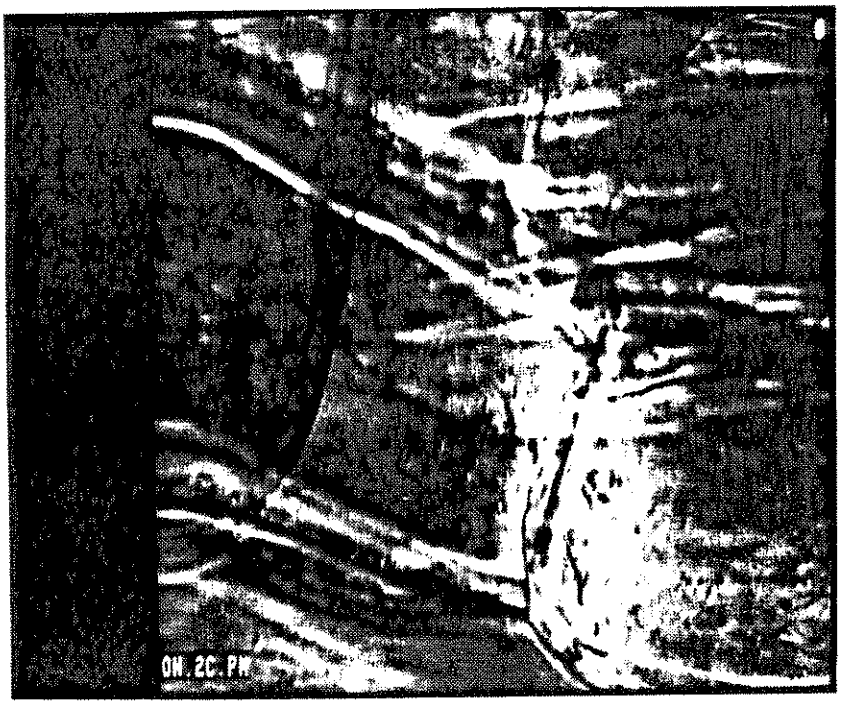


FIGURE A1.3 A Saved Image Displayed

-> for further details

-> To end this session of tbzoom, type q and return.

At this point the microscopist is able to use the trackball and switches to scroll and zoom the image. A sample of the monitor display is shown in Figures A1.4 and A1.5. To end the session of the "zoom" command, the user simply types "q" as shown below.

```
-> q
-> $
```

In order to generate a histogram of the image "i/on.2c.pm" that has been displayed as outlined above, the following command is typed:

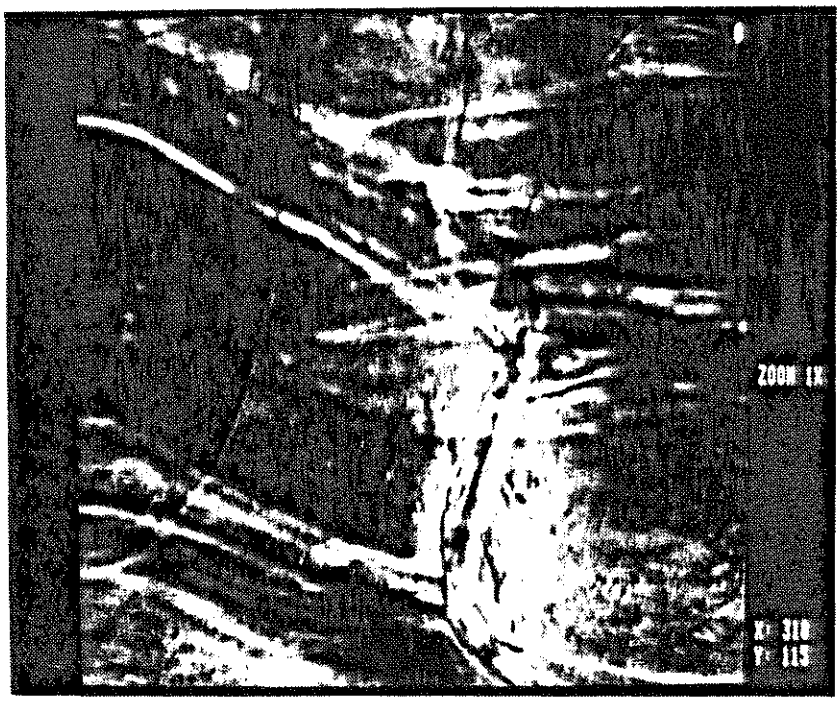


FIGURE A1.4 Zoom Magnification 1

FIGURE A1.5 Zoom Magnification 2

-> datanal histogram

The histogram is displayed on the monitor as shown in Figure A1.6. If the microscopist wants to find the gray value at a certain point in the image, then the following command is executed:

```
-> datanal gray value
-> The trackball module has a ball and three switches.
-> Pressing the white switch on the left will cause the
-> zoom magnification to toggle -- 1, 2, 4, 8 powers.
-> Pressing the white switch on the right will cause the
-> display channels to toggle -- channel 0 or 1.
-> Pressing the red switch in the center will allow 2 images
-> to be reviewed independently.  You may toggle between
```

-> reviewing sessions using the red switch.
-> There exists a method of executing system commands while
-> this program is running. It involves using the exclamation

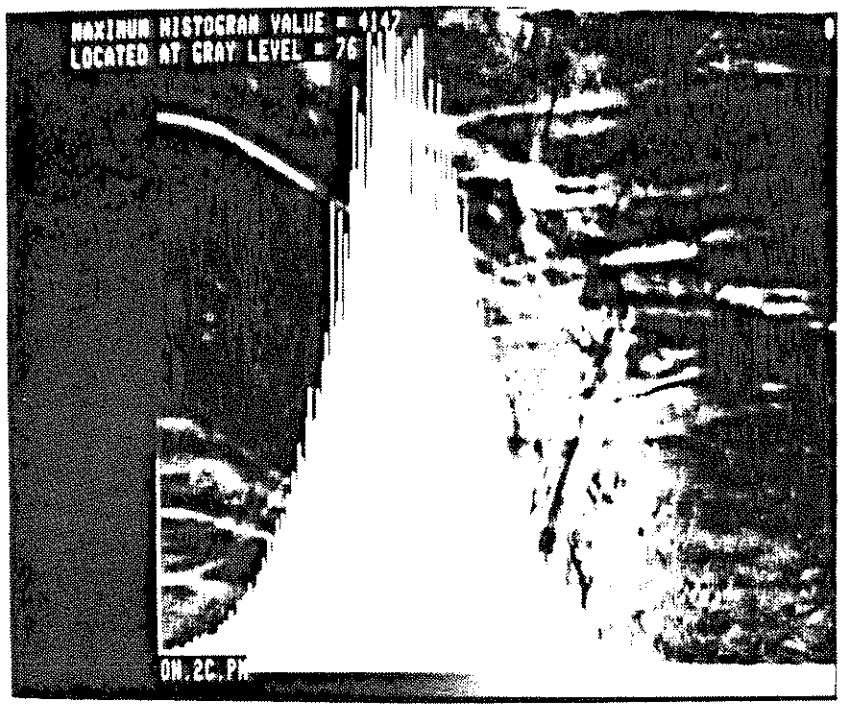


FIGURE A1.6 Histogram Display

-> point in the same manner as in the system file editor <ed>.
-> Consult the section on the editor in the Unix manual Vol. 2
-> for further details

-> To end this session of tbzoom, type q and return.

As shown in Figure A1.7, the gray value at the point under the cursor is displayed on the monitor as an annotation. This command is ended by typing "q" as shown

-> q
-> $

Finally the user can log out of the computer by typing "login."

-> login

-> login:

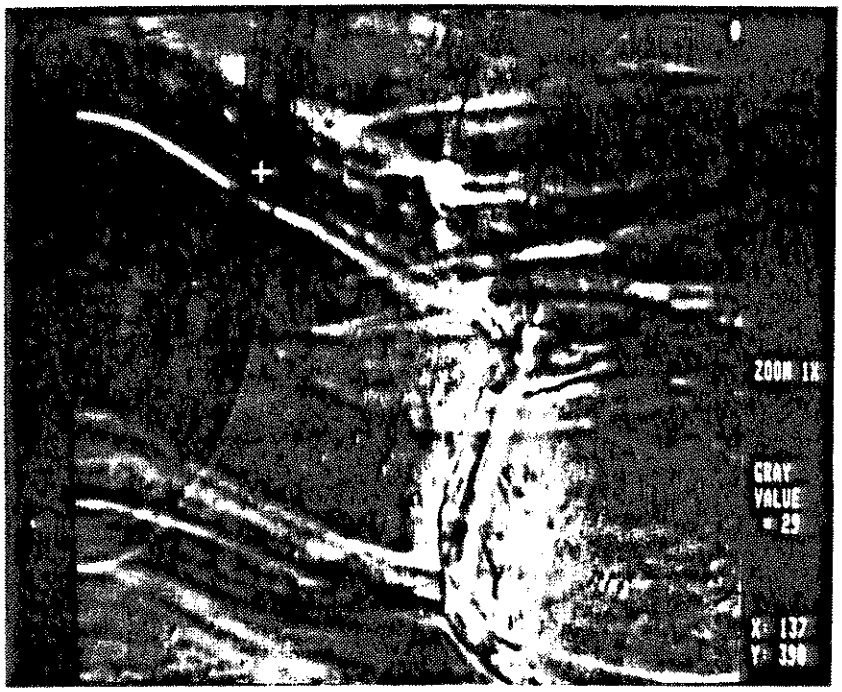


FIGURE A1.7 Datanal Gray Value Display

A1.2 The UNIX Manual Pages for Level 1 and Level 2