AN INTERACTIVE DIGITAL IMAGE PROCESSING SYSTEM

by

GEORGE FAWCETT JR.

Submitted in Partial Fulfillment
of the Requirements for the
Degree of Bachelor of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
January, 1975

Signature of Author. . . . .
Department of Electrical Engineering, January 25, 1975

Certified by . . . .
Thesis Supervisor

Accepted by. . . . .
Chairman, Departmental Committee on Theses

AN INTERACTIVE DIGITAL IMAGE PROCESSING SYSTEM

by

George Fawcett Jr.

ABSTRACT

An interactive system for processing digital images has been developed. The images are recorded on magnetic tape with a two-dimensional silicon vidicon photometer. Distortion corrections and calibrations are described. The Princeton 801 graphics terminal is presented as the major I/O device of the system. A command language and an interpreter for the language are developed. The interactive environment is implemented on an IBM 360/65 computer system with several modifications to the operating system. Finally, an example of a terminal session showing the system use is presented.

Thesis supervisor:
Thomas B. McCord
Associate Professor of Planetary Physics

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor, Tom McCord, for his support throughout this project. Working for him for the last three years has been a pleasurable experience. There are several people who have helped during this work. Harvey Baumel, who was the systems programmer at the LNS computer when this work started, contributed much of his time to helping me understand the operating system and how to modify it. Grant Snellen was instrumental in the initial installation of the Princeton terminal and helped iron out its bugs. I am deeply indebted to Paul Kinnucan who has worked closely with me throughout the entire project. His programming experience with the batch system and his suggestions for implementation of the interactive system were invaluable. Finally, a word of thanks must go to Larry Petro who made the project possible when he started me working at MITPAL in 1971.

TABLE OF CONTENTS

ABSTRACT..2

ACKNOWLEDGEMENTS................................3

LIST OF FIGURES.................................6

1. INTRODUCTION.................................7

2. BACKGROUND...................................9

2.1 IMAGE PROCESSING............................9

2.2 EXISTING SOFTWARE...........................12

3. PRINCETON 801 TERMINAL.......................14

3.1 HARDWARE FEATURES...........................14

3.2 SOFTWARE SUPPORT............................15

4. COMMAND LANGUAGE.............................18

4.1 COMMAND LANGUAGE PROCESSOR..................18

4.2 PROCESSOR ROUTINES..........................19

4.2.1 CONTROL ROUTINES..........................20

4.2.2 PARAMETER PROCESSORS......................21

4.2.3 COMMON FUNCTIONS..........................24

5. AVAILABLE COMMANDS...........................28

5.1 ACTIVE IMAGE FUNCTIONS......................28

5.2 DISK MAINTENENCE ROUTINES...................30

5.3 OTHER COMMANDS..............................31

5.4 JUSTIFICATION FOR THE ACTIVE IMAGE..........31

6. SYSTEM STRUCTURE.............................34

6.1 IPROCS......................................35

6.2 IMPLEMENTATION OF BREAK.....................38

6.3 CONTROL ROUTINE.............................40

6.4 OTHER ROUTINES..40

6.5 EXAMPLE...41

7. PERFORMANCE AND IMPROVEMENTS............................48

APPENDICES..50

A. OPERATING SYSTEM MODIFICATIONS..........................51

A.1 UNIT CONTROL BLOCK......................................51

A.2 ATTENTION INTERRUPT HANDLER.............................54

A.3 PL/1 MULTITASKING.......................................55

B. DISPLAY CAPABILITIES....................................59

LIST OF FIGURES

1. FLOWCHART OF DISTORTION CALIBRATIONS....................11

2. RELATIONSHIPS BEWTEEN COMMAND PROCESSOR ROUTINES.......22

3. RELATIONSHIPS BETWEEN ACTIVE IMAGE, PERMANENT FILE,
   AND ACTIVE IMAGE COMMANDS..............................29

4. CONTROL STRUCTURE......................................37

CHAPTER 1

INTRODUCTION

A two dimensional silicon vidicon astronomical photometer(*) has been developed at the M.I.T Planetary Astronomy Labratory (MITPAL). It has been in use at the telescope since August, 1971, primarily for direct photometric imaging through filters. The vidicon tube, the heart of the system, consists of an array of photosensitive silicon diodes. Photons are converted to charge carriers that discharge the back-biased diodes. After exposure, an electron beam recharges the diodes creating a current through the target biasing circuit. This current is the video signal which is amplified and recorded onto magnetic tape in digital form.

The signal on tape is a digital image in the form of intensity versus X-Y coordinates. The image must now be subjected to distortion corrections, calibrations, and other reductions, and then displayed in a video format. These corrections require an image processing system of some sort. An image processing system has been written that runs as a

----

(*) Thomas B. McCord and James A. Westphal, "Two-Dimensional Silicon Vidicon Astronomical Photometer," APPLIED OPTICS, Vol. 11, March 1972, pages 522-526.

Thomas B. McCord and Jeff Bosel, "Silicon Vidicon Astronomy at MIT", proceedings of the symposium "Astronomical Observations with Television-Type Sensors", U. of B.C., Vancouver, May 15-17, 1973.

batch job at the LNS computer facility. However the batch mode of operation is not suited to some of the image processing done. Therefore, an interactive system was developed.

This thesis is the result of the work the author has put into developing the interactive system. The system will be described and its peculiarities and shortcomings will be presented. This thesis, like the system, is modular in structure. Separate chapters are devoted to the major but logically independent parts of the system. First, the existing batch system will be presented as it it is foundation for much of this work. Next, the Princeton terminal will be described. The command language processor, which must operate in real time, is described. The commands themselves are presented next. Finally, the system components are merged into the total system. There are two chapters in the appendix. The modifications to the LNS system to support the interactive system are discribed. Also, the display capabilities of the Princeton terminal and features not yet implemented are presented. It is hoped that this thesis will be helpful to those using the system and those who continue its developement.

CHAPTER 2

BACKGROUND

This chapter is intended as an introduction to the image processing done at MITPAL and the work preceeding that of the author. The image processing systems run on the computer facility at the Labratory for Nuclear Sciences (LNS) at M.I.T. The LNS facility includes an IBM 360/65 computer with 512K bytes of core storage. Attached to the 360 through two selector channels are four 2311 mountable disk drives, four dual density 2314 disk drives (system resident), six 2400 tape drives, two 1403 printers, and a 2540 card read/punch. A Princeton 801 graphics terminal (described in Chapter 3) was added for the interactive system. The operating system currently running is MFT using HASP for spooling input and output. There are two user partitions, presently defined at 158K bytes and 220K bytes.

2.1 IMAGE PROCESSING

Images are recorded by the Vidicon system as a series of 12 bit integers. This allows intensity ranges from 0 to 4095. Initially, this format has to be converted to a format suitable for the 360. Two formats are used, 16 bit integers and 32 bit floating point numbers. The integer format conserves space, while the floating point format preserves precision during operations.

The images are stored on 2311 disk packs during processing. Several disk packs are maintained allowing each user to keep and operate on his own library of images. The disk packs contain a directory with pointers to images and free space, a descriptor segment on each image, and the images themselves. The images are stored by rows, with each block of data containg one or two rows, depending on the format. The standard image consists of 250 - 256 point rows, though this format is variable. Typically, a disk pack can store 40 standard size images in floating point format, and twice as many in integer format. The formats can be freely intermixed on a disk pack.

The first operation in the image processing is moving the image from the tape to the disk. The image is read from the tape, assigned a name, and written onto the disk. The user can supply up to 256 bytes of information on the image for the descriptor segment. The image is now ready for corrections and calibrations.

The first correction done is the subtraction of the dark field. This is an image taken with the shutter closed. It is subtracted to compensate for dc bias in the video circuit and for filament back lighting. Next, a flat field correction is done. A flat field is an image taken of a uniformly illuminated field to determine irregularities in the response of the target and transmission of the filters across the field. After the dark field is subtracted from the flat field, the flat field is divided into the data frame. At this point, the data is converted from integer to floating point format.

Typically, several images are taken of the same object through different filters, producing images of different colors. The images can then be ratioed to bring out color features. The images can then be formatted and written onto magnetic tape for the scan converter which produces photographic hard copy. Further processing depends on the particular data at hand.

```
 __________          __________          __________
|  DATA    |        |  DARK    |        |  FLAT    |
|__FRAME___|        |__FIELD___|        |__FIELD___|
      *                 *   *                 *
       *               *     *               *
        *             *       *             *
         *SUBTRACT*            *SUBTRACT*
          *      *              *      *
           *    *                *    *
            *  *                  *  *
       _____**_____          _____**_____
      | DATA FRAME |        | FLAT FIELD |
      |     -      |        |     -      |
      |_DARK_FIELD_|        |_DARK_FIELD_|
                *            *
                 *          *
                  *        *
                   * DIVIDE*
                    *     *
                     *   *
                      * *
                 ______*______
                | CORRECTED   |
                |___DATA______|
```

FIGURE 1

FLOWCHART OF DISTORTION CALIBRATIONS

2.2 EXISTING SOFTWARE

An image processing system has been written by Paul Kinnucan at MITPAL which performs the processing described above. It runs as a batch job at LNS. The system is a "black box", usable by nonprogrammers. It reads control statements from punched card input and performs the indicated functions. Disk maintenence is done automaticallly. The batch system, however, is lacking is some areas. One example is computing image ratios. The images to be divided must be taken at different times. The object, therefore, will not appear exactly in the same relative position on the images. A partial shift of one image relative to the other must be done in order to perform the ratio correctly. Finding the correct overlay often takes several trials. On the batch system, one trial would consist of running a job, producing a tape for the film converter, and producing a photograph to see the results. This is typically a 1/2 day to 1 day delay. If the results could be seen immediately, then the user's efficiency would be greatly increased. This long turn around time is the primary motivation for the interactive system.

The interactive system is logically and physically an extension of the batch system. The data reduction done is similar to that of the batch system. In addition, results can be displayed on the graphics terminal. The data formats of the two systems are identical to allow the processing of the same data under both systems. The initial work done on the batch system was carried over into the development of the interactive system. This includes the disk maintanence routines and the logic of the image processing routines. Several interesting new problems arise in the implementation of the interactive system, however. A new I/O device, the graphics terminal, must be controlled. Communication between the user and the system is now done in real time. The interactive environment itself, allowing the user full control of the system at all times, must be implemented. These problems have been dealt with successfully by the author. The interactive system is presently in use, although continual modifications and improvements are being done. The remainder of this thesis describes the major parts of the interactive system and their operation.

CHAPTER 3

PRINCETON 801 TERMINAL

3.1 HARDWARE FEATURES

The Princeton 801 terminal(*) is a state of the art gray scale terminal featuring a Lithicon storage tube. Data is written directly onto the storage tube as the display is being refreshed. A separate set of hardware registers maintains the cursor position. The cursor is not stored but is generated directly onto the display. Internal logic generates characters in three sizes, absolute and relative vectors, and gray scale. The selective erase mode allows erasing of any portion of the screen without affecting other areas.

The internal character set of the Princeton 801 is ASCII. In text mode, all printable ASCII characters are displayed. The interpretation of the characters in other modes depends on the mode. In the vector modes, the characters are interpreted as coordinate positions or offsets. In gray scale, characters correspond to intensity levels. If a character is prefixed with the control character, it is interpreted as a mode switching command.

The Princeton terminal is connected to the selector channel via an IBM 2701 Data Adapter. The parallel interface

----

(*) Princeton Electronics Products, Inc., North Brunswick, New Jersey.

allows data transmission rates limited only by the rate the terminal can accept the data. This ranges from 3000 bytes/second for text to 50,000 bytes/second for gray scale data.

3.2 SOFTWARE SUPPORT

A set of basic level I/O routines had to be written for the terminal because there is no IBM support for it. These routines provide read and write capabilities, translation of character codes, and error recovery. Control blocks for the I/O Supervisor are created and channel programs are constructed. The routines are written in the assembly language but are callable by both FORTRAN and PL/1.

The write routine writes a variable length block of data to the terminal. No checking is done on the data. The routine is called with two parameters, the address of the data to be written and the length. The PL/1 entry point allows the use of varying length character strings. Actually, the Dope Vector for the character string is passed as the argument. The length of the string is contained in the Dope Vector and thus only one parameter need be passed. The I/O Supervisor is called to initiate the write operation and its completion is awaited. Control is then returned to the calling program.

The read routine reads a line of input from the terminal keyboard. The length of the line read is returned to the calling program and the data is moved to an area specified by the calling program. The PL/1 entry point also uses varying length character strings. The input string is monitored for three special characters. The erase character key erases the last character and backs up the cursor one position. It sends a back space character to the computer. The back space is checked for and the read buffer pointer is backed up one position to effectively erase the character in the buffer. The delete key sends the delete character to the computer. When it is read, the buffer is emptied and the control code for the line erase function is sent to the terminal. This erases the whole line on the terminal. The carriage return is interpreted as the end of line delimiter. When it is read, the read routine exits. Presently, characters are read one at a time with a .1 second delay in between. This software multiplexing is used because there is no multiplexor channel on the computer. A buffer is being built that will collect the input from the terminal and send a whole line at a time to the computer. When this is completed, the read routine will simply read the whole line before it checks for the special characters.

A third routine reads the current cursor position from the terminal. The routine sends the read cursor control code to the terminal, causing the terminal to send the encoded cursor position to the computer. The routine then converts the encoded position to X-Y coordinates and returns them to the calling program.

The translate routine will translate from EBCDIC to

ASCII, and vice versa, with an optional CAPS only in the ASCII to EBCDIC direction. It accepts from the calling program three parameters, the direction of translation, the length of the data to be translated, and the address of the data to be translated. The translated data replaces the input data. Like the read and write routines, the PL/1 entry point allows the use of varying length character strings.

In addition to the above routines, there is a routine that performs an OPEN and CLOSE on the terminal data set. This is a preliminary operation required by the I/O Supervisor to construct control blocks and allocate the device. Also, there is a supervisor call which initializes the interrupt interface described later.

The I/O Supervisor requires that every I/O device have an associated error routine in the SVC library. This routine is loaded into the transient area whenever the termination of an I/O event indicates that an error has occurred. Error recovery for the Princeton 801 is limited to the detection of the timeout condition during read. All other errors must be caught by the terminal user and corrected manually. This is because there is no way to validate data read from or written to the terminal. Any gross errors will be detected immediately and subtle errors, though they go unnoticed, are probably unimportant. Experience has been that errors are infrequent.

CHAPTER 4

COMMAND LANGUAGE

The command language used for the interactive system is rather simple. The basic syntax is: <command> <parameter_list> where <command> is a legal command name and <parameter_list> is zero or more parameters separated by blanks. The syntax of the parameter list varies from command to command and can assume almost any form. Generally, a parameter is either an image name, a decimal constant, or a keyword parameter. An image name is eight characters long, padded on the right with blanks if necessary. The first character must be alphabetic (A-Z) and the other seven characters can be almost any other printable character. Some characters are reserved for delimiters and cannot be used. A keyword parameter has the form "keyword" = "value". "keyword" is a predetermined character string that must be entered exactly. "value" is a decimal constant. Parameters may be recognized in two ways, either by their positions in the parameter list (positional) or by a keyword. Many commands require an image name as one parameter and it must be first in the list. Other parameters may be matrix coordinates and scaling factors, though each command has its own particular requirements.

4.1 COMMAND LANGUAGE PROCESSOR

The command language processor converts a command and its parameter list to an internal format to be used by the rest of the system. A command is converted to its index in the command table, a binary integer. The parameter list varies from command to command and has no general format. Each command has a separate routine to process its parameter list. The language processor is called from the system's main control routine each time it is ready to receive a new command. The language processor reads a command from the terminal, processes it, and returns to the main control routine the command index and a pointer to the parameter list constructed. Actually, the main control routine passes to the language processor the address of an area in storage where the parameter list is to be constructed.

The command processor operates directly on the input string. The only preprocessing done is the deletion of nonessential blanks. The input string is kept in a common area available to all routines comprising the language processor. As each parameter is processed, it is deleted from the string and the string is recompressed. Additional input from prompting is inserted at the beginning of the string.

4.2 PROCESSOR ROUTINES

There are three logical groupings of the command processor routines. The first group contains the control routines. These routines check for a legal command and call the appropriate parameter processing routines. The second group is the parameter processors. They are responsible for creating a parameter list in the format suitable for the particular command. Finally, there is a group of routines that perform common functions. They include I/O routines and string handling functions.

The command processing routines, as do the rest of the system routines, follow naming conventions. All routines in the command processor begin with the letter "Z". The control routines and the common function routines have a mnemonic following the "Z". The parameter processing routines have as the rest of their name the name of the command they process or an abbreviation of that name.

4.2.1 CONTROL ROUTINES

The control routine for the command processor (ZCMDPRC) receives initial control. It initializes the command string (ZSTRING) by calling the I/O read routine (ZGET) to read a command from the terminal. Next, the string is compressed by a call to ZAP, the compress routine. If the length of the string is now zero, there is no input from the terminal. Control is returned to the main control task with a zero command index, indicating that there is no command.

ZSTRING is now checked for a valid command. A "word" of eight characters or less is extracted from the beginning of the string. The first blank in the string indicates the end of the word. The word is now compared to the entries in the command table until a match is found or the table is exhausted. If no match is found, an error message is sent to the terminal and the command processor returns with a zero command index.

If control reaches this point, a valid command has been found. The word is deleted from the beginning of the string and the string is recompressed. The director routine (ZXCTL) is now called with the command index and the pointer to the parameter area. ZXCTL has a table of entry points, one for each parameter processor. This table is indexed by the command index and the proper parameter processor is selected. Control is passed to the parameter processor and initial processing is completed.

4.2.2 PARAMETER PROCESSORS

The parameter processors (hereafter called Zprocs) are as varied as the parameter lists they generate. However, there are several general formats which represent 90% of the presently implemented commands. Four formats will be discussed here, as most commands fall into one of the four categories. The commands not represented, though they generate different parameter lists, operate similarly to the ones discussed. Thus, most of the logic of parameter processing will be presented.

The first group of commands is trivial in that they take no parameters. These commands include LOGOFF, LISTD, and PEND. The commands require no processing and the routines simply make an immediate return. They are included only to make the processing uniform. ZXCTL does not necessarily know which commands do not need parameter processing. Also, if these command are extended so that they need parameters, there are provisions for including processing.

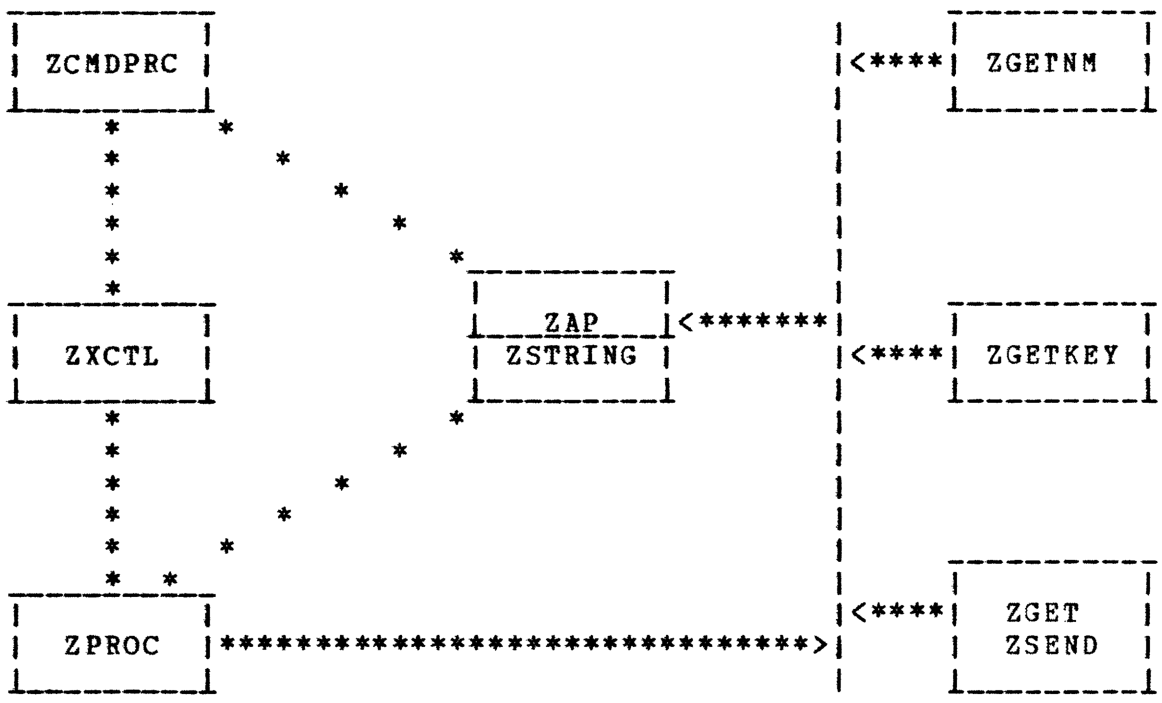

FIGURE 2

RELATIONSHIPS BETWEEN COMMAND PROCESSOR ROUTINES

The next level of parameter processing is the commands that require a single parameter, an image name. This group includes PROCESS and SAVE. This name may describe an image on the permanent file or may be a special "phantom" name that indicates the active image. The active image is described in Chapter 5. There is a common routine (ZGETNM) that searches the input string for an image name. ZGETNM performs all the error checking and prompting necessary to get a valid image name. The Zproc calls ZGETNM to get the image name. It is now finished and returns control to ZCMDPRC.

The third level of commands include those that require an image name along with several other parameters. These commands include DISPLAY, SUBTRACT, DIVIDE, SCALE, FLAT, TTOD, ADDC, SUBC, MULTC, DIVC, and STAT. The Zprocs use ZGETNM to get the image name as above. In addition, they must get the other parameters and supply default values for omitted parameters. As mentioned above, parameters may be positional or keyword. To avoid ambiguity, all positional parameters must appear before the keyword parameters. When the first keyword parameter is encountered, a flag is set to indicate that any additional positional parameters encountered are out of place. The user is prompted to enter a keyword parameter to replace each out of place positional parameter. A positional parameter is recognized by the first character being numeric (0-9 and +-.). It is converted to the proper data type and inserted into the parameter list being constructed according to its position (first, second, third, etc). Keyword parameters are recognized by the first character being alphabetic. There is another common routine, ZGETKEY, that separates the keyword form the string. The keyword is checked against a table of valid keywords for that command. If a match is found, the value associated with the keyword is converted and placed into the parameter list according to the keyword. If the keyword is not found, it is in error and the user is prompted to correct it. After all specified parameters are processed, any that are omitted are given default values. The Zprocs now return to the control routine.

The last category of Zprocs construct variable length lists of image names. The list contains the number of names as the first element. The commands that fit this category are DELETE and AVERAGE. The input string is processed identically for each name until the string is empty. ZGETNM is used. After all names are processed, control is returned to ZCMDPRC. This varying length parameter list can be extended to include lists whose elements are more than a single parameter, i.e., a variable length list of parameter lists. The restriction here is that each member of the list be identical. This in fact is done for CONTOUR.

4.2.3 COMMON FUNCTIONS

ZAP, the most commonly used function routine, compresses ZSTRING and, optionally, deletes the first N characters from the string, where N is a calling parameter. ZAP first converts the first N characters to blanks. It then proceeds to delete all nonessential blanks from the string. A blank is considered nonessential if it is first in the string, next to a blank, to the right of a left parentheses, or left of a right parentheses. The only exception to this rule is blanks inside of double "quotes". These blanks are always kept. Each character is checked from left to right and flags are set according to what character is found. When a blank is found to be nonessential, it is deleted by moving the remains of the string left one position.

The command processor has two routines which perform I/O to the terminal. They are interface routines to lower level I/O routines. In addition, they perform processing particular to the command processor. ZGET is used to get input from the terminal. It calls PGET to get input from the terminal. The input is inserted directly into the beginning of ZSTRING. ZPUT sends a message to the terminal. Presently, it is called with a message to be sent and links directly to PSEND, a routine which controls the text area of the terminal.

The use of ZGETNM has been described under the Zprocs. When it is entered, it expects to find an image name as the first element of the string. A parameter is passed to ZGETNM which indicates whether or not the image name is required. If the name is required and not found, the user is prompted to enter it. If the name is optional and not found, the "phantom" name is returned. ZGETNM proceeds as follows. If the first character in the string is numeric, there is no image name present and the proper action (above) is taken. Next, the position of the first blank is found and a "word" is extracted from the string. If this word contains an equals sign, it is a keyword parameter. In this case also, there is no image name and the appropriate action is taken (above). Otherwise, the word is considered to be an image name and is checked for validity. It must begin with an alphabetic character and contain only legal characters. If not, the user is prompted to correct it. If an image name is to contain blanks or other illegal characters, it must be enclosed in double "quotes". If the "quotes" are present, they are removed and the name is considered legal.

ZGETKEY is used to extract a keyword parameter from ZSTRING. When it is entered, the string should consist only of positional and keyword parameters. If the first character in ZSTRING is numeric, the first element is a positional parameter. In this case, ZGETKEY returns the null string as the keyword and the calling Zproc must handle the positional parameter. Otherwise, the parameter is considered a keyword parameter. The keyword is extracted form the string and returned to the calling Zproc. Also, it is deleted from its associated string and the value is moved to the first position. The Zproc may now convert the value according to the keyword returned.

In addition to the above common functions, there is an implicit function present in the form of a PL/1 ON UNIT.

This ON UNIT is entered whenever a conversion error is raised during conversion from the character representation of a decimal to the internal format. The ON UNIT sends an error message to the terminal and prompts the user to correct the illegal number. The input from the user replaces the illegal number and the conversion is retried.

CHAPTER 5

AVAILABLE COMMANDS

There are four general categories of commands available on the interactive system. The first category is the commands that perform functions on images. These include SCALE, DIVIDE, SUBTRACT, FLAT, ADDC, SUBC, MULTC, and DIVC. Next are informatory commands. These commands supply information on images and the permanent file. Included are LISTD, DISPLAY, and STAT. The third group of commands control the contents of the disk files. These are PROCESS, PEND, SAVE, DELETE, and TTOD. Finally, the LOGOFF and HELP commands are used to control the terminal session. Not all commands will be described in detail. However, the general operation of each command will be presented in some form.

5.1 ACTIVE IMAGE FUNCTIONS

The commands that perform functions on images all operate on the "active" image. The active image is stored on a separate temporary file capable of storing one standard sized image. An image is made active by entering a PROCESS command specifying the image by name. This command copies the given image from the permanent file to the temporary file and sets the active bit in the Active Image Control Block (CAIB). After an image is made active, the functions must use it as principle input. The functions that need two images for input read the secondary input directly from the permanent file, but the results are placed back into the active image. In a sense, the active image corresponds to the accumulator register on a computer and the permanent file is its memory. The accumulator must first be loaded from a storage location (PROCESS). Next, another storage location may be subtracted from the accumulator with the results replacing the contents of the accumulator (SUBTRACT). Finally, the contents of the accumulator may be stored into any memory location. The active image may also be stored with the SAVE command.

```
 ____________                    ____________
|            |                  |            |
|  ACTIVE    |     PROCESS      | PERMANENT  |
|            |<**************|             |
|   IMAGE    |                  |   IMAGE    |
|            |**************>|             |
|    FILE    |      SAVE        |    FILE    |
|____________|                  |____________|
         *                          *
         *                          *
         *                          *
     ____*__________________________*__
    |                                  |
    | ACTIVE IMAGE COMMANDS            |
    |__________________________________|
```

FIGURE 3

RELATIONSHIPS BETWEEN ACTIVE IMAGE,

PERMANENT FILE, AND ACTIVE IMAGE COMMANDS

5.2 DISK MAINTENENCE ROUTINES

There are three commands used to control the contents of the active image. These are PROCESS, SAVE, and PEND. The PROCESS command was described above. The SAVE command may be used to copy the active image back onto the permanent file. The SAVE command may be entered with or without an argument. If entered without an argument, the active image is stored under its original name, thus replacing the original image. Optionally, the SAVE command can be entered with a new name to be assigned to the image. If the new name already exists in the directory, the contents of that image are replaced by the active image. Otherwise, the name is entered into the directory, space is allocated for it, and the new image is created. The PEND command is used to return the active image to the inactive status. It will prompt the user to save the image if it has been altered. Otherwise, it sets the inactive bit in the CAIB and exits. In addition, the DELETE command may be used to delete any image from the image file and the TTOD command will restore an image from a library tape to the permanent file.

The LISTD command provides the user with an edited listing of the contents of the permanent file. Each image on the file is listed by name along with several items of information about the image. The amount of free space on the disk is listed at the end.

5.3 OTHER COMMANDS

The DISPLAY and STAT commands may operate either on the active image or on any image on the permanent file. These functions are "read only" in that they do not modify the image. The phantom image name (mentioned in Chapter 4) is used to distinguish between the active image and all other images. If the phantom image name is passed to the routine, it will use the active image for its input. Otherwise, it will attempt to use an image on the permanent file for its input. The DISPLAY command is used to display an image on the terminal in gray scale. Display modes are discussed more in the appendix. The STAT command provides the user with pertanent statistics on any submatrix of any image.

To control the terminal session, the user has two commands available. The LOGOFF command is used to end the terminal session. The HELP command will provide the user with information on how to use the system and enter commands.

5.4 JUSTIFICATION FOR THE ACTIVE IMAGE

The active image mode of operation was motivated by several considerations. The major consideration was in the reduction of I/O wait time. If two images to be combined are physically distant on the disk, the read/write heads have to jump from image1 to image2 and back for each row of the image. However, if the images are on distinct disk drives, the heads can remain relatively stationary. The cost of copying the image to the active image file and back must be considered, though.

An average disk file will be 60% to 80% full during a processing session. However, if images are continuously being added and deleted, any particular image has equal chance of occupying any slot on the disk. Therefore, we can assume that, on the average, any two images will be spaced 1/2 disk pack away. Average seek time for an IBM 2311 disk drive is 75 ms and the average rotational delay is 12.5 ms. To combine two images on the same pack would require: a) 12.5 ms rotational delay to access row1 of image1, b) 75 ms seek time to image2, c) 12.5 ms rotational delay to access row1 of image2, d) 75 ms seek time back to image1, and e) 12.5 ms rotational delay to replace row1 with the combined rows, for a total of 187.5 ms delay for each row. If the images are on separate packs, the times can be divided into PROCESS time, combination time, and SAVE time. To copy the image from the permanent file to the active image file requires 12.5 ms rotational delay on each drive for each row copied, or 25 ms per row. The SAVE time is identical. The combining requires 37.5 ms delay time per row because each row has to be read (25 ms) and the result must be written (12.5 ms). This gives a total of 87.5 ms delay time, or a savings of 57%. This is a minimum savings. The usual practice is to perform several functions on the active image before saving it. This is an additional 150 ms savings per row for each function performed. The actual real time savings is more complicated to compute. The read time, computation time, and overhead were ignored in the above analysis. However, a simple estimate is to multiply the savings per row by the number of rows (250) to give a savings of 25 seconds. In actual practice, it takes approximately 15 seconds to PROCESS or SAVE an image and 30 seconds to combine two images on the interactive system for a total of 60 seconds. To combine two images on the batch system takes approximately 90 seconds, so the estimate is reasonable.

The second consideration in the active image philosophy is maintaining consistency in the permanent file. It is undesirable to have an inconsistent image in the file. An image is considered inconsistent when a function has been performed on a fraction (< 1) of it. Since the user may interrupt the system at any time, this could create many instances of inconsistent images. However, by making all changes to the active image, most inconsistentcies can be confined to it. Recovery is simple and involves only the restoration of the active image from the permanent file. The SAVE function, if interrupted, can still leave an inconsistent image on the image file, but at least the user will be aware of the possibility in this instance. The system cannot be completely "idiot proof".

CHAPTER 6

SYSTEM STRUCTURE

The interactive system may be thought of as a collection of processes and a control routine which synchronizes their operation. There is a logical process associated with each command implemented and one for the language processor. Although only one logical process may be active at any time, several tasks may be active in the implementation of that process.

There is one routine per command that actually performs the function of that command. These routines are the lowest level routines in this discussion. They follow the naming convention "A" followed by the name of the command they implement, and will be called "Aprocs". Most of the logic for the individual Aprocs was developed on the batch system and will not be discussed here. One level higher than the Aprocs are the Iprocs. They serve as interfaces between the Aprocs and the control structure. They will be presented as they are quite important in the implementation of the interactive environment. At the highest level are the control routines and control blocks used in maintaining the system status. The control routines begin with "C" and end with a mnemonic describing their function. The control blocks begin with "C", end with "B", and have a mnemonic in the middle. The logic of the interactive system will be developed from the ground up, beginning with the Iprocs.

The implementation of the "break" function will be discribed. The control routine will be presented along with the user/system interface. Finally, a portion of a typical session will be stepped through showing where the various routines interact.

6.1 IPROCS

The Aprocs are designed to perform a particular function without any knowledge of the interactive environment. The environment is transparent to the Aprocs. Under normal circumstances, an Aproc would be entered with a parameter list, it would perform its function, and then exit. However, operation under an interactive environment is not quite so simple. First, there is the possibility of obtaining parameters dynamically. One example is inputting coordinates of a point in an image. If the coordinates are known, the user may enter them with the command. The alternative is to position the cursor to the desired point on an image on the terminal display and have the system read the coordinates from the terminal. In the latter case, the coordinates are not known until the command is being executed. The second deviation from normal operation procedures is the "break" function. A break is when the user wishes to terminate the currently active command and regain control of the system. Some means of terminating an active Aproc is necessary. The Iprocs were developed to aid in the implementation of the break. In addition, some checking is done on the legality of the parameters passed to the Aproc and on the status of the system. Although dynamic parameter processing is not implemented to a great extent at the present, this function is a natural addition to the Iprocs because of their structure and logical placement in the control structure.

To implement a break, subtasking is used. Each breakable function is attached as a subtask of the control routine. When a break is indicated by the user, the subtask is detached. This causes the operating system to abnormally terminate the subtask and then release all resources allocated to it. The necessity for subtasking will become apparent later. In PL/1 there is no explicit way to detach an active subtask. However, there is an implicit way. A return from a PL/1 proceedure, unlike most languages, is not really a return but a call to a PL/1 library routine. This routine frees all resources allocated to the proceedure. In addition, it detaches all tasks attached in the proceedure. The Iprocs are such proceedures. The main function of an Iproc is to attach its corresponding Aproc and then wait for either its normal completion or the occurrance of a break. When either one of these events is satisfied, the Iproc returns. If the Aproc has terminated normally, the return is much like a normal return. The process is complete and its resources are released. If the break occurs before the Aproc has completed, it is detached and thus abnormally terminated. In either case, control returns to the terminal and the user may enter a new command. All functions that can be initiated by a command are breakable. There is one Iproc and one Aproc for each command. In addition, the command language processor is breakable. It has no Iproc but is attached by the control routine directly. However, since it does "know" something about the interactive environment, the need for an Iproc is not as great.

```
-----------                            ---------   -----
|         |                        |   |       |   |   |
| CONTROL |***********************>|   | CPIC  |<****| T |
|_________|                        |   |_______|   | E |
     *                             |       *       | R |
     *                             |   ---------   | M |
     *                             |   |       |   | I |
-----------                        |****>| PGET  |<****| N |
|         |                        |   |_______|   | A |
|  IPROC  |***********************>|   ---------   | L |
|_________|                        |   |       |   |   |
     *                             |****>| PSEND |****>|   |
     *                             |   |_______|   |___|
     *                             |
-----------                        |   -----------
|         |                        |   |         |
|  APROC  |***********************>|   |         |
|_________|                        |****>| CONTROL |
     *                             |   | BLOCKS  |
     *                             |   |_________|
     *                             |
--------------------               |
|                  |               |
| UTILITY ROUTINES |               |
|       AND        |**************>|
| DISK MAINTENENCE |               |
|__________________|               |
```

FIGURE 4

CONTROL STRUCTURE

6.2 IMPLEMENTATION OF BREAK

When a break is intended by the terimnal user, the Iprocs detach their respective Aprocs to terminate them (above). However, the reception of a break by the system is a different story. The attention interrupt facility of the terminal is used to initiate communication between the user and the system. However, this communication may be either for the purpose of entering a command or for break. Its interpretation depends on the current state of the system.

The path of the attention interrupt from the terminal through the operation system to the interactive system is discussed fully in the appendix. For our purposes here, we can assume that the operating system signals the interactive system when the interrupt occurs. This is done by posting the completion of an event associated with the terminal interrupt. A task must be awaiting the completion of this event in order for it to be meaningful. Thus, the use of subtasks becomes necessary. While the interactive system is processing a command, there must still be a task waiting for an interrupt from the terminal. Otherwise, the user would not be able to issue a break.

The interactive system has a rouinte (CPIC) whose sole purpose is to monitor the terminal for an interrupt. When initially attached, it sets up an event control block (ECB) to be used by the operating system to post the occurrance of the interrupt. It then waits for the interrupt to occur. When the completion of the ECB is posted, CPIC "wakes up" and handles it as described below. It then goes back into wait until the next interrupt occurs.

As the interactive system is now set up, commands may only be entered when the system is idle, i.e., not processing a command. An interrupt issued while the system is active is interpreted as a break. (*) There is a control block, CPICB, used to maintain the status of the system (active or idle). When CPIC wakes up, it checks the status of the system. If the system is idle, CPIC posts the completion of the WAKE_UP ECB in CPICB. This is to signal whatever task that is waiting for terminal input (usually the language processor) that the user is ready and to continue processing. If the system is active, the BREAK ECB is posted. Usually, an Iproc is waiting for the break. It then detaches its Aproc to terminate the current command. The active/idle flag in CPICB is set to "idle" by any routine expecting input from the terminal. Before control is returned to the waiting task, CPIC set the flag to "active". Thus, the status of the system is always available.

(*) It might be noted here that this is an unnecessary restriction. The system was initially developed this way because it is less complex. However, an extension to overlapping processing with command entering may be accomplished by creating another task for the language processor so that commands may be entered while the system is active. However, this raises questions of contention for the terminal and makes the interpretation of a break more complex.

6.3 CONTROL ROUTINE

The control routine (CONTROL) for the interactive system is very simple. Basically, CONTROL executes in a loop, accepting a command from the user and calling the appropriate processor to execute it. The language processor is attached as a task and CONTROL waits for it to return with a processed command. Next, the command index and the address of the parameter list are passed to CXCTL, the director routine for the control task. CXCTL transfers control the the appropriate Iproc to initiate command execution. CONTROL waits for the command to terminate, either normally or abnormally, and then prepares for the next command.

6.4 OTHER ROUTINES

There are two other routines used by the system to control the terminal. These are PSEND and PGET. Although they are I/O routines, they are presented here because of their interface with the control structure. PGET in particular is related to the above discussion. Its main function is to read one of line of input from the terminal. However, it first waits for the interrupt from the user. Upon entry, PGET sets the "idle" flag in CPICB and then waits for the WAKE_UP ECB to be posted. When PGET wakes up, the user is ready to input data and the read is initiated. PGET may be called from any task, and that task is suspended until the user is ready. In this way, the other routines in the system needing input from the terminal do not have to set up the interrupt protocal. PSEND does not fall into the above interrupt processing discussion, but it does have a control function. The bottom third of the terminal display is reserved for text communication between the user and the interactive system. This is a limited space and must be used conservatively. The cursor is rotated from top to bottom as each new line is entered. The last 12 lines are always visible. PSEND maintains a control block, CTEXTB, that contains information on the size of the text area, the current line position, and the coordinates of the lines. At any point, PSEND may be called to output a line to the terminal. The cursor is positioned to the proper line and the line is sent. The next line position is erased so that the user can see which is the current line. This frees the other routines from worrying about where on the terminal the line is to be written.

6.5 EXAMPLE

At this point, an example is in order. A portion of a typical terminal session will be presented. We will start with the initial job set up and then process an image. The interactings of the various portions of the system will be shown. This is by no means a complete description of the total capabilities of the system. Rather, it is intended to demonstrate each major portion of the system and how it works.

The first thing that must be done before running the interactive system is the computer operator's setup. The disk pack must be mounted on a drive and the terminal powered on. In this example, we will use the disk named PAL3. The operator mounts the disk on the drive and enters a command to the operating system to recongnize the mounting. The terminal is now powered on (it is normally off) and the job for the interactive system is read in through the card reader. The job contains mostly JCL describing the program and the data sets used. The load module for the interactive system is on a library on the permanent storage for the computer system.

When the job starts running, CONTROL receives control. It must first perform some initialization for the interactive system. First, it initializes the active image file. The active image file is a temporary file that is created each time the job is run and is deleted at the end of the job. Next, the image file on PAL3 must be opened. The directory for the file is read into core where it resides during processing. Finally, CPIC is attached as a task to monitor the terminal. The system is now ready to receive commands from the user.

CONTROL now attaches the command language processor as a task. ZCMDPRC sends a message to the terminal to indicate to the user that it is now ready. The message usually is "OK". ZCMDPRC calls ZGET which calls PGET to read a line of input from the terminal. PGET sets the "idle" flag in CPICB and waits for WAKE_UP to be posted. The system is now in a wait state. When the user is ready to enter a command, he presses the BREAK key on the terminal. This key sends the attention interrupt to the system. CPIC receives control from the operating system by means of posting the ECB set up by CPIC. CPIC must now determine what action is to be taken. It checks the active/idle flag in CPICB and discovers that the system is "idle". It posts the WAKE_UP ECB and then returns to the wait state for the next interrupt. When WAKE_UP is posted, control is returned to PGET which reads input from the terminal. PGET returns to ZGET which places the input into the beginning of ZSTRING. Since ZSTRING is initially empty, the input is the only data in the string. ZCMDPRC now checks the string for a legal command. It finds that the user has entered the command "LISTD". The command index is set to 4 because LISTD is the fourth command in the command table. The Zproc for LISTD is called (ZLISTD) to process the parameter list. LISTD takes no parameters so ZLISTD returns to ZCMDPRC. The command processor is now complete and returns to CONTROL.

CONTROL has been waiting patiently for the completion of either ZCMDPRC or the break ECB. Since the user did not interrupt the system, ZCMDPRC returns with the processed command. CONTROL checks the command index (4) and finds that a legal command has been found. CXCTL is called which in turn calls ILISTD, the Iproc for the LISTD command. ILISTD attaches ALISTD and waits for it to complete. ALISTD receives control and says, "Aha, the user wants to see what images are in the directory." It then proceeds to list the contents of the directory on the terminal display. When it has finished, it returns to ILISTD, which cleans up and returns to CONTROL. The system is now ready for the next command.

The user sees that he has four images on the disk, named DARK, FLAT, MOONRED, and MOONBLUE. These images represent the dark field, the flat field, and two data frames taken of the moon in the red and blue regions, respectively. The user wants to calibrate the two data frames and compute their ratio. The first thing he does is to subtract the dark frame from the flat field. He does this by the following sequence of commands.

```
PROCESS FLAT
SUBTRACT DARK
SAVE FLATD
PEND
```

Thus, a new image is created, called FLATD, which is the result of subtracting the dark field from the flat field. Each of the commands entered above is handled similarly to the LISTD command above.

Next, the user must subtract the dark field from each of the data frames and then remove the flat field. In this example, we will assume that he first attacks MOONRED. First, he enters a PROCESS MOONRED command to make the image active. Next, he will enter SUBTRACT DARK. This will create an image with the dark field correction. Now, for pedigogical reasons, let's assume that he wants to later look at the dark field corrected image. He might enter SAVE REDDARK to save the image with a new name. Now, while the dark field corrected image is still active, he would want to remove the flat field. He might enter FLAT FLAT, which is the command to flat field correct with image FLAT. However, after he enters the command, he discovers that he wanted to use image FLATD instead of FLAT. He now enters break to interrupt the system.

In general, the language processor runs too fast for the user to interrupt it before it completes processing. We can therefore assume that the FLAT FLAT command has actually started executing when the user realizes his mistake. When the interrupt is issued by the user, AFLAT is removing the flat field FLAT from the active image while IFLAT is waiting for it to complete. CPIC receives control when the interrupt is issued. It checks CPICB and realizes that the system is "active". Therefore, it posts BREAK, the ECB to indicate that the user wants to interrupt processing. IFLAT is waiting for either BREAK or AFLAT to complete. When BREAK is posted, IFLAT receives control and returns. This causes AFLAT to be abnormally terminated and control is returned to CONTROL. ZCMDPRC is called to prepare the next command from the user. Since part of the active image has been changed by the erroneous FLAT command, it must be recreated. The user enters the PEND command to release the active image without saving it. Since he has saved the dark field corrected image, he does not have to recreate it. He now enters PROCESS REDDARK and then the correct FLAT FLATD commands. When the correct flat field has been removed, he saves the active image with the SAVE REDDF command to create a new image with the dark and flat fields removed. With the following sequence of commands, he can correct the MOONBLUE image for distortions.

```
PROCESS MOONBLUE
SUBTRACT DARK
SAVE BLUEDARK
FLAT FLATD
SAVE BLUEDF
```

At this point, the user has corrected his images for distortions and saved them under new names. If these are the only two images he wishes to work on, he might discard the DARK and FLAT images. He can enter DELETE (DARK FLAT) to delete the two images from the file and release their space. Now, he may look at the images he has created and decide what to do with them next. The DISPLAY command is entered specifying the names of the images to be displayed and the images are drawn on the terminal display. The user looks at the images and decides that the blue image is shifted approximately 14 points in the X direction and 37 points in the Y direction with respect to the red image. To compute the ratio of the two images, he must enter the DIVIDE command with the overlay parameters. Since the BLUEDF image is still active (the PEND command was not entered), he does not have to enter a PROCESS command for it. He might enter DIVIDE REDDF 14 37 to ratio the two images. Next, a DISPLAY command is entered to display the active image, the new ratio. The command language processor will pass the "phantom" image name to ADISP because no name was entered with the command. ADISP realizes that it must display the active image and not look for the image on the image file. When the ratio image is displayed, the user finds, much to his surprise, that he has correctly guessed the overlay on the first try. He saves the active image with a SAVE RED/BLUE command to create an image named "RED/BLUE". He enters a PEND command to delete the active image and decides that such success must be shared with the other members of MITPAL. The LOGOFF command is entered to terminate the terminal session. The terminal is powered off, the disk is dismounted, and the example session is complete.

CHAPTER 7

PERFORMANCE AND IMPROVEMENTS

The interactive system as described here has been in operation since September, 1974. It is being used for the reduction of images taken of Mars and the Moon. Most of the current work is in the area of color ratios, as experience with the batch system has made this the easiest to implement. Future developments should extend the capabilities of the system to include Fourier filtering and creation of mosaics.

There are several needed improvements to the interactive system. First, the scheduling of its operation is a problem. The interactive system uses a full partition at the LNS computer. Since there are only two partitions to begin with, this is a heavy drain on available resources. However, if the system could be made smaller with respect to core usage, only a small partition would be necessary, and there would be enough core left over to run two regular partitions in addition to the interactive system. This would mean less impact on normal operations and more available time to run interactively.

Another needed improvement for the system is response time. Processing times are on the order of 15 - 40 seconds to perform image functions. Although the system is limited by the speed of the disk drives for I/O time, there are several areas where processing can be overlapped to improve response. For example, with double buffering, an operation could be performed on one row of an image while the next row is being read in. Also, several functions could be performed simultaneously, such as dark field and flat field corrections. One row of an image could be read in, the dark field subtracted, and then the flat field divided before the results are written out. This means that the active image is read and written only once instead of twice to perform the two corrections, or a savings of 1/3 in I/O time.

The most needed improvement to the system is the display capabilities. The possible terminal displays have not even been explored as yet. Appendix B discusses several of the improvements planned.

Other than the improvements mentioned, the interactive system is working quite well. The basic frame work and command language are fairly well developed and fixed in structure. The improvements can be made within the present structure. The author hopes the system will be useful in MITPAL's image processing future.

APPENDICES

APPENDIX A

OPERATING SYSTEM MODIFICATIONS

Several modifications to the LNS operating system were necessary to allow the use of the Princeton terminal and the implementation of the interactive system. The addition of a Unit Control Block (UCB) for the terminal was necessary for the I/O Supervisor. An interrupt handling routine was added to direct the processing of attention interrupts. The ATTACH service routine and the GETMAIN service routine were modified to allow the use of the multitasking facilities of the PL/1 language on an MFT system. The addition of the UCB is normally done during system generation (SYSGEN) and is relatively uninteresting. The interrupt handler would not be necessary except in the absense of a multiplexor channel. The PL/1 fix could be helpful to users of other MFT systems who want to do multitasking.

A.1 UNIT CONTROL BLOCK

The UCB is an area of supervisor storage that contains information on a particular I/O device. There is one UCB for every device on the system. The I/O Supervisor uses the UCB's to schedule I/O operations, to handle I/O interrupts and errors, and to allocate devices. The UCB contains information on the present status of the device, both logical (online/offline, allocated, ...) and physical (ready, intervention required, not operational,...). It also has pointers to routines to do error handling and interruption processing. The UCB Lookup Table, another area of supervisor storage, contains pointers to the UCB's. The UCB Lookup Table is searched in a tree fashion, indexed by the unit address. The UCB's and the UCB Lookup Table are normally generated during SYSGEN. However, the one for the Princeton terminal was done in a novel way.

When the current system was generated, an extra UCB was added that was not being used. Since it was free, the author decided to steal it for the Princeton terminal rather than do another SYSGEN. The IBM service aid, IMASPZAP, was used to change the unit type and unit address fields in the UCB to reflect the Princeton terminal. Also, the UCB Lookup Table was rearranged to swap the unit addresses. There are several things to be said about changing the system without doing a SYSGEN. What would appear to be a small change might affect parts of the system least expected. The author was not sure the above fix would work, but is was tried and found to be successful. Later investigation discovered under what circumstances such a fix would work.

There are two major areas of the operating system where UCB's are used. They are device allocation and device handling. The job schedular must translate the unit field specified on a data definition statement to a particular device on the system. This is done in one of three ways. The most basic method is to specify absolute device addresses. This allows only the specified unit to be allocated. The second way is to specify an IBM device type, such as 2314 disk or 1403 printer, etc. This method allows any unit of this type to allocated. The most general method is to specify a generic name, such as SYSDA, TAPE, etc. Each method has its advantages and disadvantages, and are handled differently. When absolute address is specified, the allocation scheme is: does the address exist?, and, is it available?. The other two methods are similar and proceed as follows. The Device Name Table (DNT) is searched for the specified name. If the name is found, it will have an associated mask which, indirectly, specifies what devices may be allocated to that name. The mask is then checked against the unit type field in each UCB until a match is found that can be allocated. The method presupposes the existance to the DNT. This table is generated during SYSGEN. Once a device is allocated, it is handled completely through the UCB and unit address. For device handling, a UCB must exist and be pointed to by the UCB Lookup Table. For allocation, either the unit must be specified by address, or a corresponding name must exist in the DNT. Therefore, the above mentioned fix will work under the following circumstances. First, the UCB Lookup Table must be valid. The device can now be allocated by address and handled correctly. Secondly, if a similar device already exists, a name will exist in the DNT and the device can be allocated by name. Thirdly, the DNT can be expanded to include the new device name. All three of these methods assume that there is an existing UCB or room in the I/O Supervisor to insert one. If not, there is no alternative but to do a SYSGEN. The author used an existing UCB and absolute address allocation.

A.2 ATTENTION INTERRUPT HANDLER

The operator's console is handled in a special way. Input and output go through the Master Schedular which is responsible for operator/system communication. When the operator wishes to enter a command, he presses the REQUEST key on the console, which interrupts the system with the attention condition. The attention interrupt is an unsolicited interrupt. This means that it is not associated with any system initiated action. There is a field in the UCB which points to a routine to handle the attention interrupt on devices capable of initiating it. Most devices generate an attention interrupt when they go from the not ready to the ready state. The standard procedure for this action is to set the ready flag in the UCB and initiate any I/O operation awaiting this device. The console is different in the interpretation of the attention interrupt. This interrupt is not to signal that the device is ready, but to initiate communication with the Master Schedular. This requires a pointer to a different attention interrupt handler and the required routine to handle the interrupt. The routine for the console signals the Master Schedular, by means of the POST service routine, that the operator wants to communicate. The Master Schedular, which has been in the WAIT state, now "wakes up" and prepares to read the input from the console. The interactive system operates in much the same way. It waits until the user signals that he is ready to enter a command. The user presses the BREAK key on the terminal when he wishes to communicate. The terminal sends the attention interrupt to the computer. The I/O Supervisor must then pass control to the appropriate attention interrupt handler. This routine was written by the author and added to the I/O Supervisor. It has one main function, to signal to the interactive system that user wants to communicate. It does this in the same manner as the console routine; it posts the waiting task to continue processing. A Supervisor Call routine was also added to the SVC library to aid in this communication and is called by the interactive system when it begins operation. It sets up an Event Control Block (ECB) to be used for the WAIT/POST communication. In this manner, the interactive system does not have to continually monitor the terminal for input. In fact, this is not possible because there is no multiplexor channel.

<u>A.3 PL/1 MULTITASKING</u>

The third system modification involves the use of the multitasking facilities of the PL/1 language on an MFT system. In this context, multitasking means having more than one active task under the same job step as opposed to having concurrent job tasks. Some means of creating tasks within a job is necessary. This is done by means of the ATTACH service routine of the operating system. This is a standard feature of MVT systems and is optional with an MFT system. The PL/1 language allows the creation of subtasks within the rules of the language. When tasking is specified in a PL/1 procedure, a different version of the PL/1 storage supervision library routine is linked with the program. This routine handles all "automatic" storage and also the tasking features such as ATTACH, WAIT, and POST. The multitasking routine (IHETSAP) was written for an MVT environment, using features of the storage supervision not available under MFT. Since it was necessary to use subtasks in the interactive system it was highly desirable to use the facilities of a higher level language in doing so. The PL/1 library, or the operating system, or both, had to be modified to allow the use of this option.

IHETSAP assumes the MVT environment, and in particular, the use of subpools in storage management. Subpools allow the logical grouping of storage under a subpool number, both within a single task, and between tasks. Two advantages of the scheme are that storage gotten in small pieces can be freed all at once, and that storage can be gotten and freed on a task basis. IHETSAP uses the second feature, freeing core when a subtask terminates. Although the library routines keep track of most of the storage for each procedure, they expect the operating system to free the initial core gotten by first procedure within a task.

The MFT system with the subtasking option already has a facility for freeing core on a task basis, though only to a limited extent. This facility is limited to storage gotten by the supervisor to create the subtask. A Gotten Queue Element (GQE) is created to describe this storage. When the subtask is terminated, storage described by GQE's for the task is released by the supervisor. This processing was extended to cover PL/1 subtasks also. The storage supervision routine (GETMAIN) makes several checks to decide whether to create a GQE or not. These include: a) is this a subtask?, and b) is the supervisor calling?. By adding one more test, is PL/1 calling?, we will have simulated subpools for the PL/1 environment. The only problem lies in determining if PL/1 is calling or not. This was solved through the ATTACH routine by specifying another calling parameter. This parameter is to specify that PL/1 is doing the attach. The ATTACH routine then flags a field in the Task Control Block (TCB) for the task it creates. The TCB is available to all supervisor routines, and in particular, GETMAIN. It need only test this TCB flag to determine if PL/1 is calling.

Thus we require three modifications to allow the use of PL/1 multitasking. First, the library routine IHETSAP must specify to the ATTACH routine that PL/1 is calling. Second, the ATTACH routine must set the flag in the TCB. Finally, GETMAIN must check this flag and create GQE's for the task.

The first modification requires only a IMASPZAP fix. The second and third modifications require the addition of one and four instructions, respectively, and the reassembly of the ATTACH and GETMAIN routines. Anyone desiring to do similar modifications may contact the author for details.

APPENDIX B

DISPLAY CAPABILITIES

The Princeton terminal display can be the most powerful device on the interactive system. However, it is presently the least developed. The author spent most of his time developing the interactive environment and language processor and little time was spent on developing the display format. This appendix will outline the display currently implemented and will present several alternatives for development.

The current display is fixed in format. The lower third of the screen is reserved for text messages. The upper portion is divided into six equal sectors, each capable of displaying one standard size image (256x250). The user may specify, as a parameter to the DISPLAY command, the sector in which the image is to be displayed. If the sector is not specified, the one least recently used is re-used. If there already is an image in the specified sector, it is erased before the new image is displayed. When doing ratios, it is convenient to have several images displayed simultaneously to compare the results. However, the fixed format is not readily extendable to other display functions.

There are several other displays that would be useful with the interactive system. One would be enlarged images. The resolution of the display is limited and not all features of an image can be seen in the small size. It is desirable to expand an image to a 512x510 size. Or, one could expand a sub-matrix of an image to any desired size to locate a feature more exactly.

Another desired capability would be partial rotation and shifting of image segments on the display. Several images could be combined to form a mosaic. Piecing together a mosaic by hand is a long involved process. The images must be produced on photographic paper with the same intensity ranges, cut, and pasted onto a backing. On the interactive system, each part of the mosaic could be scaled to fit the remaining part and fitted into place in real time. When the product is complete, it could be saved in digital form for future use.

Finally, there are several displays in the form of graphs instead of images that are useful to have. A contour plot, having contours of equal intensity, is a useful alternative to an image. Also, a histogram showing intensity distributions across an image is useful for statistical analysis.

Although each of these display modes is not hard to implement in theory, some scheme is needed to control the contents of the terminal display. Variable sized displays require a flexible format on the terminal. The user must retain ultimate control over the terminal contents, being able to maintain any combination of displays concurrently. The display routines must not overwrite displays that are to be kept, and in particular, must not destroy portions of several displays.

One possible solution to the display contention problem is the development of a separate routine to control the terminal contents. Although it still seem desirable to keep the text area fixed, the upper portion of the screen should be as variable as possible. Some form of allocation/release of terminal areas could be implemented. Each routine could make an allocation call to the control routine to obtain an area to write its display. There could be some form of default freeing of areas that are to be re-used. However, the user would have to assume responsibility for freeing contents that are to be retained. In this manner, there is no limit on the types of display combinations possible.