THE IMAGE PROCESSOR

The image processor for which this set of programs were developed was IP-5500 System manufactured by De Anza, now a division of Gould. The principal components in this system consisted of:

1 MB of image memory organized in 4 pages of 512x512 bytes.
A dual 8 bit pipeline ALU array processor.
An LSI-11 microcomputer as its host controller.
A dual floppy drive storage system.
6 bit video A/D converter for realtime image digitization.
4 8 bit video D/A's for image displaying.
Dual cursor controller and joystick controller.
Alphanumeric annotation memory.
Miscellanious image enhancement accessories.

IMAGE PROCESSING SOFTWARE

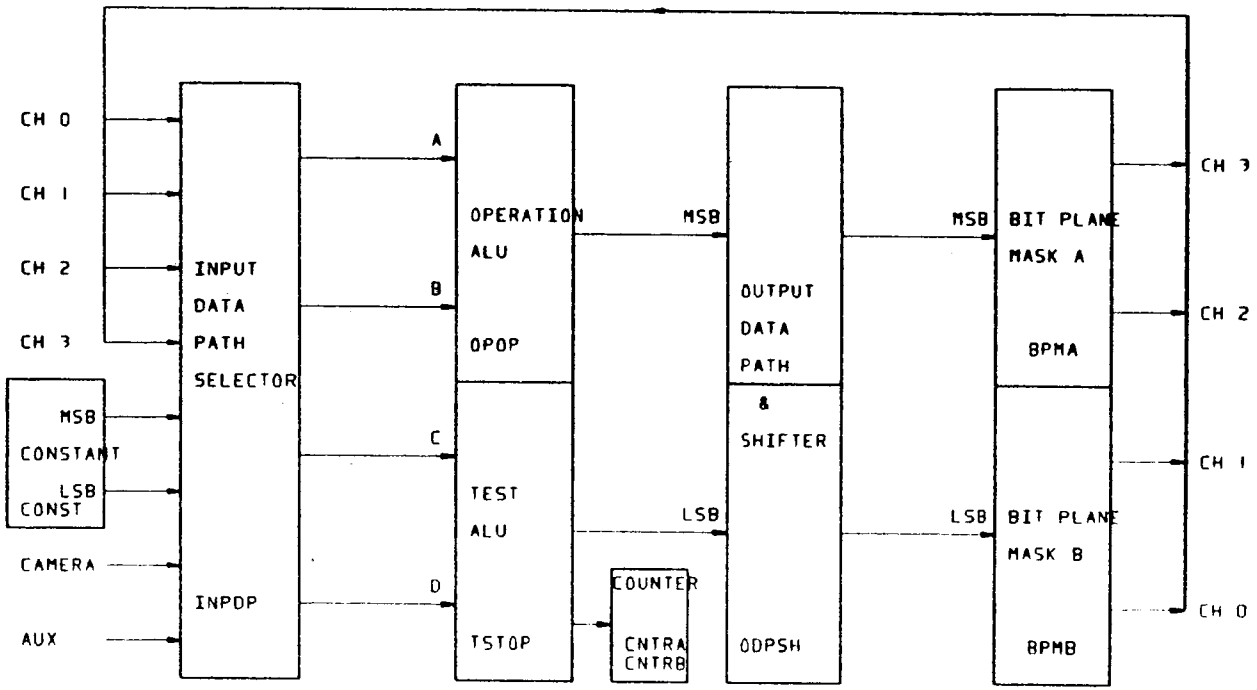
De Anza provided a library of image processing programs and library modules so that users can use them to perform quite extensive image processing operations without any programming efforts. The user can also customize specialized programs in FORTRAN and call approriate subroutines from the library, guided by the programs supplied by De Anza. It was rather flustrating if one has to learn how to use the image processor by writing FORTRAN programs to operate the image processor.

With a poly-FORTH system installed in the LSI-11 computer, I was able to interact with the image processor directly by storing data patterns into the registers in the image processor and observe the results immediately. I cannot think of any better way to converse with the image processor.

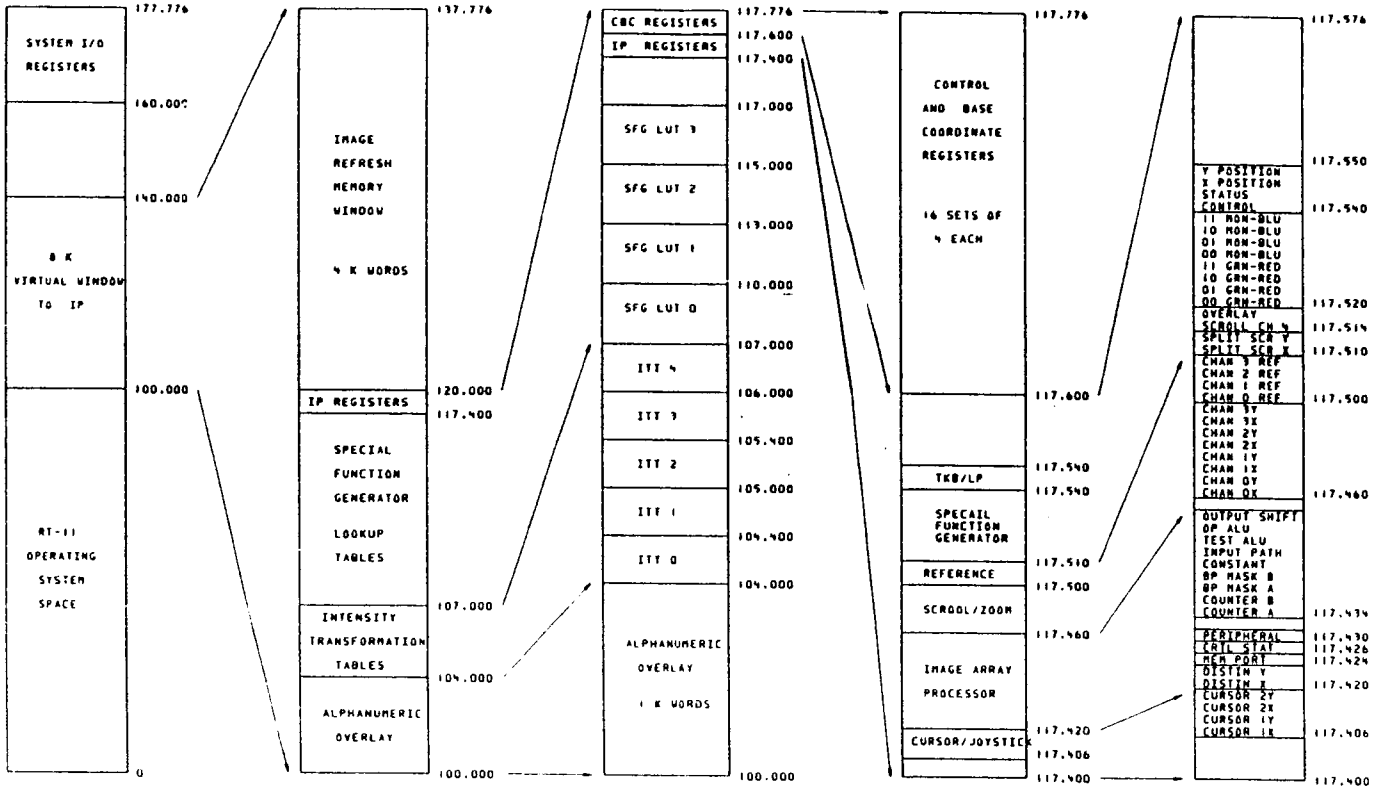
A SHORT COURSE OF IMAGE PROCESSING

Through FORTH, it is possible to access all the registers and image memory in the image processor interactively. Many of the registers produce immediately visible results on images stored in memory. Since the user can directly interact with the image processor, he can learn the in's and out's of the image processor very quickly and can do quite a lot of programming on a short learning curve.

The following is a set of screens intended to be used with the Image Processor Programming Manual provided by De Anza. Without actually using the image processor, the Manual is very dry reading. However, with FORTH one can learn the materials in the Manual in a much more interesting way with lots of experimenting.



2. Image Array Processor in IP5500



3. Memory Mapped Registers in IP5500

REGISTERS IN THE IMAGE PROCESSOR

The registers in the image processor are mapped into a 16KW window in the upper half of the LSI-11 memory, staring at octal 100000. Each register has a unique address in this window.

IS	A defining word to define named registers with the offset address from octal 100000.

Names of registers are defined following closely those names used in the De Anza Programming Manual and in the image processing library. One limitation in poly-FORTH is that names are unique only up to the first three characters and the length. Thus in many generic names, the significant characters are moved towards the beginning three character fields.

READY AND PAUSE

GO	Set the 0th bit in the ICSR register will initiate an operation, processing one frame of image data.
READY	Wait until the current image processing operation to complete by testing the 7th bit in ICSR.
FMOPER	Start an image operation and wait until it is finished in 1/30 second.
KEY-START	Clear the CSR of the console interface board, preparing it to accept a key stroke on the keyboard.
KEY-END	Test the console CSR. If a key was pushed, return a true flag. Otherwise, a false flag. This simulates the standard FORTH word ?TERMINAL.
SETUP	Clear an initialize the registers in IP for normal operations.

PEEK AND POKE

The wide acceptance of microprocessor BASIC shows its marks in FORTRAN. Many versions of FORTRAN include the famous PEEK and POKE either as libray calls or subroutines by necessity. They are simply the fetch @ and the store ! instructions in FORTH. Basically, they are just about all what's needed to control the image processor.

PEEK and POKE in this screen are the FORTH implementations of the two examples given in the IP Programming Manual. PEEK fetches out the contents of the ICSR register. POKE stores a predefined pattern, octal 101000 into ICSR.

```
  120 LIST

( IP MNEMONICS, CHT, 11/5/82)  OCTAL
: IS 100000 + CONSTANT ;
0 IS IAOVLY   4000 IS 0ITT   4400 IS 1ITT   5000 IS 2ITT
5400 IS 3ITT
7000 IS 0SFIT    11000 IS 1SFITT  13000 IS 2SFITT
15000 IS 3SFITT
17400 IS DVRGO   17406 IS X1CUR   17410 IS Y1CUR
17412 IS X2CUR   17414 IS Y2CUR   17416 IS CURCTL
17420 IS XDEST   17422 IS YDEST   17424 IS MEMPC
17426 IS ICSR    17434 IS ACNTR   17436 IS BCNTR
17440 IS ABPM    17442 IS BBPM    17444 IS CONST
17446 IS INPDP   17450 IS TSTOP   17452 IS OPOP
17454 IS ODPSH   17460 IS X0SCR   17462 IS Y0SCR
17500 IS 0REF  17502 IS 1REF   17504 IS 2REF   17506 IS 3REF
17600 IS CBCRG   20000 IS IMAGE
DECIMAL


  121 LIST

( READY AND PAUSE, CHT, 5-NOV-82)
OCTAL
: GO       1 ICSR +! ;
: READY    GO   BEGIN ICSR C@ 200 AND END ;
: PAUSE    BEGIN ICSR C@ 20 AND 0= END ;
: FMOPER   READY ;
: KEY-START    CR 1 177560 ! ;
: KEY-END      ( --- FLAG, 200 IF A KEY IS PUSHED.)
               177560 @ 200 AND ;
: KEY      ( --- ASCII)   0 'S 1 EXPECT ;
: EMIT     ( ASCII --- )  'S 1 TYPE DROP ;
: ?TERMINAL    KEY-END ;
: SETUP    XDEST 60 ERASE   3210 117510 ! ;
SETUP      ( INITIALIZE THE IMAGE PROCESSOR )
DECIMAL


  122 LIST

( PEEK AND POKE, CHT, 5-NOV-82)
OCTAL
101000 CONSTANT IBITS
: PEEK ICSR @ ;
: POKE PEEK IBITS OR ICSR ! ;
DECIMAL

EXIT
( SOME EXAMPLES OF USING PEEK AND POKE: )
PEEK .
POKE
PEEK .
PEEK U.
```

POKING AT THE IP REGISTERS

This is another example in manipulate the IP registers.

POKE-MEMPC Clear the upper 12 bits in the MEMPC register and OR 13 into its lower 4 bits. The net visible effect of this operation is to route the output of Chs. 0, 1, and 3 through their respective intensity transformation tables. If these tables were previously written some intensity transforming functions other than a normal linear function, the effect of this command is to turn on these image enhancement features.

THE CONTROL/STATUS REGISTER OF THE IMAGE PROCESSOR

This register at octal address 117426 is the most crucial register in controlling the image processor from the LSI-11. Bit 0 when set will initiate a video array processor function. Bits 1 to 8 either show the status of the image processor or are used to enable interrupts. Functions of other bits are even more profound.

The only bit which will show some demonstratable effects is bit 13, which enables the annotation overlay. The two words A-ON and A-OFF thus defined will toggle this bit without affecting other bits. If the image processor was turned on without any actions on the annotation memory, executing A-ON will cause a screenful of random characters to appear on the CRT. A-OFF will turn off this annotation.

SOME ASSEMBLY ROUTINES

ASHC This is a PDP-11 instruction only available in the EIS (extended instruction set) as an option in the LSI-11 computer. Since I had this chip in my LSI-11, I can include this instruction in the assembler.

ISHIFT Use the EIS machine code ASHC to shift a 16 bit integer either left or right.

ISWAB Swap the two byte in the integer on top of the stack. It simply makes the machine code SWB available in high level.

ICOM Complement the top item on stack.

DEPOSIT Shift ICHAN 10 bits to the left and deposit it into the UCLC (Upper Channel Lower Channel Register).

```
  123 LIST

( MEMPC POKING, CHT, 5-NOV-82)
OCTAL
13 CONSTANT IBITS
177760 CONSTANT MASK
: POKE-MEMPC     MEMPC @    MASK AND    IBITS OR
                 MEMPC !    ;
DECIMAL
EXIT
( EXECUTION EXAMPLES: )

OCTAL
MEMPC ?
POKE-MEMPC
MEMPC ?
DECIMAL


  124 LIST

( CONTROL AND STATUS REGISTER ICSR, CHT, 5-NOV-82)
OCTAL
: A-ON    ICSR @ 20000 OR    ICSR !    ;
: A-OFF   ICSR @ 157777 AND ICSR !    ;
DECIMAL


  125 LIST

( ISHIFT FUNCTION, CHT, 5-NOV-82)
ASSEMBLER   OCTAL
73000 BIN ASHC
CODE ISHIFT    0 CLR    2 S )+ MOV    1 S )+ MOV
               2 0 ASHC    S -) 1 MOV    NEXT
CODE ICOM      S ) COM    NEXT
CODE ISWAB     S ) SWB    NEXT

CBCRG CONSTANT ICBCRG
ICBCRG 6 + CONSTANT IUCLC
10 CONSTANT ICHAN
: DEPOSIT    ICHAN 12 ISHIFT IUCLC ! ;
DECIMAL
EXIT
```

WRITE INTO THE ANNOTATION MEMORY

The annotation memory is the best show piece to demonstrate the control over the image processor. If one knows where the annotation memory is in the LSI-11 memory map, he can put any character anywhere on the CRT display.

FILL — Fill a range of memory with a given byte. If this byte is a displayable character and the memory range is within the annotation memory, the characters will appear immediately on the CRT display.

FILL-ANNOTATION — Given a byte value on the stack, the whole CRT screen will be filled with this character. If you have a color monitor with the IP, you can generate characters in different colors on several background colors.

ZOOM AND SCROLL

The is also a good demonstration. You can scroll the image across the CRT display with the joystick. You have also the option of zooming and scrolling with zooming factors of 1, 2, 4, and 8. The image processor, after all, is a glorified video game.

SCROLL — Read the coordinates of cursor 1, scroll the image so that the point specified is oriented at the lower left corner of the CRT display.

ZOOM — Given a zoom factor, move the magnified image following the joystick movements.

1X, 2X, 4X, and 8X are simplified commands to zoom and scroll the displayed image.

MORE FIREWORKS

SCROLL — This command cause blackout on wrapearound regions.

CONTROL — Interactive control on SCROLL.

SCALE — Display a reference intensity scale at the bottom of the Channel 3 image.

DISABLE — Eliminate the reference scale.

There are 7 different cursor shapes. The cursors can also blink at two different rates. The following command exercise the optional features to display the cursors.

```
  126 LIST

( ANNOTATION MEMORY, CHT, 5-NOV-82)
: FILL ( ADDR COUNT BYTE --- )
       SWAP >R   OVER C!   DUP 1+   R> 1-   MOVE ;
: FILL-ANNOTATION ( BYTE ---)
       >R   IAOVLY 2048   R> FILL ;

EXIT

32 FILL-ANNOTATION
65 FILL-ANNOTATION
0  FILL-ANNOTATION
```

```
  127 LIST

( ZOOM AND SCROLL, CHT, 12-JAN-83)
OCTAL
17474 IS X3SCR   17476 IS Y3SCR
: SCROLL ( MAG --- )   Y1CUR @ + Y3SCR !   X1CUR @ X3SCR !   ;
: ZOOM   ( ZOOM-FACTOR --- )
         KEY-START BEGIN DUP SCROLL ?TERMINAL END DROP ;
: 1X     0 ZOOM ;
: 2X     2000 ZOOM ;
: 4X     4000 ZOOM ;
: 8X     6000 ZOOM ;

DECIMAL
EXIT
STORE DIFFERENT VALUES INTO X3SCR AND Y3SCR WOULD CONTROL THE
SCROLLING AND ZOOMING OF CHANNEL 3 IMAGE.
```

```
  128 LIST

( SCROLL WITH WRAPEAROUND, CHT, 12-JAN-83)    OCTAL
: SCROLL   X3SCR @ 7000 AND   X1CUR @ 777 AND    + X3SCR !
           Y3SCR @ 7000 AND   Y1CUR @ 777 AND    + Y3SCR !   ;
: CONTROL  KEY-START BEGIN SCROLL ?TERMINAL END ;

( INTENSITY REFERENCE SCALE IN CH3 DISPLAY)
: SCALE    ( SET CH3 REFERNECE SCALE)   0 3REF ! ;
: DISABLE  ( ERASE CH3 SCALE )          4 3REF ! ;

( CURSOR CONTROLS )
: CURSORS  ( ENABLE CURSORS )    67 CURCTL ! ;
: BOX      ( CURSOR BOX      )   60 CURCTL ! ;
: CROSS    ( CURSOR CROSSES )    62 CURCTL ! ;
: BLINK    ( SLOW BLINK )        367 CURCTL ! ;
: FAST     ( FAST BLINK    )     377 CURCTL ! ;
DECIMAL
```

SPECIAL FUNCTION GENERATOR

Special function generator is an option in De Anza IP. It allows the user to scroll though the entire 1 MB image memory organized as one huge 1024x1024 image. Only a 512x512 window in this superimage is visible on the CRT display.

DEMO — Like CONTROL in the last screen, but scroll the superimage.

Modifying the JFILE, XSPLT, and YSPLT registers in the special function generator add some interesting variations to the DEMO command.

CONTOLLING THE DIGITAL VIDEO ARRAY PROCESSOR

Since we are going to do lots of register storing and register fetching, it is more appealing to have some specialized tools to facilitate the use of these functions.

R/W — Defines registers which can be read and written. It is simply the CONSTANT in FORTH.

R/O — Defines read only registers. Invoking the register name would return the register contents on the stack.

W/O — Defines write-only registers. Invoking the register will store a stack value into the register.

Using these defining words to define IP register eliminates lots of @'s and !'s, thus making the program much concise and more readable. My earlier programs looked ugly with @'s and !'s sprinkle all over the screens.

SYSTEM INITIATION

SETUP — Initialize all the IP registers in a regular De ANza IP-5500 without special function generator option.

3DUP — Make 3 copies of the top stack item.

SFGINT — Initialize all the relevant registers in the special function generator.

SYSINT — Setup command for IP with special function generator.

```
  129 LIST

( SPECIAL FUNCRION GENERATOR, CHT, 8-NOV-82)
OCTAL   17510 IS XSPLT   17512 IS YSPLT
17520 IS JFILE
: DEMO KEY-START BEGIN 1XCUR @ 4000 OR XSPLT !
       1YCUR @ 4000 OR YSPLT ! KEY-END END ;
0 JFILE !
0 JFILE 10 + !
3131 JFILE 2 + !
3131 JFILE 12 + !
606 JFILE 4 + !
606 JFILE 14 + !
1303 JFILE 6 + !
723 JFILE 16 + !
4140 XSPLT !
4600 YSPLT !
DECIMAL


  130 LIST

( DIGITAL VIDEO PROCESSOR, CHT, 8-NOV-82)
OCTAL
: R/W CONSTANT ;
: R/O CONSTANT DOES> @ @ ;
: W/O CONSTANT DOES> @ ! ;
117420 R/W XDEST   117422 R/W YDEST   117424 W/O MEMPC
117434 R/W ACNTR   117436 R/W BCNTR
117440 W/O ABPM    117442 W/O BBPM
117444 W/O CONST   117446 W/O INPDP
117450 W/O TSTOP   117452 W/O OPOP
117454 W/O ODPSH
DECIMAL
EXIT
R/O DEFINES READ-ONLY REGISTERS, AND W/O DEFINES WRITE-ONLY
REGISTERS.  THEY WILL SAVE PROGRAMMING SPACE IN HANDLING THE
DIGITAL VIDEO PROCESSOR FUNCTIONS.


  131 LIST

( SYSTEM INITIATION, CHT, 10-NOV-82)
OCTAL
: SETUP    ICSR @ >R   XDEST 60 ERASE   R> ICSR !
           50120 OPOP   10 MEMPC   67 CURCTL !   ;
: 3DUP     DUP 2DUP ;
: SFGINT     4 3DUP 0REF 2!     0REF 4 + 2!
             0 3DUP 117510 2!   117510 4 + 2!
           504 3DUP 117520 2!   117520 4 + 2!
           706 3DUP 117530 2!   117530 4 + 2! ;
: SYSINT   SETUP SFGINT ;

: ENABLE   1 ICSR +!   BEGIN ICSR C@ 200 AND END   ;
DECIMAL
```

IMAGE MEMORY INITIATION AND COPYING

These are the most elementary image processing operations. Four image channel are assigned dedicated functions in our design of realtime radiographic applications. Channels 0 and 1 are used as a 16 bit deep image accumulator for processings requiring extended precision. Ch. 3 is the display channel and Ch. 2 is a temporary image storage area.

CLRMEN Clear Channel 3 memory by writing zero through the video array processor. 3 TSTOP specifies 0 output, and BBPM enables writing to Channel 3 memory.

SETMEM Write the byte value on stack to all memory in Ch. 3. Generate an uniform image.

CPYMEM The source channel number and the write enable masks are given on the stack. Source channel image is copied to any or all other channels.

Other words are simple extensions of CPYMEM.

IMAGE INPUT AND DIGITIZATION

PICTURE Digitize one frame of image from the input video camera and store it in Ch. 3 memory for displaying.

CAMERA Continuously input images from the video camera and dispaly the live images in Ch. 3. The sequence can be stopped by pushing any key on the keyboard. The last frame will be frozen in Ch. 3.

IMAGE INTEGRATION

Realtime video images are noisy, especially under low illumination. Image integration is the most effective way in reducing this noise, assuming the scene is stationary.

CLEAR Zero Ch. 0 and 1, to be used as image accumulator.

INTEGR Given the number of frames to be integrated, add that number of input video images to the image accumulator.

SHIFT The accumulated image has to be left-justified so that it can be displayed with the identical grayscale as that for the realtime images. The justification is accomplished by shifting rather than division due to SHIFT uses the Output-data-path-shifter in the video array processor to shift the 16 bit accumulated image for the bit position specified on stack.

```
  132 LIST

( CLRMEM, SETMEM, CPYMEM, CHT, 10-NOV-82)
OCTAL
: CLRMEM    SETUP   3 TSTOP   177400 BBPM   ENABLE   ;
: SETMEM    ( N --- )   SETUP   CONST   4 INPDP   52 TSTOP
            177400 BBPM   ENABLE   ;

: CPYMEM    ( CH# ABPM BBPM --- )
            SETUP   BBPM ABPM INPDP   52 TSTOP   ENABLE   ;
: 0COPY     0 0 177400 CPYMEM ;
: 1COPY     1 0 177400 CPYMEM ;
: 2COPY     2 0 177400 CPYMEM ;
: 0SAVE     3 377 0 CPYMEM ;
: 1SAVE     3 177400 0 CPYMEM ;
: 2SAVE     3 0 377 CPYMEM ;
DECIMAL


  133 LIST

( PICTURE, CAMERA, CHT, 10-NOV-82)
OCTAL
: PICTURE    ( GRAB ONE FRAME INTO CH 3)
          302 ODPSH   0 ABPM 177400 BBPM
          100 INPDP   20 TSTOP   ENABLE ;

: CAMERA ( SHOW PICTURES IN CH 3, UNTIL KEY STOKE )
          KEY-START   BEGIN   PICTURE   ?TERMINAL   END   ;
DECIMAL


  134 LIST

( CLEAR, INTEGR, SHIFT, CHT, 10-NOV-82)
OCTAL
: CLEAR    SETUP   3 TSTOP   -1 ABPM   ENABLE   ;
: INTEGR   ( FRAMES --- )
           10100 INPDP   31 TSTOP   50100 OPOP
           5000  ODPSH   -1 ABPM 0 BBPM
           0 DO   2000 XDEST +!   ENABLE   LOOP
           CR ." DONE."   ;
: SHIFT    ( N --- )
           5030 OR ODPSH   10000 INPDP   52 TSTOP
           50120 OPOP   -1 ABPM 0 BBPM   ENABLE   ;
DECIMAL
```

EXTRACT EXPONENTS FROM NUMBERS

EXAMINE Divide top item on stack by 2 and increment the second number by 1. If the top item is 0, leave the stack numbers unmodified.

EXPON Reduce the top number on stack to the nearest power of two and replace it with this power.

WHOLE Given n on the stack, replace it by 2**n.

EXPONENT A variable storing the 2's power of the number of frames to be integrated. It will be used to left justify the integrated image by logical shifts.

FRAME Variable storing the number of frames to be integrated. Number of frames must be 2's power.

FRAMES Convert the stack number to the largest power of 2 smaller of equal to it and store the result in FRAME. Also store the 2's power in EXPONENT.

IMAGE INTEGRATION

10COPY Copy Ch. 0 to Ch. 1. The result of image integration should end up left justified in Ch. 1. However, the Output-Data-Path-Shifter in the video array processor shift the image accumulator left 1 to 3 bits or shift right 1 to 4 bits if the shift number is from 4 to 7. When the justified image ends in Ch. 0, it must be moved to Ch. 1 for output.

ADJUST If frame number is greater than 256, the accumulator is shifted to right. Execute 10COPY in this case.

INTEGRATE Clear the image accumulator and add the number of digitized frames to it as specified by FRAME. Shift the accumulated image by EXPONENT bits and move the left justified result into Ch. 3 for display.

TAKE A COLOR PICTURE --- TAKPXC

To get a color image, one has to digitize three realtime images in the red, green, and blue spectral regions.

?READY Pause with a prompting message to allow th user to put the appropriate filter in front the lens.

TAKPXC Take in three digitized images and store them into Ch. 0, 1 and 2, which are then output to the red, green, and blue channel of a color CRT monitor.

```
  135 LIST

( EXAMINE, EXPON, WHOLE, CHT, 10-NOV-82)
: EXAMINE    ( EXPON N --- EXPON+1 N/2 )
         DUP    IF 2/ SWAP 1+ SWAP THEN    ;
: EXPON    ( N --- POWER-OF-2)
         -1 SWAP    BEGIN    EXAMINE DUP 0=    END    DROP ;
: WHOLE    ( N --- 2**N )
         1 SWAP    0 DO 2* LOOP ;
VARIABLE EXPONENT    VARIABLE FRAME
: FRAMES    ( N --- )
         EXPON    DUP EXPONENT !    WHOLE FRAME !    ;
```

```
  136 LIST

( TAKPIX, CHT, 10-NOV-82)
OCTAL
: 10COPY    0 177400 0 CPYMEM    ;

: ADJUST    FRAME @    200 <    IF 10COPY THEN    ;

: INTEGRATE    ( USE THE FRAME# DEFINED BY FRAMES AND INTEGRATE)
            ( THAT MANY FRAMES.  SHOW RESULT IN CH 3.)
            CLEAR    FRAME @ INTEGR    12 EXPONENT @ - SHIFT
            ADJUST    1COPY    ;
DECIMAL
```

```
  137 LIST

( TAKPXC, CHT, 10-NOV-82)

: ?READY    CR ." TYPE A KEY WHEN YOU ARE READY."
            KEY DROP ;

: TAKPXC    ?READY    PICTURE 0SAVE
            ?READY    PICTURE 1SAVE
            ?READY    PICTURE 2SAVE
            CR ." COLOR IMAGE SNAPPED."    ;
```

A SCREEN EDITOR FOR IMAGE ANNOTATION

This screen editor allows the user to put messages with the image anywhere on the CRT display. The displayed texts can be save on disk for latter recall.

KPOINTER Screen cursor pointer containing the position of the current cursor in terms of absolute memory address.
ANNOT Memory origin of the annotation memory page.
KREVERSE Display the current character in reverse video to serve as a visible cursor.
KBEG From the cursor pointer on stack, return the pointer to the beginning of the current line.
KNORMAL Reset the current character from reversed video to normal video of white on black background.
KPUT Store the address on stack into KPOINTER and display that character in reverse video.

SCREEN EDITOR

KBACK, KUP, KLF, KHOME, KCR
Move the cursor back, up, down, home, or to a new line, respectively, by manipulating KPOINTER and setting modes for pointed characters.
KCLEAR Clear the entire annotation memory to zero.
KINSERT Insert the character on the stack to the current cursor and move the cursor one place forward.
NGAIN, NOFFSET, FRAME, NDISTANCE
Variables to be display as default annotations.
SELECT Get an Ascii code from keyboard. If it is a function code, do the specified function. Otherwise insert the code into the screen. If the code is ESC, leave a true flag on the stack.

SCREEN EDITOR

A-ON Turn on the annotation overlay.
A-OFF Turn off the annotation overlay.
TITLER Enter the screen editor to put texts on screen. Exit by pressing the ESC key.
MESSAGE A super string. The substrings are extracted and put on the screen as default annotations.
'MESSAGE Return the address of a substring in MESSAGE.
TITLE Copy the MESSAGE substrings and arrange them on the lefthand side of the screen.

```
  138 LIST

( SCREEN EDITOR, 5-22-80 )  OCTAL
VARIABLE KPOINTER       100120 CONSTANT ANNOT
: KREVERSE KPOINTER @ DUP C@ DUP IF 200 + ELSE DROP 240 THEN
           SWAP C! 0 ;
: KBEG DUP 3777 AND 120 /MOD DROP - 103777 AND ;
: KNORMAL  KPOINTER @ DUP C@ 77 AND SWAP C! ;
: KPUT     KNORMAL KPOINTER ! KREVERSE ;
: KBACK    KNORMAL KPOINTER @ 1- KPUT ;
: KUP      KNORMAL KPOINTER @ 120 - KBEG  KPUT ;
: KLF      KNORMAL KPOINTER @ 120 + KBEG  KPUT ;
: KHOME    KNORMAL ANNOT KPUT ;
: KCLEAR   ANNOT DUP 4000 ERASE KPUT ;
: KINSERT  KNORMAL KPOINTER @ C! 1 KPOINTER +! KREVERSE ;
: KCR      KNORMAL KPOINTER @ DUP 120 + KBEG  DUP ROT
            DO 0 I C! LOOP KPOINTER ! KREVERSE ;
DECIMAL


  139 LIST

( SCREEN EDITOR, CONT'D)  OCTAL
VARIABLE NGAIN VARIABLE NOFFSET VARIABLE FRAME
VARIABLE NDISTANCE
: SELECT ( LEAVE FLAG, TRUE IF ESC, FALSE IF OTHER KEYS)
         KEY   DUP 33 = IF 1 KNORMAL   ELSE
               DUP 0 = IF KHOME   ELSE DUP 4 = IF KCLEAR
               ELSE DUP 15 = IF KCR   ELSE DUP 12 = IF KLF
               ELSE DUP 177 = IF KBACK ELSE DUP 10 = IF KUP
               ELSE DUP KINSERT
         THEN THEN THEN THEN THEN THEN THEN SWAP DROP ;

: A-ON   20000 ICSR ! ;
: A-OFF  0 ICSR ! ;
: TITLER A-ON ANNOT KPOINTER ! KREVERSE
         BEGIN SELECT END ;
DECIMAL


  140 LIST

( TITLE, 5-27-80) OCTAL
: MASSAGE    ( A SUPER STRING TO BE ARRANGED ON CRT)
 ." R I P L DEMONSTRSYSTEM  FRAMES  GAIN    OFFSET  DISTANCE" ;
: 'MASSAGE   ( --- ADDR,  THE BASE ADDRESS OF THE SUPER STRING)
       ['] MASSAGE 3 + ;
: TITLE      ( ARRANGE STRINGS ON CRT.)
        'MASSAGE ANNOT OVER OVER 10 MOVE
        OVER 10 + OVER 120 + 10 MOVE
        OVER 20 + OVER 240 + 10 MOVE
        OVER 30 + OVER 2400 + 10 MOVE
        OVER 40 + OVER 2640  + 10 MOVE
        OVER 50 + OVER 3100 + 10 MOVE
        SWAP 60 + SWAP 3340 + 10 MOVE ;
DECIMAL
```

WRITE NUMBERS TO THE SCREEN

.CRT (addr n ---)
Convert the number n into a string of 4 Ascii characters and copy them to the addr in the annotation memory.

.DATA Fill the appropriate fields on the screen with data stored in the respective variables.

TITLE Redefine TITLE to include the numeric data.

DISPLAY THE SCREEN EDITOR MENU ON SCREEN

SHOWMENU This is a demonstration of screen editor. It moves the contents of block 144 and display them as 16 lines of texts on the screen.

SAVE AND RETRIEVE ANNOTATION TEXTS TO/FROM DISK

SVPANX Given the block number on the stack, store the current contents of the annotation memory in two consecutive blocks on the disk.

SHPANX Given the block number on the stack, read two consecutive blocks of texts and display then on the screen.

```
  141 LIST

( WRITE NUMBERS TO CRT, CHT, 11-NOV-82)
OCTAL
: .CRT 0 SWAP PAD 10 ERASE <# #S #> DROP SWAP 4 MOVE ;
: .DATA    ANNOT DUP 2520 + FRAME @ .CRT  DUP 2760 + NGAIN @
 .CRT DUP 3220 + NOFFSET @ .CRT 3460 + NDISTANCE @ .CRT ;
: TITLE TITLE .DATA ;
DECIMAL
```

```
  142 LIST

( SHOWMENU, CHT, 11-NOV-82)

: SHOWMENU    ( MOVE CONTENTS OF BLOCK 144 TO ANNOTATION.)
       KCLEAR
       16 1 DO
       144 BLOCK I 64 * +   ANNOT I 80 * +   64 MOVE
       LOOP   ;
```

```
  143 LIST

( SVPANX, SHPANX, CHT, 11-NOV-82)
: SVPANX ( BLOCK --- ,SAVE ANNOTATION ON 2 CONSECUTIVE BLOCKS.)
      ANNOT   OVER BLOCK 1024 MOVE   UPDATE
      1+ BLOCK   ANNOT 1024 +   SWAP 1024 MOVE
      UPDATE FLUSH   ;

: SHPANX ( BLOCK ---, SHOW ANNOTATION.)
      DUP BLOCK ANNOT 1024 MOVE
      1+  BLOCK ANNOT 1024 +   1024 MOVE   ;
```

HISTOGRAM OF AN IMAGE

The De Anza IP has a counter in the video array processor, which can be incremented depending upon the ALU results. This counter can be set up so that it will accumulate histograms.

MEMORY An double integer array to store a histogram.
SETUP-HIST Set up the IP registers to control the counter incrementation.
HISTO Run through the counting loop 256 times and store the histogram data in MEMORY. Note that the last bin of gray level 255 has to be processed separately.
GTHIST Collect the histogram from the Ch. 3 image. Reset all IP registers after the operation.
HDUMP Dump the contents of the MEMORY array to the console.

HISTOGRAM

HISTOGRAM Given the range of gray levels, print the histogram data as double integers in this range. HDUMP in the last screen prints data in 16 bit integers, which are alright to the programmer but not quite appropriate to the final user.

MEASURING DISTANCES BETWEEN TWO CURSORS

0SQRT Given the cursor coordinates on the stack, return the square of the distance as a double integer.
1SQRT From the differences in x and y directions, calculate the seed of square root as the larger distance plus half of the shorter distance.
2SQRT, 3SQRT Apply Newton's rule twice to get the square root. The choice of seed assures the accuracy of the result.
DISTANCE Get the cursor coordinates from the cursor registers and return the calculated distance on the stack.
RATI A variable containing the scale facter for the value of distance to be displayed with the image.
RATIO Change the scale factor RATI to the stack value.
Q Measure the cursor distance and return the scaled value.

```
  147 LIST

( HISTOGRAM UTILITIES, CHT, 11-NOV-82 )
OCTAL
VARIABLE MEMORY 2000 ALLOT
: SETUP-HIST   SETUP   32064 INPDP   100006 TSTOP
          13026 OPOP  ;
: HISTO   400 377 0 DO
             0 ACNTR ! DUP CONST ENABLE
             ACNTR 2@ SWAP I 2* 2* MEMORY + 2! 401 +
          LOOP DROP
          170026 TSTOP   0 ACNTR !   -1 CONST   ENABLE
          ACNTR 2@ SWAP   MEMORY 1774 + 2!   ;
: GTHIST  SETUP-HIST HISTO SETUP ;
: HDUMP   MEMORY 2000 DUMP ;
DECIMAL



  148 LIST

( HISTOGRAM, 5-16-80 )
31 LOAD 37 LOAD ( MATH FUNCTIONS )
OCTAL
: HISTOGRAM ( INIT FINAL --- )
            ( PRINT HISTOGRAM VALUES BETWEEN LIMITS.)
         1+ SWAP DO
         I 10 /MOD DROP 0= IF CR I 5 .R THEN
         MEMORY I 4 * + 2@   10 D.R
         LOOP ;
DECIMAL









  149 LIST

( DISTANCE BETWEEN CURSORS, 5-16-80) OCTAL
: 0SQRT ABS SWAP ABS 2DUP 2DUP ROT M* 2SWAP M* D+ ;
: 1SQRT 2SWAP 2DUP MAX ROT ROT MIN 2/ + ;
: 2SQRT DUP 2OVER ROT M/ + 2/ DUP ;
: 3SQRT 2SWAP ROT M/ + 2/ ;
: SQRT 0SQRT 1SQRT 2SQRT 3SQRT ;

OCTAL 117406 CONSTANT X1 117410 CONSTANT Y1
      117412 CONSTANT X2 117414 CONSTANT Y2    DECIMAL
: DISTANCE X1 @ 511 AND X2 @ 511 AND - Y1 @ 511 AND Y2 @
           511 AND - 2DUP OR IF SQRT ELSE DROP DROP 0 THEN ;

( SCALING THE DISTANCE)
VARIABLE RATI   1000 RATI !
: RATIO RATI ! ;
: Q DISTANCE RATI @ 1000 */ . ;
```

ACCESSING IMAGE MEMORY

It is quite difficult to understand how the image memory is accessed in the De Anza IP, inspite of the rather lengthy discussions in the De Anza IP Programming Manual. Like most technical manuals, you will have to understand the materials before you read them. Otherwise, they offer little help.

The entire 1MB image memory is mapped through a 8 KB window in the LSI-11 memory. There are 16 sets of Control and Base Coordinate Registers, and each set defines a 512 btye block in the image memory and maps it to a block of LSI-11 memory. If it is that simple, there will be no problems. However, there are 4 different modes to use these registers, and there are three more bits in the ISCR register having no obvious effects on these modes.

TRACING A PROFILE IN THE IMAGE

I will not try to explain the mess here. I can't, even if I wanted to. All I can do is to present some working examples and hope that they will be useful to the reader.

CSR	The Control/Status Register of the image processor.
CURSOR	The first cursor coordinate registers.
CBCR	The first Control and Base Coordinate Registers.
IMAGE	The first image data window controlled by the above CBCR.
ST-CBCR	Four numbers on the stack are loaded into the first set of CBCR's.
ROW	Define the first set of CBCR's to access the nth row of the image in Ch. 3. Mode 1, auto indexing in x, zero upper byte on read, no indexing in y, both input channels are set to Ch. 3, auto increment after reading 256 words, delta mode set in CSR. These bits in CBCR and CSR allow me to read one line of 512 bytes of data in Ch. 3, th row number is given on the stack as input.
R-ROW	Using data mode, reading IMAGE will automatically increment the image coordinate. Use the cursor to point to the row I want to draw a profile, and this word will grab 512 pixel data and pile them onto the data stack.
S-ROW	Print the profile data from stack to console for verification.
PROFILE	Draw each point of the profile on Ch. 3 image at the 255th gray level showing a white trace. Each dot is drawn by putting the coordinates directly into the x/y CBCR registers.
PROFIL	Draw the profile in black dot at the 0th gray level so that the trace will be visible in the white image area

```
  150 LIST

( PROFILE, CHT, 12-JAN-83)    OCTAL
117426 CONSTANT CSR    117406 CONSTANT CURSOR
117600 CONSTANT CBCR   120000 CONSTANT IMAGE
: ST-CBCR ( N M K L ST-CBCR LOAD CBCR'S.)
        CBCR 4 + 2! CBCR 2! ;
: ROW    ( READ A ROW, MODE 1 )
        100000 + 42000 7777 377 ST-CBCR 20000 CSR ! ;
: R-ROW ( READ ONE ROW TO STACK )
        CURSOR 2+ @ 777 AND ROW 1000 0 DO IMAGE @ LOOP ;
: S-ROW 1000 0 DO 14 0 DO 6 U.R LOOP CR 14 +LOOP ;
: DRAW  1000 0 DO 777 I - CBCR ! CBCR 2+ ! 377 IMAGE ! LOOP ;

: PROFILE ( GET ONE ROW AT CURSOR, DISPLAY PROFILE ON CRT )
        R-ROW  DRAW ;
DECIMAL


  151 LIST

(  STORE IMAGE DATA ) OCTAL
117426 CONSTANT CSR    117406 CONSTANT CURSOR
117600 CONSTANT CBCR   120000 CONSTANT IMAGE
: ST-CBCR ( N M K L ST-CBCR LOAD CBCR'S.)
       CBCR 4 + 2! CBCR 2! ;
: ROW  (  N ROW SELECT N'TH ROW OF IMAGE FOR READING.)
       100000 + 42000 7777 377 ST-CBCR 20000 CSR ! ;
: R-ROW  ( READ ONE ROW TO STACK )
         CURSOR 2+ @ 777 AND ROW 1000 0 DO IMAGE @ LOOP ;
: S-ROW 1000 0 DO 14 0 DO 6 U.R LOOP CR 14 +LOOP ;
: PROFILE 1000 0 DO 777 I - CBCR ! CBCR 2+ ! 377 IMAGE ! LOOP ;
 : PROFILE ( GET ONE ROW AT CURSOR, DISPLAY PROFILE ON CRT )
          R-ROW  PROFILE ;
: PROFIL 1000 0 DO 777 I - CBCR ! CBCR 2+ ! 0 IMAGE ! LOOP ;
: PROFIL R-ROW  PROFIL ;
DECIMAL
```

IMAGE SUBTRACTION

This screen is from a very early program in which the registers in the De Anza Image processor were not named. It is very difficult to figure out what's going on. It was also wasteful because numbers were stored as literals.

SUB	Subtract Ch. 1 from Ch. 2 and store result in Ch. 0.
0MOVE	Subtracted image in Ch. 0 is in 2's complement. It is shifted up by 128 gray levels so that positive and negative values can be displayed contiguously. Result is moved into Ch. 3 to be displayed.
0CMOVE	Same as 0MOVE. Copy only the area bound by cursors.
SUBTRC	Subtract Ch. 1 from Ch. 2 and display the shifted result in Ch. 3.
CSUBTRC	Same as SUBTRC. Display only the cursor bound area.
CCOPY	Copy the cursor bound area from Ch. 0 to Ch. 3.

LOCAL AVERAGING OR SUBGROUP AVERAGING

X3SCR, Y3SCR The scroll registers for Ch. 3 image output.

SADD	Add the Ch. 3 image to the image accumulator(Ch. 0&1) through the scrolling circuitry. The displacement is specified by the contents in X3SCR AND Y3SCR.
SUBGROUP	Perform a 4x4 local averaging on the image in Ch. 3. The result is copied back to Ch. 3. The original images in Ch. 0, 1, and 3 are distroyed.

A, B, C, D, E, F are shorthand definitions for CAMERA, INTEGRATE 2SAVE, SUBTRC, 1COPY, and 2COPY, which are the most often used commands in a realtime image processor. The operator really does not like to do much typing.

LOW AND HIGH PASS IMAGE FILTERING

Local averaging is a very effective way of removing the high frequency noises in real time images. They serve well as low pass image filters. Subtracting the local averaged image from the original image, we can synthesize high pass image filter.

4HPASS	Save Ch. 3 in Ch. 2, do a 4x4 local averaging on Ch. 3 subtract averaged image from the original, and display the high pass filtered image in Ch. 3.
8SUBG	Perform an 8x8 local averaging on Ch. 3 image.
8HPASS	Using the 8x8 local average to get a high pass image.
16SUBG	16x16 local averaging.
16HPASS	16x16 high pass filtering.

```
  153 LIST

(   IMAGE SUBSTRACTION : SUBTRC   ) OCTAL
: SUB 41 117446 ! 6 117450 ! 0 117454 ! 377 117440 !
        0 117442 ! FMOPER ;
: 0MOVE  4 117446 ! 200 117444 ! 31 117450 ! 177400 117442 !
        0 117440 ! FMOPER ;
( CSUBTRC MOVES CURSOR BOUNDED IMAGE TO DISPLAY, 5-20-80)
: 0CMOVE 4 117446 ! 200 117444 ! 431 117450 ! 177400 117442 !
        0 117440 ! FMOPER ;
: CSUBTRC SUB CLRMEM 0CMOVE ;
: SUBTRC SUB 0MOVE ;
: CCOPY 0 117446 ! 420 117450 ! 0 117454 ! 177400 117442 !
        0 117440 ! FMOPER ;
DECIMAL




  154 LIST

( SUBGROUG AVERAGING, CHT, 12-JAN-83 )  OCTAL
117474 W/O X3SCR   117476 W/O Y3SCR
: SADD  10003 INPDP   31 TSTOP   50100 OPOP
        5000  ODPSH   ENABLE   ;
: SUBGROUP   CLEAR   1003 777 DO   I X3SCR
        1003 777 DO   I Y3SCR   SADD   LOOP LOOP
       4 SHIFT   CH1   ;
DECIMAL
```

IMAGE ENHANCEMENTS

The image enroute to the D/A can be modified by an image transformation table (ITS), adding enhancements to the image. Transformation functions can be specified in the ITS, which occupies a 256 byte block starting at octal 105400.
ITS-SHOW Route Ch. 3 image through the ITS table to CRT.

We use an S shaped 3 piece straight lines to represent the intensity transformation curve. The central line is centered about the gray level specified by the variable NOFFSET, which will be transformed to level 128 as output. A straight line is drawn through this point with a slope of 1 up to 128, enhancing the levels about NOFFSET with a gain as specified by NGAIN. This straight line is clipped to 255 at the high side and to 0 at the low side, resulting in the S-shaped functions.

IMAGE ENHANCEMENTS

ITS-SHOW	Route Ch. 3 image through the ITS table to CRT. The displayed image is transformed according to ITS.
+!ITS	Given a byte value and the offset from the beginning address of ITS table, store that byte into the proper location in the ITS table.
UNITY	Write a straight line in ITS. The output image is not modified.
HVALUE	Fill the upper half of the ITS, starting at the point specified by the offset parameter in NOFFSET.
LVALUE	Fill the lower half of the ITS.
IP5ITS	Draw the ITS curve as specified by NOFFSET and NGAIN, route Ch. 3 image through it, and update the numeric annotation to show the updated NOFFSET and NGAIN.
GAIN	Set the gain to the value given on stack.
OFFSET	Set the offset of ITS according to the value given.

INTERACTIVE CONTROL OF IMAGE ENHANCEMENTS

KK	Decrement NOFFSET and update the ITS curve.
KJ	Increment NOFFSET and update the ITS curve.
KH	Increment NGAIN and update the ITS curve.
KL	Devrement NGAIN and update the ITS curve.
KEY	The standard keyboard input command.
KITS	Keyboard-controlled ITS. It monitors the keyboard and responds to the H, J, K, L, and RETURN keys. H increases the gain, L decreases the gain, J increases the offset thus darkens the image, and K decreases the offset thus brightens the image. RETURN terminates the loop. This command allows the user to enhance the image interactive to the point where he gets the visually most appealing enhancement,

```
  156 LIST

( INTENSITY TRANSFORM, CHT, 7-MAR-83)
105400 CONSTANT 3ITT
VARIABLE NGAIN          VARIABLE NOFFSET
: SHOW    10 MEMPC ;
: !ITT    3ITT + C! ;
: UNITY   400 0 DO I DUP !ITT LOOP SHOW ;

: HVALUE  200 NGAIN @ - 400 NOFFSET @
          DO NGAIN @ + 377 MIN DUP I !ITT LOOP DROP ;
: LVALUE  200 0 NOFFSET @ DO
          NGAIN @ - 0 MAX DUP I !ITT -1 +LOOP DROP ;

: TRANSFORM  HVALUE LVALUE SHOW  ;
: GAIN NGAIN ! TRANSFORM ;      : OFFSET NOFFSET ! TRANSFORM ;
DECIMAL



  157 LIST

(   KEYBOARD CONTROL OF ITT   )
OCTAL
: KK    NOFFSET @ 1- 0 MAX NOFFSET ! TRANSFORM ;
: KJ    NOFFSET @ 1+ 377 MIN NOFFSET ! TRANSFORM ;
: KH    NGAIN @ 1+ 100 MIN NGAIN ! TRANSFORM ;
: KL    NGAIN @ 1- 1 MAX NGAIN ! TRANSFORM ;

: ENHANCE    BEGIN KEY
               DUP 110 =    IF KH THEN
               DUP 112 =    IF KJ THEN
               DUP 113 =    IF KK THEN
               DUP 114 =    IF KL THEN
             15 = END   ;
DECIMAL



  158 LIST
```

SAVING IMAGE TO DISK

In addition to the accessing to image memory data, here we will have to deal with the disk controller also. The disk used here was FT-101 made by Scientific Micro Systems, Inc. It can read or write directly from or to memory upto 32KB a time. What one has to do is set up a parameter packet in the memory, specifying the exact nature of the next transaction, and then give the address of the packet to the disk control/status register. When the disk finishes the current transaction, kick the disk controller by setting a bit in the control register. The disk controller will use the DMA scheme to transfer data either from the disk into memory or from memory to disk.

SETUP DISK CONTROLLER AND THE IP CBCR

CBCR	The first set of Control and Base Coordinate Registers
SPAC	The address of the command packet to disk controller.
R/W	Wait till disk is not busy, point the packet address to SPAC, and start the disk transaction.
OPRNDS	Setup the SPAC parameters to transfer one track of data, an 8 KB block.
SET-CBCR	Setup CBCR for image memory access in mode 2, upper and lower channels are both Ch. 3, autoincrementing row address after transferring 512 bytes.
TRACKS	Read or write 32 contiguous tracks from the track specified by top stack item.
READ	Read a frame of image or 256 KB of data from disk, starting at the track number given on stack.
WRITE	Write a frame of image data from Ch. 3 to the disk.
FORMAT	Format the diskette in the second disk drive.

IMAGE READ AND WRITE

CHOOSE	A double density 8" diskette can hold 2 frames of images. The user can choose to access either image by giving a number on the stack. 1 will cause R/W to start at track 35, otherwise start at track 0.
READ	Read one image from disk to Ch. 3.
WRITE	Write one image from Ch. 3 to disk.

Some examples are given here, showing how to use the read/write commands.

```
   ( SMS ARGUMENT PACKET FORMATION )     OCTAL
117600 CONSTANT CBCR
CREATE SPAC    0 , 0 , 100000 , 17 , 160000 , 0 , 144120 ,
16 ALLOT
: R/W    BEGIN 176020 @ 0< NOT END
         0 176024 !    SPAC 176026 !    5000 176024 !     ;
: OPRNDS    ( TRACK --- )
         1000 + SPAC 2 + !          107001 SPAC 4 + !
         1 SPAC 16 + !              20001 120000 SPAC 20 + 2!
         ;
: SET-CBCR    41000 117426 !       100000 CBCR !
         2000 CBCR 2+ !             -1 CBCR 4 + !
         66377 CBCR 6 + !           ;
DECIMAL
```

```
   ( SMS  READ / WRITE)
OCTAL
: TRACKS ( TRACK --- )
         40 OVER + SWAP DO    I OPRNDS R/W    LOOP ;
: READ   ( TRACK --- )
         SET-CBCR    5400 SPAC !    TRACKS ;
: WRITE  ( TRACK --- )
         SET-CBCR    6000 SPAC !    TRACKS ;
: FORMAT ( FORMAT DRIVE 1 )
         SET-CBCR    5217 SPAC !    1000 SPAC 2 + !
         117001 SPAC 4 + !     117 SPAC 16 + !
         R/W    ;
DECIMAL
```

```
   ( IMAGE READ & WRITE )  156 LOAD 157 LOAD
: CHOOSE ( N --- TRACK )
         1 =    IF 1  ELSE 35    THEN    ;
: READ   ( N --- )
         CHOOSE READ ;
: WRITE  ( N --- )
         CHOOSE WRITE ;

EXIT
1 READ   read the first image on drive 1 into channel 3.
2 READ   read the second image on drive 1.
FORMAT   format the diskkete in drive 1 to double density,
         1024 bytes/sector, 8 sector/track format.
1 WRITE  &  2 WRITE  are the reverse of READ commands.
```

RUN LENGTH CODING OF A BINARY IMAGE

Run length coding is a very commonly used technique to compress the image data. It is very effective for black/white type of binary images.

220 LOAD Load in the IP register definitions and READY routine. Several CBCR registers are defined in this block by REG, which is identical to W/O for write-only register

LENGTH Variable holding the pixel length of the current segment under processing.

ADDRESS Variable holding the address of the next available location of the list of run length codes. New code will be added to this address.

COLOR 0 or 1, the color of the current segment.

INIT Initialize the IP to access Ch. 3 image in delta mode, row scanning mode 2. Initialize the run length code list, starting at PAD.

ENCODING THE IMAGE

APPEND Append the run length of the segment just ended to the list of run length codes. If the segment is longer than 255 pixels, break out segments of 255 pixels and add pairs of 255 and 0 to the list of RLC.

LINE Process one row of image. Reduce the pixel length from 512 to 256. The encoded image will be of the 256x256 format.

ENCODE Process 243 alternate rows in the Ch. 3 image. Append at least two zeros to the end of the RLC list as end of record mark.
The run length codes will be generated starting at PAD and the ending address will be in ADDRESS.

DECODING THE RLC CODED IMAGE

RUN Given the length of the next segment on the stack, plot a new segment in Ch. 3. The color of the segment is determined by COLOR.

DECODE Decode the RLC list in PAD to recreate a 256x256 black and white image in Ch. 3.
The decoded image will only occupy a quarter of the normal image display. It can be blown up to fill the display by setting the zoom factor to 2 and scroll the image to fill the display.

```
  219 LIST

( RUN LENGTH CODING, CHT, 27-JAN-83)
: REG ( ADDR --- ) CREATE , DOES> @ ! ;
OCTAL
117600 REG XCBC   117602 REG YCBC   117604 REG BPMR
117606 REG UCLC   120000 CONSTANT IMAGE
117426 REG CSR
VARIABLE LENGTH   VARIABLE ADDRESS   VARIABLE COLOR
: INIT   1000 CSR   100000 XCBC   2000 YCBC   -1 BPMR
         66377 UCLC   PAD ADDRESS !   0 COLOR !   0 LENGTH ! ;
DECIMAL
220 LOAD ( ENCODE )
221 LOAD ( DECODE )
222 LOAD ( PUT TO DISK )
223 LOAD ( RETRIEVE )
224 LOAD ( PICTURE  )


  220 LIST

( APPEND, ENCODE, CHT, 27-JAN-83)
: APPEND   BEGIN   LENGTH @ 255 >   ( LONG SEGMENT? )
           IF   ADDRESS @   255 OVER  C!
                0 OVER 1+ C!   2 + ADDRESS !
                -255 LENGTH +!
           AGAIN
           LENGTH @ ADDRESS @ C!   1 ADDRESS +!
           0 LENGTH !   ;
: LINE     256 0 DO
           IMAGE @ 0=   COLOR @ -   ( NEW SEGMENT?)
           IF   COLOR @ 0= COLOR !   APPEND   THEN
           1 LENGTH +!   LOOP   ;
: ENCODE   INIT   485 0 DO   I YCBC LINE   2 +LOOP   ( 256X256)
           APPEND APPEND ( END OF RECORD)
           ADDRESS @ 1 AND   IF APPEND THEN ; ( EVEN BOUNDARY)


  221 LIST

( RUN, DECODE, CHT, 27-JAN-83)
OCTAL
: RUN   ( BYTE --- )
        COLOR @ IF -1 ELSE 0 THEN   SWAP
        0 DO   DUP IMAGE !   LOOP   DROP   ;
: DECODE   ( RUN THE LIST FROM ADDRESS UNTIL TWO ZEROS)
        INIT 60177 UCLC   177400 BPMR   ( 256X256 FORMAT)
        BEGIN   ADDRESS @ C@   1 ADDRESS +!
                ?DUP   IF RUN 0   ELSE ADDRESS @ C@ 0=   THEN
                COLOR @ 0= COLOR !
        END ;
DECIMAL
```

STORE THE RUN LENGTH CODES ON DISK

>BLOCK Given a memory address and a block number, copy 1024 bytes from the memory to this block and flush it to the disk.

PUT Given a starting block number on the stack, copy the entire run length code list to the disk. The starting address of RLC is PAD and the ending address is in ADDRESS.

RETRIEVE RUN LENGTH CODES FROM DISK

<BLOCK Given the memory address and a block number, copy the block of data, 1024 bytes, to that memory.

RETRIEVE Given a block number and a block count, retrieve that many blocks to the memory area starting at PAD.

After the RLC codes are read in from the disk, one can use the DECODE command to regenerate the image in Ch. 3. However, one has to know exactly where the RLC codes of a picture was stored on the disk and the number of blocks they span. Since the RLC codes are of variable lengths, depending upon the complexity of the original image, I need some tools to manage the picture library.

A LIBRARY OF PICTURES ON DISK

PICTURE A defining word which will create a named entry in the dictionary associated with a picture store on disk. When the name is invoked, the picture will be retrieved from disk and displayed on Ch. 3 of the image processor. When the picture word is defined, two parameters are needed on the stack, the starting block number and the number of blocks of RLC.

P Abbreviation of PICTURE.

The picture definitions are stored in several blocks, which serve well as the library directory. These blocks are loaded and all the picture names are available to be invoked in recalling a picture to the IP display.

```
  222 LIST

( PUT CODE TO DISK, CHT, 28-JAN-83)
: >BLOCK    ( ADDR BLK# --- )
            BLOCK 1024 MOVE UPDATE ;
: PUT       ( BLK# --- , FROM PAD TO ADDRESS MOVED TO DISK )
            ADDRESS @ 1+ PAD -    ( BYTE COUNT )
            1024 / 1+    ( BLK# BLK-COUNT )
            PAD ROT ROT    OVER + SWAP DO
               DUP I >BLOCK    1024 +    LOOP    DROP
            FLUSH ;
```

```
  223 LIST

( RETRIEVE, CHT, 28-JAN-83)
: <BLOCK    ( ADDR BLK# --- )
            BLOCK    SWAP 1024 MOVE ;
: RETRIEVE  ( BLK# BLK-COUNT --- )
            PAD ROT ROT   ( COPY DISK TO PAD )
            OVER + SWAP DO
               DUP I <BLOCK    1024 +
            LOOP DROP    ;
```

```
  224 LIST

( RETRIEVING PICTURES, CHT, 28-JAN-83)
: PICTURE   CREATE , ,   ( BLK# BLK-COUNT)
            DOES> 2@    RETRIEVE
                  INIT    DECODE    ;
( EXAMPLE: 10 2 PICTURE ANIMAL )
```

CATALOG OF THE PICTURE LIBRARY

TICK	Delay for the slow printer.
CATALOG	Print the animal name that follows and also the block number and block length where the encoded picture is stored.
P	Redefine P as CATALOG, so that when the loading block of picture directory is loaded, the names and the locations of the encoded pictures will be printed as a well formated catalog.
P	Redefine P to drop the block number and block length from the stack so that the names of the animals can be executed to display the pictures. This is a very coding and decoding of black-white pictures.

A LIBRARY OF ANIMAL PICTURES

I collected about a hundred pictures of animal drawings and compressed the image using the RLC method. They are stored on a single piece of 8" single density single side diskette. Here is part of the directory. The intended usage of this picture library is educational. If the IP system is setup right, a child of lst grade can type in the name of a animal. If he types the name correctly and if the animal is in the library, he will see a picture of the animal appearing in the display. Otherwise, he will get a ? on the console. This might help him learn the names of many animals. (In this system, he only has to type the first three characters correctly with enough charaters to fill in the length.)

Well, I don't think many school can afford an expensive image processor for this purpose. However, with optical disks comming along, this type of system might be a reality very soon.

```
  225 LIST

( CATALOG, CHT, 7-FEB-83)
: TICK     10000 0 DO LOOP ;
: CATALOG HERE 21 BLANK
          CR 32 WORD 1+ 16 TYPE
          5 U.R 5 U.R   TICK    ;
: P       CATALOG ;

( : P   2DROP ;  )
```

```
  226 LIST

3 2 P AARDVARK 5 4 P ABALONE 9 3 P ADDAX 12 2 P ADDERS 14 2 P
AGOUTI 16 2 P ALBATROSS 18 5 P ALPACA 23 2 P AMOEBA 25 3 P ANEMO
N 28 2 P ANHINGA 30 2 P ANDA 32 2 P ANTEATER 34 2 P ANTLION 36
3 P ARMADILLO 39 1 P ANK 40 1 P AVOCET 42 2 P BADGER 44 2 P
BARNACLE 46 3 P BASS 49 2 P BEE-EATER 51 3 P BIRD-PARADISE
54 2 P BITTEN 56 3 P BLACKBIRD 59 2 P BLACKSNAKE 61 3 P BOA
64 2 P BOAR 66 3 P BOBCAT 69 2 P BRISTLETAIL 71 2 P BUNTING
73 3 P BUZZARD 76 3 P CADDIS-FLY 79 2 P CANARY 81 2 P CARACARA
83 3 P CARDINAL 86 3 P CARIBOU 89 2 P CATBIRD 91 3 P CENTIPEDE
94 2 P CHAMALEON 96 2 P CHAMOIS 98 3 P CHEETAH 101 2 P CHIMPANZE
E 103 2 P CHIPMUNK 105 2 P CHITTON 107 2 P CLAM 109 3 P COBRA
112 2 P COCKATOO 114 2 P COCKLE 116 2 P CONCH 118 3 P CONDOR
121 2 P CONY 123 2 P COOT 125 2 P CORAL 127 2 P CORAL-SNAKE
129 2 P CORMORANT 131 3 P COUGAR 134 4 P CRAB 138 2 P CRANE
140 3 P CRAYFISH 143 4 P CREEPER 147 3 P CROCODILE
150 2 P CUCKOO 152 3 P DINGO 155 2 P DIPPER
```

```
  227 LIST

157 3 P DODO 160 2 P DOVE 162 2 P DUCK 164 2 P EARWIG
166 2 P EGRET 168 2 P ELK 170 3 P EMBILD 173 4 P EMU
177 2 P FLAMINGO 179 2 P FLEA 181 2 P FOX 183 2 P FROGMOUTH
185 2 P GAR 187 2 P GAZELLE 189 3 P GIBBON 192 2 P GILA-MONSTER
195 3 P GMU 198 2 P GRACKLE 200 2 P GRASSHOPPER 202 2 P GROUSE
204 1 P HAGFISH 205 2 P HERON 207 2 P HERRING 209 2 P HYENA
211 2 P IBEX 213 3 P IGUANA 216 2 P JABIRU 218 3 P JACANA
221 2 P JACKAL 223 3 P JAGUAR 226 2 P JAY 228 3 P JELLFISH
231 2 P KINGFISHER 233 2 P KINJOU 235 1 P KIWI 236 2 P KOALA
238 2 P LAMPREY 240 2 P LARK 242 1 P LEECH 243 2 P LEMUR
245 4 P LLAMA
```

CONNECTIVITY ANALYSIS

Connectivity analysis is the technique in isolating and characterizing connected areas in a binary image in which each picture element (pixel) has either a value of 1 or 0. The algorithm was developed by G . J. Agin at SRI International. The original programs were written partly in assembly and partly in BLISS. Rather than translating his programs from these languages into FORTH, I found it more convenient to implement directly from the algorithm itself. My objective was simply implement the very minimum structure and leave most of the embellishments out. They can be added when they are needed.

TECHNICAL TERMS

Segment Descriptor: An array or a block of data that contains at least two items: a column number and a component number.

Active Line: An ordered list of segment descriptors. The current segment pointer points to a segment descriptor on this list. The active segment is the segment descriptor pointed to by the current segment pointer.

Blob Descriptor (Blob): An array or a block of data that contains at least two items: a color which may be either 0 or 1. Additional items in a blob may be used for feature analysis as well. A component number is an index or address which identifies a blob.

TO PROCESS AN IMAGE:

Obtain a blob descriptor to represent the background. Set the color word of the background blob to 0.

Initialize the active line to contain two segment descriptors. The first segment descriptor should have a column number smaller than zero, and a component number pointing to the background blob. The column number of the second segment descriptor should be a large positive number, a number greater than the number of columns in the image. The component number of the second descriptor is immaterial.

Process each row of the image, as described later.

If the image has a fixed number of rows of data, finish by processing an extra row consisting of all zeros.

TO PROCESS A ROW:

Initialize the current segment pointer to point to the first segment descriptor in the active line.
Obtain the run-length representation of the row. The run-length data should start with a negative number and end with a large positive number.
For every pair of adjacent numbers in the run-length data, in turn, call the segment-processing operation.
While the current segment pointer does not point to the last segment in the active line, perform the deletion operation. Repeat this step until the current segment pointer does point to the last segment.

TO PROCESS A SEGMENT, given a starting column number and an ending column number:

While the starting column number is greater than or equal to the column number of the segment desriptor after the current segment in the active line, do the deletion operation.
If the ending column number is less than the column number of the current segment, perform the insertion operation, passing on the starting and ending numbers of the segment.
If feature extract is being done, do additional case 3 processing now.
Copy the starting column number to the column number of the current segment.
Advance the current segment pointer to point to the next segment descriptor in the active line.

INSERTION OPERATION, given starting and ending column numbers:

Obtain the component number of the segment descriptor preceding the current segment. Call that component the surrounding component.
Obtain a new blob descriptor. Call it the new component. Set the color word of the new component to the opposite of the color of the surrounding component.
Obtain two new segment descriptors and insert them in the active line immediately before the current segment, calling the first segment "A" and the second "B". Segment A receives the new component number and the starting column number. Segment B receives the surrounding component number, and the ending column numbers.
If feature extraction is being done, perform additional case 2 processing as required.

DELETION OPERATION:

Call the component number of the current segment the terminated component. Call the component number of the segment preceding the current one the left component and the component number of the segment following the current one the right component. Call the left component the replacing component and the right component the replaced component. If the right component points to the background, then call it the replacing component and the left component the replaced component, otherwise call the left component replacing and the right component replaced.

If feature extraction is being performed, do additional case 1 processing as required.

DELETION OPERATION (Continued):

If the replacing component and the replaced component are different, then find all instances of the replaced component number in the active line, and change them to the replacing number.

Delete the current component and the segment following the current one from the active line. Let the current segment pointer point to the first segment after the deleted one.

Search for instances of the terminated component number in the active line. If there are no remaining instances, call application-dependent subroutines, as appropriate, passing the address of the terminated component's blob descriptor.

REGISTER DEFINITIONS AND INITIALIZATION

CSR Control-Status register of the De Anza Image Proc'r.
CBCR Image memory access control register.
IMAGE Base address of the image data memory.
ACTLINE Base address of the active line under analysis.
RUNCODE Starting address of the run-length codes of an image.
TBLOB Base address of the terminated blob descriptor table.
ABLOB Address of memory storing active blob descriptor.
ABLOBS Pointer to the active blob descriptor.
TBLOBS Pointer to the current terminated blob descriptor.
ACTSEG Pointer to the active segment in the active line.
RLC-INIT Initialize image processor to derive run-length codes from an image stored in the image memory.
INIT-CONN Initialize image processor to do connectivity analysis
TEST Test the data moving mechanism of the image processor.

RUN-LENGTH CODES FROM IMAGE

ONE-LINE (row ---) Process one row of image data and generate one line of run-length code, given the row number on top of the stack.
Image data is accessed through IMAGE register and the run-length codes are stored via RUNCODE register.

RUN-LENGTH-CODE
Scan through 485 lines of image and produce the same number rows or run-length codes needed for connectivity analysis.

CLEAR Clear the memory used to store run-length codes to all zeros.

CONNECTIVITY ANALYSIS

INITIALIZE
Initialize all the image processor registers and memory pointers for connectivity analysis. The first active line is also initialized.

IMAGE-PROCESS
The highest level instruction to perform a complete connectivity analysis.

```
     60 LIST

(   CBCR INITIALIZATION, 2-3-81) OCTAL
: IS CONSTANT ;
117426 IS CSR   117600 IS CBCR   120000 IS IMAGE
120000 IS ACTLINE   122000 IS RUNCODE   124000 IS TBLOB
130000 IS ABLOB
VARIABLE ABLOBS   VARIABLE TBLOBS   VARIABLE ACTSEG
: CBCR+!   CBCR + 2! ;
: RLC-INIT   0 100000 0 CBCR+!   66777 0 4 CBCR+!
    0 100400 10 CBCR+!   66777 0 14 CBCR+!
    0 100400 30 CBCR+!   20377 377 34 CBCR+!
    100000 CSR !   0 100000 20 CBCR+!   20377 377 24 CBCR+! ;
: INIT-CONN   20 0 DO   20377 -1 I 10 * 4 + CBCR+!
    400 I + 40000 I 10 * CBCR+!   LOOP
    0 40400 30 CBCR+!   100000 CSR !   ;
: TEST   DO IMAGE I + @ RUNCODE I + ! LOOP ;
DECIMAL


   61 LIST

(   RUN LENGTH CODE, CHT, 2-2-81)   OCTAL
: ONE-LINE   ( Y ---)
    DUP 100000 0 CBCR+!   DUP 100400 10 CBCR+!
    2/ DUP 100000 20 CBCR+!   100400 30 CBCR+!
    RUNCODE 0 OVER !   2+ 1   ( RLC BACKGROUND-COLOR)
    1777 4 DO
      IMAGE I + @ 0=   OVER - IF ( TRANSITION)
      NOT ( TOGGLE COLOR) SWAP   I 2/ 2/ OVER !   2+ SWAP   THEN
    4 +LOOP
    NOT IF ( LAST COLOR NOT BKGD)   376 OVER !   2+   THEN
    377 SWAP ! ;
DECIMAL
: RUN-LENGTH-CODE   CLEAR   RLC-INIT
    485 0 DO   I ONE-LINE   2 +LOOP ;


   62 LIST

(   IMAGE PROCESSING, CHT, 2-4-81)   OCTAL
: INITIALIZE   ABLOB ABLOBS !   100000 ABLOBS @ ! ( BKGD COLO)
      TBLOB TBLOBS !   ACTLINE 0 OVER !   ABLOB OVER 2+ !
      377 OVER 4 + !   0 SWAP 6 + !   100000 CSR !   ;
DECIMAL
: IMAGE-PROCESS   INIT-CONN   INITIALIZE
      243 0 DO   CR I . I ROW-PROCESS   LOOP   ;
```

ROW PROCESSING

ZZ Delete all segments until the last column. This is used at the end of the active line.

ROW-PROCESS (row ---)
Process one row of run-length codes whose row number is given on the stack.

ROW Alias of ROW-PROCESS. Save lots of typing during testing.

SEGMENT PROCESSING

SEGMENT-PROCESSING (start-column end-column ---)
With the starting column number of the current segment and the starting column number of the next segment on stack, compare the current segment with the active segment in the active line. If the two segments overlap, extend the current blob descriptor to include the current segment. If the current segment leads the active segment, create a new blob descriptor. If the current segment lags behind the active segment, terminate the current blob descriptor.

S Alias of SEGMENT-PROCESSING.

INSERTION

INSERTION (starting-column end-column ---)
Insert the current segment into the active line. Get the next blob descriptor and create a new blob for this segment.

```
   63 LIST

(   ROW PROCESSING, CHT, 2-4-81)    OCTAL
: ZZ   BEGIN   ACTSEG @ DUP @   377 < SWAP 4 + @ 377 <   AND
     IF   DELETION   AGAIN ;
: ROW-PROCESS   ( ROW# ---)
     ACTLINE ACTSEG !   DUP CBCR 32 + !  CBCR 22 + !    ( YCBCR)
     RUNCODE    ( BASE ADDR OF CURRENT RLC)
     BEGIN
        DUP @ ( START-COL)   OVER 2+ @ ( END-COL)
        DUP IF ( IF END OF SEGMENT, EXIT)
            SEGMENT-PROCESSING
            2+   ( INCREMENT RUNCODE)
     AGAIN    DROP 2DROP
     RUNCODE 2+ @ 377 = IF ZZ THEN  ( BLANK LINE, CLOSE BLOB)
     ;
: ROW ROW-PROCESS  ;
DECIMAL


   64 LIST

(   SEGMENT PROCESSING, CHT, 2-4-81)   OCTAL
: SEGMENT-PROCESSING   ( RLC START-COLUMN END-COLUMN --- RLC')
     ACTSEG @ >R
     I @   377 -   ( ACTSEG=377 SKIP DELETION)
     IF   OVER I 4 + @ < NOT   ELSE 0   THEN
     IF DELETION 2DROP 2 - ( RUNCODE MUST STAY THE SAME)
     ELSE   DUP I @  <
            IF 2DUP INSERTION 2CASE   2DROP
            ELSE   EXTENSION DROP I !
            THEN
            4 ACTSEG +!   ( MOVE POINTER TO NEXT SEGMENT)
     THEN   R> DROP
     ;
: S SEGMENT-PROCESSING ;
DECIMAL



   65 LIST

(   INSERTION, CHT,2-4-81)   OCTAL
: INSERTION   ( START-COLUMN END-COLUMN ---)
     ACTSEG @ 2 - @ ( SURR-COMP)
     SWAP OVER C@   ( S-COL SUR-COMP E-COL CL)
     NOT 100000 +   NEXTBLOB   !   ( SET COLOR)
     ROT ABLOBS @   SWAP ( SURR END NEW START )
     INSERT-SEGMENTS
     ;
DECIMAL
```

NEXTBLOB

NEXTBLOB (--- addr, address of next blob descriptor)
Return the address of the next available blob descriptor. Abort if the active blob area is exhausted.

INSERT-SEGMENT

SEG Temporary storage space to hold the active segment during the insertion operations.

INSERT-SEGMENT (surround-component end-column
new-component starting column ---)
Insert the current segment into the active line and update the active segment.

DELETION

?DELETE (---)
Examine the current segment. If either the starting-column or the end-column is 377 octal, abort the connectivity analysis because it has reached the last segment.

DELETION (---)
Delete the current active segment from the active line. Terminate the current active blob and move its statistics to a new terminal blob if this component does not appear in the rest of the active line. In this case, a closed area is isolated and its statistics are recorded in the TBLOB area.

```
   66 LIST

(   NEXTBLOB, CHT, 2-3-81)    OCTAL
: NEXTBLOB    (  --- ABLOB-ADDR, FIND NEXT FREE ABLOB)
      ABLOBS @    DUP
      BEGIN
         DUP 137760 < NOT ( END OF FREE ABLOB SPACE?)
         IF DROP 130000    ELSE 20 +    THEN
         2DUP =    ABORT" ABLOB EXHAUSTED!"
         DUP @ 0<    NOT    ( BLOB NOT IN USE)
      END
      DUP ABLOBS !    SWAP DROP
     ;
DECIMAL
```

```
   67 LIST

(   INSERT-SEGMENTS, CHT,2-3-81)    OCTAL
VARIABLE SEG 10 ALLOT
: INSERT-SEGMENTS ( SURR-COMP END-COL NEW-COMP START-COL ---)
      ACTSEG @
      BEGIN
        >R
        I SEG 10 MOVE ( SAVE ACTIVE SEGMENTS)
        I 2!    I 4 + 2!    ( INSERT SEGMENTS)
        SEG 4 + 2@    SEG 2@    ( PUSH OLD SEGS TO STACK)
        SEG DUP @ 377 = SWAP 4 + @ 377 = OR
        R> 10 + SWAP
      END
      DUP >R 2!    R> 4 + 2!    ;
DECIMAL
```

```
   68 LIST

(   DELETION, CHT, 2-3-81)  OCTAL
: ?DELETE    ACTSEG @ >R    I @ DUP 0= SWAP 377 = OR
      R> 4 + @ DUP    0= SWAP 377 =  OR OR
      IF    CBCR 22 + @ .    ABORT" DELETION ERROR " THEN ;
: DELETION    ( --- )    ?DELETE
      ACTSEG @    DUP 2+ @ SWAP    ( TERM-COMP ACTSEG)
      DUP 2 - @ ( LEFT-COMP)    SWAP 6 + @ ( RIGHT-COMP)
      DUP ABLOB = ( BKGD BLOB?)    IF SWAP THEN
      ( TERM-COMP REPLACING-COMP REPLACED-COMP)
      lCASE
      REPLACE-COMPONENTS
      DELETE-TWO-SEGMENTS
      TERMINATE
      ;
DECIMAL
```

CASE 3, EXTEND BLOB

3CASE (start-column end-column --- start-col end-col)
Extend the current blob descriptor to inclde the statistics of the current segment.

EXTENSION Alias of 3CASE.

CASE 2, INSERT BLOB

2CASE (start-col end-col --- start-col end-col)
Initialize the newly created blob descriptor with the statistics of the current active segment.

1CASE

1CASE (---)
Store the current row number, which is available in one of the CBCR registers, into the current active blob descriptor.

MERGE (left-component right-component ---
left-component right-component)
Merge the statistic of the right segment into the blob descriptor of the left component.

```
   69 LIST

(   CASE 3, EXTEND BLOB, CHT, 2-3-81)
: 3CASE   ( START-COL END-COL)
     ACTSEG @ 2+ @ >R    ( COMP)
     2DUP SWAP - I 2+ +!    ( AREA)
     OVER I 4 +   DUP >R @    MIN R> !    ( X-MIN)
     DUP R>   6 + DUP >R @    MAX R> !    ( X-MAX)
     ;
: EXTENSION 3CASE ;
```

```
   70 LIST

(   CASE 2, INSERT BLOB, CHT, 2-3-81)  OCTAL
: 2CASE   ( START-COL END-COL)
     ACTSEG @ 2+ @ >R    ( COMP)
     2DUP SWAP - I 2+ !     ( AREA)
     OVER I 4 + !      ( X-MIN)
     DUP I 6 + !      ( X-MAX)
     CBCR 22 + @   R> 10 + !    ( Y-MIN)
     ;
DECIMAL
```

```
   71 LIST

(   CASE 1, DELETE BLOB, CHT, 2-3-81)   OCTAL
: 1CASE   ( --- )
     CBCR 22 + @    ( Y-MAX)
     ACTSEG @ 2+ @ 12 +  !
     ;
: MERGE   ( REPLACING-COMP REPLACED-COMP)
     >R
     I 2+ @ OVER 2+ +!          ( AREA)
     I 4 + @   OVER 4 + DUP @   ROT MIN SWAP !      ( X-MIN)
     I 6 + @   OVER 6 + DUP @   ROT MAX SWAP !      ( X-MAX)
     I 10 + @  OVER 10 + DUP @  ROT MIN SWAP !      ( Y-MIN)
     I 12 + @  OVER 12 + DUP @  ROT MAX SWAP !
     ( I 20 ERASE )
     R> ;
DECIMAL
```

REPLACE-COMPONENTS

REPLACE-COMPONENTS (replacing-comp replaced-comp ---)
Merge the statistics of the replaced component to that of the replacing component. Search through the active line for the replaced component nubmers and substitute them with the replacing component number.

DELETE TWO SEGMENTS

DELETE-TWO-SEGMENTS (---)
Delete the active segments in the active line. Move the rest of the segments to fill up the gap.

TERMINATE

TERMINATE (terminal-component ---)
Terminate a blob descriptor. Move the blob statistics in ABLOB to TBLOB for storage.

```
   72 LIST

(   REPLACE-COMPONENTS, CHT, 2-3-81)
: REPLACE-COMPONENTS    ( REPLACING-COMP REPLACED-COMP ---)
     2DUP - IF    ( NOT THE SAME COMP)
        MERGE
        ACTLINE
        BEGIN     ( REPLACING)
           2+ DUP @ ( COMP)    DUP
        IF ( EXIT IF COMP=0)
           >R OVER R> = IF    ( COMP=REPLACED-COMP)
                     >R OVER I ! R>    THEN
           2+     ( UPDATE ACTLINE)
        AGAIN
        DROP
     THEN
     2DROP ;



   73 LIST

(   DELETE-TWO-SEGMENTS, CHT, 2-3-91)    OCTAL
: DELETE-TWO-SEGMENTS    ( --- )
     ACTSEG @
     BEGIN
        DUP 10 + OVER    10 MOVE
        DUP @ 377 =    OVER 4 + @ 377 =    OR    ( ANY ZERO COMP?)
        SWAP 10 + SWAP
     END
     DROP ;
DECIMAL









   74 LIST

(   TERMINATE, CHT, 2-3-81)    OCTAL
: TERMINATE    ( TERM-COMP ---)
     >R    0 ( END OF LINE FLAG)    ACTLINE
     BEGIN        DUP @ 377 - IF ( EXIT AT LAST SEGMENT )
        2+ DUP @     ( FLAG COMP-PTR COMP)
        I =     ROT  +     ( UPDATE END FLAG)
        SWAP 2+   ( FLAG PTR )
     AGAIN       DROP
     NOT ( FLAG IS ZERO IF NO TERM-COMP REMAINS IN ACTLINE)
     IF    TBLOBS @ ABLOB > ABORT" TBLOB EXHAUSTED!"
           I TBLOBS @ 20 MOVE    ( SAVE ABLOB TO TBLOB)
           TBLOBS @ 14 DUMP    I 20 ERASE
           20 TBLOBS +!    ( FREE SPACE FOR TERMINATED BLOBS)
     THEN
     R> DROP ;
DECIMAL
```

```
   78 LIST

(    SOME DEBUGGING TOOLS, CHT, 2-11-81)
: DELAY 2000 DO LOOP ;
: TDUMP DO I 16 * TBLOB + 16 DUMP DELAY LOOP ;
```

```
   79 LIST

(    TEST RUN LENGTH CODES, CHT, 2-5-81)    OCTAL
: RLC ( CODES-IN-REVERSE-ORDER N Y --- )
     100000 20 CBCR+!
     0 DO    RUNCODE I 2* + !    LOOP ;
: Y ( Y ---)   DUP CBCR 22 + !     CBCR 32 + ! ;
DECIMAL
: TEST-IMAGE
    255 7 4 0 4 0 RLC
    255 8 5 3 1 0   6 1 RLC
    255 9 7 6 4 3 2 0   8 2 RLC
    255 9 8 6 2 0   6 3 RLC
    255 9 8 5 2 0   6 4 RLC
    255 9 7 6 4 3 2 0   8 5 RLC
    255 8 5 3 2 0   6 6 RLC
    255 7 6 0   4 7 RLC
    255 0   2 8 RLC   ;
```

```
   80 LIST

( CONNECTIVITY ANALYSIS LOAD BLOCK, CHT, 2-4-81)
: THRU   1+ SWAP DO I LOAD LOOP ;
: CLEAR ;
60 61 THRU
69 74 THRU
66 68 THRU
65 LOAD ( INSERTION)
64 LOAD ( SEGMENT)
63 LOAD ( ROW)
62 LOAD ( IMAGE)
79 LOAD ( TEST RLC)
: INIT INIT-CONN   INITIALIZE   0 CBCR 18 + ! ;
: TEST INIT TEST-IMAGE INIT ;
INIT
: .RLC DO CR I . CR I Y RUNCODE 100 DUMP LOOP ;
```

A SIMPLE GRAPHICS SYSTEM

This simple graphics system is a very good demonstration unit if you have a dot addressable graphic display in your computer. It has commands to draw lines, rectangles, circles and arrows on the CRT monitor. It can be easily expanded into a full graphics system for entertainment or serious purposes.

Block 123 is the load block with three demonstration words:

CIRCLES	Draw a set of concentric circles.
LINES	Draw a set of straight lines going through the center of CRT.
RECTS	Draw a set of rectangles.

8* , 8/ , 64* , 64/	Multiply or divide by shifting.
PAINT	The only machine dependent code. It takes the x,y coordinate pair and paint a white dot on CRT screen. It assumes a 512x512 display.
CALU	(x1 y1 x2 y2 --- 64*slope x2 x1) Given coordinates of two points, leave on the stack (y2-y1), 64*y1, x2, and x1. They will be used by XVEC
XVEC	(x1 y1 x2 y2 ---) Draw a line between two points along the x axis. x2 must be greater than x1.
YCALU	(x1 y1 x2 y2 --- 64/slope y2 y1) Similar to CALU. Leave parameters used in YVEC.
YVEC	(x1 y1 x2 y2 ---) Draw a line between two points incrementing along the y axis.
MODE	Specify point addressing mode to the graphic system hardware.

XPREV, YPREV	Variables containing position of the last plotted position.
?RANGE	(x1 y1 x2 y2 --- x1 y1 x2 y2) Check the ranges of all coordinates, Abort if any one exceeds 511.
VECTOR	(x1 y1 x2 y2 ---) Draw a line between two points. It does range checking and draws a densely spaced line
SIDES	(x1 y1 x2 y2 ---) Draw two opposing sided of a rectangle.
RECTANGLE	(x1 y1 x2 y2 ---) Draw a complete rectangle.
MOVE	(x y ---) Store the x,y pair in XPREV, YPREV.
LINE	(x y ---) Continue aline from previous point.
TEST	Test the line drawing words.

```
   81 LIST

( GRAPHICS DEMONSTRATIONS, 1-13-81, CHT)
EMPTY 201 LOAD 87 LOAD
82 LOAD 83 LOAD 84 LOAD 85 LOAD 86 LOAD

: CIRCLES 256 256 25 0 DO 2DUP I 10 * CIRCLE LOOP ;
: LINES 255 255 511 0 DO 2DUP 0 I VECTOR 2DUP I 0 VECTOR
     511 I 2OVER VECTOR I 511 2OVER VECTOR 10 +LOOP 2DROP ;
: RECTS 256 0 DO I I 511 I - DUP RECTANGLE 10 +LOOP ;
: PATTERNS 0 8 1 DO 8 1 DO I 64 * OVER J 64 * SWAP BULLEYE
     1+ 15 AND LOOP LOOP DROP ;
```

```
   82 LIST

(  GRAPHICS, 1-11-81, CHT)
CODE 8* S ) ASL   S ) ASL   S ) ASL   NEXT
CODE 8/ S ) ASR   S ) ASR   S ) ASR   NEXT
: 64* 8* 8* ;   : 64/ 8/ 8/ ;  ( X Y PAINT ---)
CODE PAINT CBCR 2+ S )+ MOV CBCR S )+ MOV IMAGE -1 # MOV NEXT
: CALU ( X,Y -- 64H Y1 X12)  SWAP >R ROT   I OVER - SWAP >R >R
     64* 0 ROT 64* 0   2OVER 2OVER D- R> M/  >R
     2SWAP 2DROP DROP   R> SWAP   R> R> SWAP ;
: XVEC ( X1 Y1 X2 Y2 --  , X2 > X1 )   2DUP PAINT CALU DO
     I OVER 64/ PAINT   OVER + LOOP   DROP DROP ;
: YCALU ( X,Y--64H X1 Y12)  >R SWAP I OVER -   SWAP >R >R
     64* >R 64* 0   R> 0   2OVER D-   R> M/
     SWAP DROP SWAP   R> R> SWAP ;
: YVEC ( X1 Y1 X2 Y2 --  ,  Y2 > Y1 )   2DUP PAINT
    YCALU DO DUP 64/ I PAINT   OVER + LOOP   DROP DROP ;
: MODE   0 CSR !   -256 CBCR 4 + !   24576 CBCR 6 + ! ;
```

```
   83 LIST

(   VECTOR, LINE, RECTANGLE, 1-11-81, CHT)
 VARIABLE XPREV   VARIABLE YPREV
: ?RANGE 2OVER 2OVER 4 0 DO 511 > ABORT"  ?RANGE" LOOP ;
: VECTOR ( X1 Y1 X2 Y2 ---)  ?RANGE
    >R SWAP >R ( X1 X2 --)   OVER OVER - ABS R> DUP I - ABS
    SWAP >R   > IF   ( DX > DY, USE XVEC)
       OVER OVER >   IF SWAP R> R> ROT ROT   ELSE R> SWAP R>
                     THEN XVEC
       ELSE R> DUP I >   IF SWAP R> 2SWAP   ELSE SWAP R>
                     THEN YVEC              THEN ;
: SIDES   2OVER 2OVER DROP OVER VECTOR
          2OVER 2OVER >R DROP OVER R> VECTOR ;
: RECTANGLE  SIDES 2SWAP SIDES ;
: MOVE ( X Y -- )   YPREV ! XPREV ! ;
: LINE ( X Y -- ) 2DUP XPREV @ YPREV @ VECTOR MOVE ;
: TEST 512 0 DO 0 I 511 I VECTOR LOOP ;
```

ROOT	Calculate the square root of the double number using the seed also provided on the stack.
SEGMENT	Given two sides of a right triangle, return the length of the third side or the cord length.
CORNERS	Paint the corners of a rectangle, given the center of the rectangle and the half values of its sides.
CIRCLE	Draw a full circle with its center and radius of the circle. Eight points on the circle are generated and plotted at once.

XSLOPE	Leave the slope of the line between two points on the stack.
YSLOPE	Leave the inverse of the slope between two points.
XARROW	Draw a line between two point, incrementing in the x direction. Both ends are decorated with arrow heads pointing outwards.
YARROW	Similar to XARROW but incrementing in the y-direction.

ARROW	Smart arrow drawing routine. It checks ranges and also select the denser drawing paths.
BULLEYE	A demonstration word to draw several common shapes needed in engineering drawings.

84 LIST

```
(   CIRCLE, ROOT, SEGMENT, CORNERS.  1-13-81, CHT)
: ROOT ( D N1 -- N2,  D: SQUARE, N1: SEED, N2: ROOT)
    3 0 DO >R 2DUP I M/ R> + 2/ LOOP >R 2DROP R> ;
: SEGMENT ( N1 N2 -- N3 ,  N3=SQUARE-ROOT'N1**2-N2**2' )
    2DUP = IF 2DROP 0
    ELSE   2DUP SWAP 2/ < ( IF N2 < N1/2, SEED=N1 )
         IF    2DUP 2/ 2/ -
         ELSE  2DUP - 2*  ( N2>N1/2, SEED='N1-N2'*2  )
         THEN  >R DUP M* ROT DUP M* 2SWAP D- R> ROOT
    THEN ;
: CORNERS ( DX DY X Y --)
    2OVER 2OVER D+ PAINT   2OVER 2OVER 2SWAP D- PAINT
    2OVER MINUS 2OVER D+ PAINT   2SWAP MINUS D- PAINT ;
: CIRCLE ( X Y R --)   DUP 3 4 */ 1+ 0 DO   DUP I SEGMENT
    2OVER 2OVER SWAP DROP I   2DUP >R >R   2OVER CORNERS
    R> R> SWAP 2SWAP CORNERS   DROP LOOP   2DROP DROP ;
```

85 LIST

```
(   ARROWS, 1-12-81, CHT)
: XSLOPE ( X1 Y1 X2 Y2 -- SLOPE*64 )        CALU 2DROP DROP ;
: YSLOPE ( X1 Y1 X2 Y2 -- YSLOPE*64 )      YCALU 2DROP DROP ;
: XARROW ( X1 Y1 X2 Y2 --   , X2 > X1 )
    2OVER 2OVER XSLOPE   DUP 20 + 8/ >R   20 - 8/ >R
    OVER 8 -   OVER I -   2OVER VECTOR
    2OVER OVER 8 +   OVER R> +   VECTOR
    OVER 8 -   OVER I -   VECTOR
    OVER 8 +   OVER R> + VECTOR ;
: YARROW ( X1 Y1 X2 Y2 --   ,  Y2 > Y1 )
    2OVER 2OVER YSLOPE   DUP 20 + 8/ >R   20 - 8/ >R
    OVER I -   OVER 8 -   2OVER VECTOR
    2OVER   OVER R> +   OVER 8 +   VECTOR
    OVER I -   OVER 8 -   VECTOR
    OVER R> +   OVER 8 +   VECTOR ;
```

86 LIST

```
(   ARROW, BULLEYE, 1-12-81, CHT)
: ARROW ( X1 Y1 X2 Y2 ---)  ?RANGE
    >R SWAP >R ( X1 X2 --)   OVER OVER - ABS R> DUP I - ABS
    SWAP >R   > IF   ( DX > DY, USE XVEC)
       OVER OVER >   IF SWAP R> R> ROT ROT   ELSE R> SWAP R>
                     THEN  2OVER 2OVER XARROW XVEC
       ELSE R> DUP I >   IF SWAP R> 2SWAP   ELSE SWAP R>
                     THEN   2OVER 2OVER YARROW YVEC
    THEN ;
: BULLEYE ( X Y R --)   >R
    OVER I - I 2/ 2/ - OVER MOVE   OVER I 2/ - OVER LINE
    OVER I 2/ 2/ - OVER MOVE       OVER I 2/ 2/ + OVER LINE
    OVER I 2/ + OVER MOVE          OVER I + I 2/ 2/ + OVER LINE
    2DUP I - I 2/ 2/ - MOVE        2DUP I 2/ - LINE
    2DUP I 2/ 2/ - MOVE            2DUP I 2/ 2/ + LINE
    2DUP I 2/ + MOVE   2DUP I + I 2/ 2/ + LINE   R> CIRCLE ;
```

ACCURACY IN LINE DRAWING

The line drawing routine is limited in it accuracy due to the scaling factor of 64. A line drawn across the screen may accumulate a error up to 7 pixels at the end. The advantage was that points are calculated using only additions. If speed must be traded for accuracy, points must be calculated using intrapolation with multiplications and divisions.

SLOPE Given two point coordinates on the stack, return the slope in two numbers, the numerator and the denominator. Ratios can thus be calculated more precisely.

OFFSET Given a pair of coordinate and a pair of slope ratio, return the intersect between this line and the Y axis.

COEFFICIENTS From the coordinates on stack, calculate the slope and offset and store them in memory for later uses.

REVISED LINE DRAWING ROUTINE

The line drawing rouitnes in the graphic package can be rewritten to use the interapolation rouitne. The accuracy should be in the order of 1 pixel. However, the speed of drawing is slowed down considerably. The intrapolation routine can be optimized further to reduce some of the memory shufflings, but one */ per point will still be a significant factor in the computations.

```
( LINEAR INTRAPOLATION, 29-JUL-83, CHT)

: SLOPE    ( X1 Y1 X2 Y2 --- NUMERATOR DENOMINATOR)
           ROT - >R   SWAP - R>   SWAP   ;
: OFFSET   ( Y1 X1 NUMERATOR DENOMINATOR --- OFFSET )
           */ -   ;
: COEFFICIENTS   ( X1 Y1 X2 Y2 ADDR --- )
           >R   2OVER SLOPE   2SWAP SWAP 2OVER OFFSET
           I 4 + !   R> 2!   ;
: COMPUTE   ( X ADDR --- Y )
           >R   I 2@ */   R> 4 + @ +   ;
```

```
( LINE DRAWING, 29-JUL-83, CHT)
VARIABLE COEF 4 ALLOT
: CALU     ( X1 Y1 X2 Y2 --- )   COEF   COEFFICIENTS   ;
: XVEC     ( X1 Y1 X2 Y2 --- , X2 > X1 )
           2OVER 2OVER CALU   DROP SWAP DROP   SWAP DO
           I DUP COEF COMPUTE   PAINT   LOOP   ;

: YCALU    ( X1 X2 Y1 Y2 --- )   SWAP 2SWAP SWAP 2SWAP
           COEF COEFFICIENTS   ;
: YVEC     ( X1 Y1 X2 Y2 --- , Y2 > Y1 )
           2OVER 2OVER YCALU   SWAP DROP ROT DROP SWAP DO
           I COEF COMPUTE   I   PAINT   LOOP   ;
```

```
( GRAPHICS DEMONSTRATIONS, 1-13-81, CHT)
EMPTY 201 LOAD 87 LOAD
82 LOAD 442 LOAD 443 LOAD
83 LOAD 84 LOAD 85 LOAD 86 LOAD

: CIRCLES 256 256 25 0 DO 2DUP I 10 * CIRCLE LOOP ;
: LINES 255 255 511 0 DO 2DUP 0 I VECTOR 2DUP I 0 VECTOR
     511 I 2OVER VECTOR I 511 2OVER VECTOR 10 +LOOP 2DROP ;
: RECTS 256 0 DO I I 511 I - DUP RECTANGLE 10 +LOOP ;
: PATTERNS 0 8 1 DO 8 1 DO I 64 * OVER J 64 * SWAP BULLEYE
     1+ 15 AND LOOP LOOP DROP ;
```