# A SOFTWARE SYSTEM FOR THE HONEYWELL OCULOMETER

by

MICHAEL TERRY ERRECART

Submitted in Partial Fulfillment

of the Requirements for the

Degree of Bachelor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1972

Signature redacted

ature of Author . . . . . Department of Electrical Engineering

Signature redacted

ified by . . . . . Thesis Supervisor

Signature redacted

pted by . . . . . Chairman, Departmental Committee of Theses

The Honeywell Oculometer is an instrument capable of determining the direction in which a person is looking through a remote camera. The Oculometer utilizes a digital computer which processes optical data.

This report describes and analyzes a software system implemented on the digital computer used in the Oculometer. It gives an overview of the entire Oculometer, a thorough analysis of the design decisions involved in creating the software, and a complete documentation of the software implemented on the system.

TABLE OF CONTENTS

INTRODUCTION.....4

PART I. .....4

A. CAMERA.....6

B. COMPARATORS.....9

C. DIGITAL INTERFACE SYSTEM.....9

D. H112 COMPUTER.....12

PART II.....14

A. SYSTEM.....16

B. PROCESSING.....23

1. Data Analysis.....23

2. Computation Analysis.....32

3. Processing sequence.....42

C. OUTPUT CONTROL.....45

D. CALIBRATION.....54

APPENDIX A. SOFTWARE LISTING.....61

# INTRODUCTION

This report is a summary and analysis of the software system I developed at Honeywell Radiation Center on the Oculometer Project. It is intended to provide a comprehensive overview of the oculometer, a thorough analysis of the constraints upon the software subsystem and of the software itself, and a complete, detailed documentation of the software system.

The project was headed by John Merchant, who first pioneered the oculometer concept. Dick Morrissette was responsible for maintaining the hardware systems. I was responsible for designing and maintaining the software systems. The project was an internal development program.

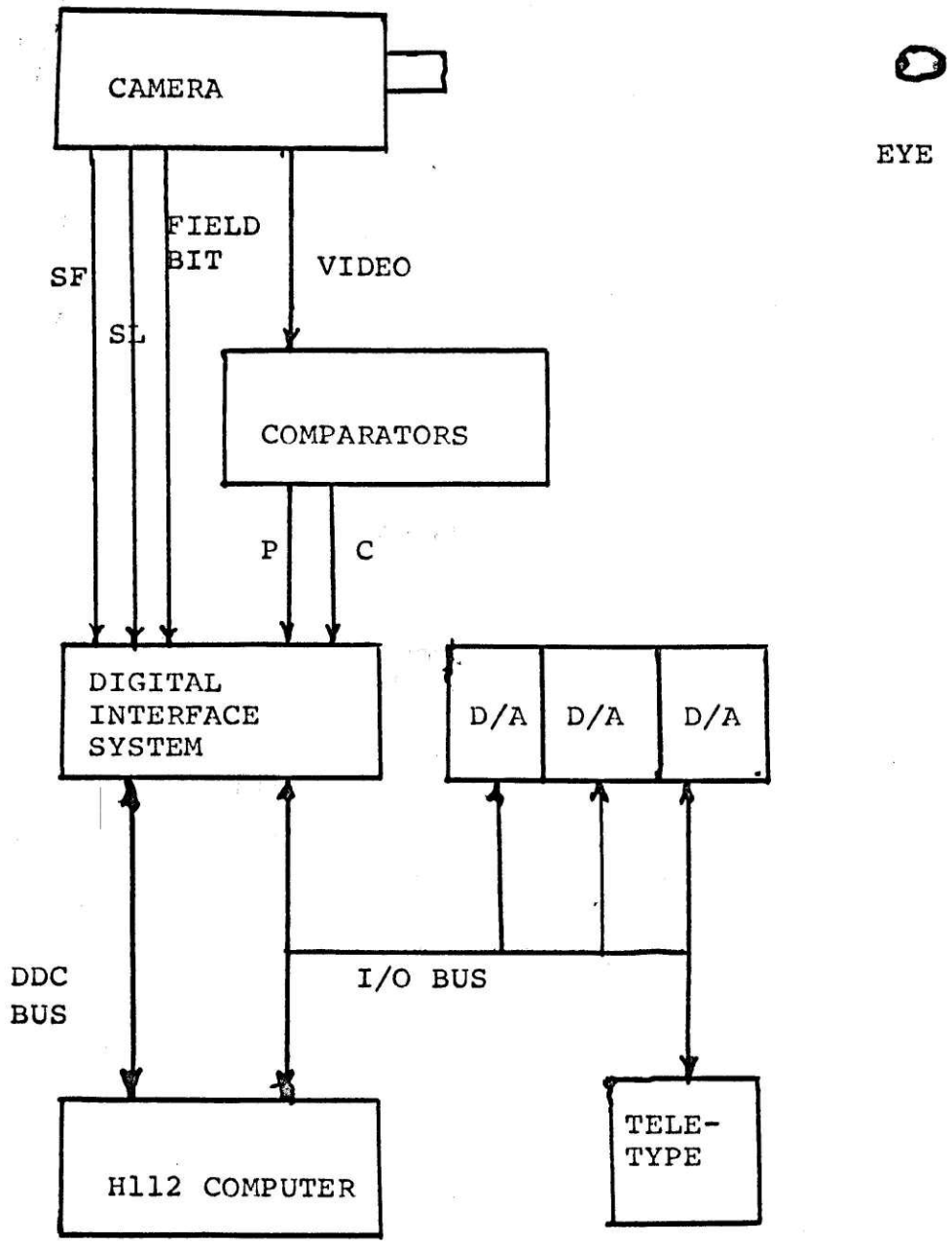
The purpose of this project was to produce an oculometer whose components were readily reproducible subsystems. Previous oculometers had been constructed of special purpose analog circuits, of a complexity not readily reproducible. This project developed an oculometer in which digital systems performed the bulk of the data processing operations.

Part I of this report details the hardware environment which constrains the software system. Part II analyzes and documents the software system I developed.

# PART I.

Briefly speaking, the oculometer principle is this: a

OCULOMETER SYSTEM



light shone into a person's eye produces a corneal highlight. The relative position of this highlight to the center of the person's pupil gives an accurate indication of the direction in which the person is looking. Oculometers have been built with errors of less than ½o if eye rotation.

The oculometer is composed of four major subsystems, plus associated input/output devices. These subsystems are the camera, the comparator circuitry, the Digital Interface System, and the H112 computer. Peripheral devices include three digital to analog converters, a television monitor, an x-y oscilloscope, and an ASR-33 teletype.

The systems operates in the following manner: The camera is focused upon an eye. Its output video signal triggers pupil and cornea comparators. The Digital Interface System assigns x,y coordinates to each transition of a comparator, building up a file in the H112 memory containing the coordinates of the outline of the pupil and the cornea. The H112 determines the centers of the pupil and cornea and the relative positions of these centers, outputting the results upon analog channels.

This part is divided into four sections, one for each major subsystem. The subsystems are covered in the order that data would propagate through the system.

## A.

A stationary camera with light source is directed towards an eye which is positioned directly in front of the camera and approximately 33 inches away ( This distance can be altered.).

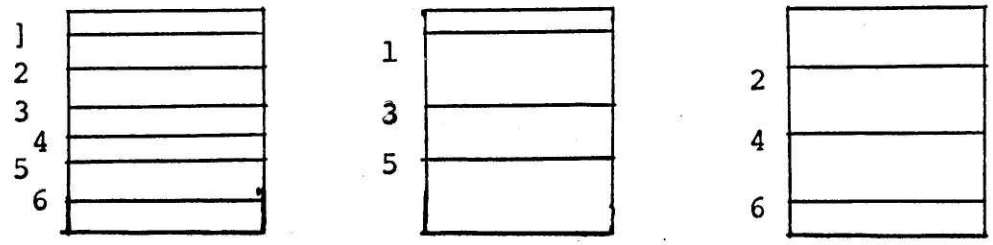
The eye can move within approximately two cubic inches of space and produce acceptable camera output.

The camera is the master timing control of the oculometer. It assumes this role because it sets the rate at which data is fed into the system. Obviously, all other components must process this data at least as fast as it is sent, on the average, or the data will not be utilized to its fullest.

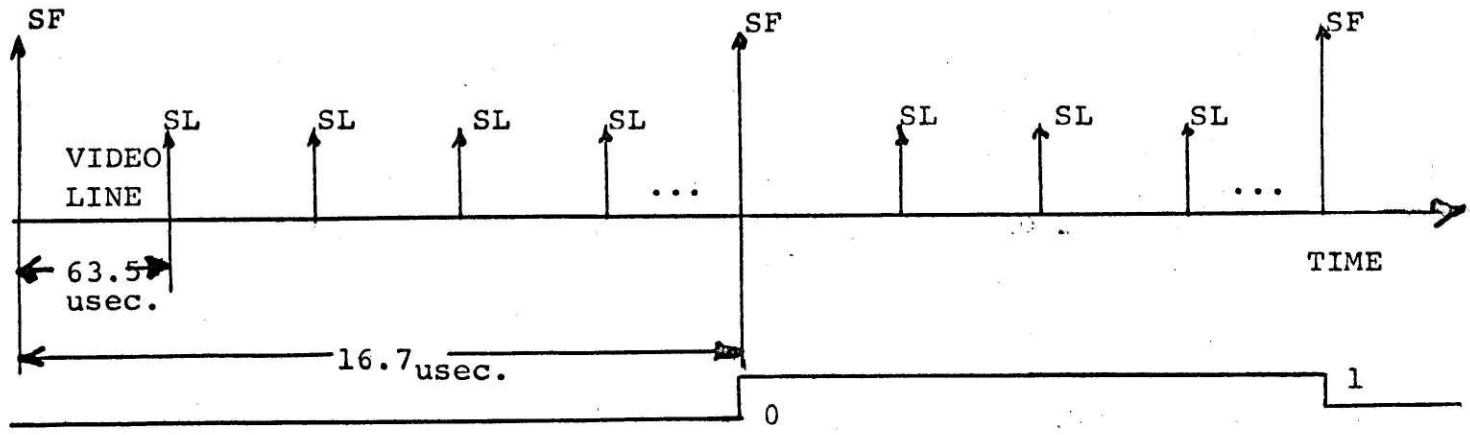
The camera scans the incoming image 30 times per second. Each scan is called a frame. A frame represents 525 parallel horizontal sweeps across the input image. Each sweep is a line. A line is transmitted every 63.5 microseconds. The frame is scanned from the top line down, scanning every other line. Thus half of the lines are left out as the trace works down the screen. To output a complete frame the scan from top to bottom must be executed twice, once outputting even numbered lines and once outputting oddd numbered lines. Each time across is called a field. To sum up, the output of the camera is a succession of frames, each frame consisting of two fields, and each field consisting of 262+ lines of video data, each field in a frame representing different video lines.

The control signals are logical results of this method of data transmission. At the beginning of each field, a Start Field (SF) pulse is transmitted. At the beginning of each line, a Start Line (SL) pulse is transmitted. Another line is low during the first field in a frame and high during the second.

CAMERA OUTPUT

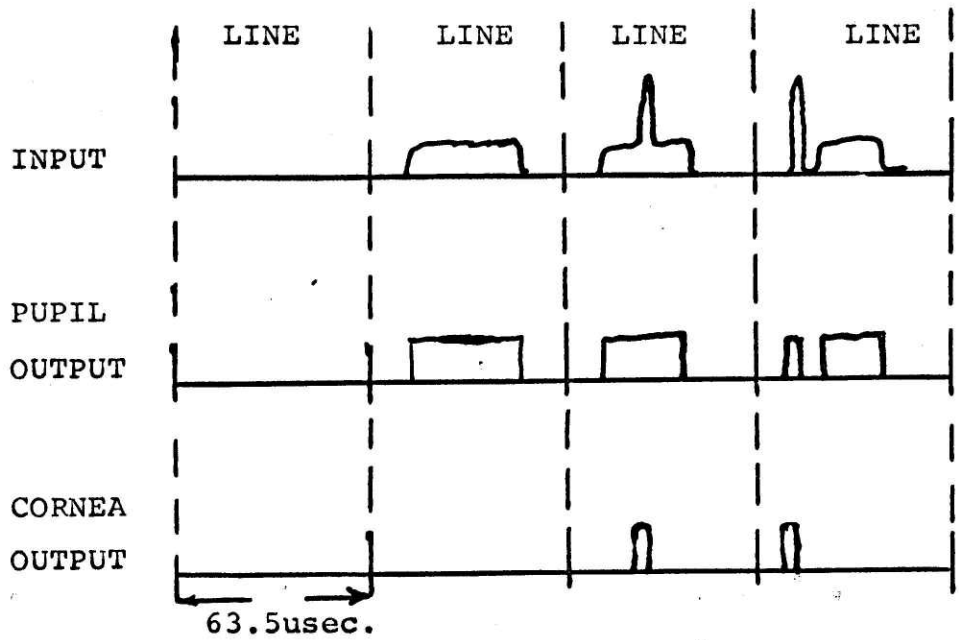



FRAME FIELD 1 FIELD 2




FIELD BIT

COMPARATOR OUTPUT



B.

The comparators perform the function of detecting the boundaries of the pupil and cornea. This operation is necessary in order to be able to eventually compute the centers of the pupil and cornea and thus the relative position of these centers.

Two comparators are used to detect the presence of the pupil and cornea, the corneal comparator being set considerably higher than the pupil comparator, due to the relative signal strengths. Thus there are two output channels from the comparators, one for the pupil and one for the cornea. These output signals are logical signals whose transition points signify when the pupil/cornea was encountered and then left on any particular line sweep.

Note that by detecting the cornea and pupil in this manner that a cornea will always trigger the pupil comparator. As long as the cornea is within the pupil boundary and not too close to that boundary of the pupil, this will have no effect. However, if the cornea is outside the pupil, this method will produce two pupils, the true pupil and the virtual pupil about the cornea. The boundary of the true pupil can become distorted since a cornea near the boundary will cause the pupil comparator to fire slightly sooner than it would otherwise.

C.

The Digital Interface System (DIS) maps a digitized Cartesian coordinate system onto the TV camera image. Each transition by an input comparator signal is assigned a coordinate in this grid. Each quantized transition is then written into

the H112 memory. Thus the DIS write digital data files into the H112 memory corresponding to the position of the outline of the pupil and the outline of the cornea upon the TV camera.

The grid system is created by two counters. The Line Counter (LC) creates the y axis. It is initialized to zero upon the receipt of an SF from the camera. Each subsequent SL increments the line counter. Thus every video line is assigned a unique line number. This is subject to the proviso that since alternate lines are scanned in a field, the LC will vary from 0 to 263 each each field even though 525 distinct lines are sent on consecutive fields. The Clock Counter (CC) establishes the x axis. It is reset to zero by every SL pulse. A ten megahertz clock increments the counter. Thus a grid is established whose coordinates range 0 to 263 top to bottom and 0 to 635 left to right.

The DIS writes three items into the H112 memory for each pupil or cornea encountered in a line. (The DIS allows up to two pupils or two corneas per line.) These items are the Clock Counter reading when the pupil/cornea was encountered, the Clock Counter reading when the pupil/cornea was left, and the Line Counter value for that line. Two data lists are created, one for all pupil encounters and one for all cornea encounters. The H112 has control over the initial address of each list. The DIS increments the appropriate address pointer after each item transfer.

Immediately after the DIS receives a Start Field pulse from the camera it accesses six control words from the H112 memory. Four of these words outline a "Gate" in the camera image. The Gate consists of two LC values and two CC values. The DIS is constrained to reject data transitions from the comparators whose coordinates would be outside this gate. The other two words are the initial addresses at which the DIS can begin to write data files.

When ever the LC is equal to one of the two LC values of the Gate, the DIS generates an interrupt. This causes the H112 to stop normal processing and enter a special routine to read two status bits. The DIS transmits a bit corresponding to the field ( 0 = first field, 1 = second field), and a bit corresponding to the interrupt ( 0 = first interrupt of field, 1 = second interrupt of field ).

To sum up the events over an entire frame: First, the DIS receives an SF from the camera. It sets the line Counter to zero. It reads in two file start addresses and four Gate values from the H112. Each Start Line pulse then increments the Line Counter and resets the Clock Counter to zero. Next, when the LC equals the top Gate value, the DIS generates an interrupt. The H112 responds and reads a 0-field,0-interrupt status. On subsequent lines, until the bottom Gate value is encountered, the DIS will input data directly into the H112 memory corresponding to the CC readings and the LC reading when comparator logic signals make transitions. Data is input

into memory locations starting with the file start addresses read in after the SF pulse. When the botton Gate is encountered the DIS generates an interrupt. The H112 responds and reads a 0-field,1-interrupt status. No further data transfers take place this field. The second field repeats the above process except that the interrupts are 1-field,0-interrupt and 1-field, 1-interrupt status.. This constitutes the events of an entire frame.

D.

The H112 is a single sequence, single accumulator, single processor, synchronous digital computer with 4096 words of twelve bit memory. It is a relatively slow minicomputer with memory access times of around two microseconds compared to other machines with 800 nanosecond times. The H112 can be interfaces with a variety of input/output devices through the use of a programmable input/output bus or a nonprogrammable Direct Data Bus.

The H112 processes the data files written by the DIS. It provides control words for the DIS and reads status bits from the DIS. The Oculometer imposes these constraints upon the H112 software:

1. In order not to lose data, the H112 must process a field of data and output any results wihtin 16.7 milliseconds after the field has been received. This is a weak constraint in that for part of this processing time, the DIS will be writing into the H112 memory the next data file. Writing via the Direct Data Channels locks out the execution of H112 machine code.

2. The H112 must be able to respond to interrupts from the DIS at any time. Interrupts times are completely asynchronous with respect to the execution of the coding. It is therefore necessary that the H112 have the capability of resuming its former machine state the interrupt.

3. The H112 must control the Gate such that valid data flow is maximized.

4. The H112 must manage the start of file addresses such that the inputting of the next data field by the DIS will not affect the processing of the last data field by the H112.

# PART II.

The software system presented here is not intended to describe the ultimate answer to the oculometer data processing problem. Rather, I will describe a system I developed to meet specific data processing requirements, a system which has existed in literally scores of forms as our knowledge has grown about the nature of the data and the most useful types of interaction between the user and the system. The system documented and analyzed here is in actual use at Honeywell Radiation Center, except for the calibrated eye direction output facility, which is not in use.

The software system was written in machine code. I had access to two compilers, but neither was adequate. One compiler was implemented on a time-sharing system in Basic. Unfortunately, it did not allow programming to the full capability of the H112. The Honeywell supplied compiler was adequate except that it was a very tedious process to load and compile a program due to the small amount of core storage available. As a result, I found it simpler to hand compile most of the algorithms used.

This part is composed of four sections. The first section details the SYSTEM algorithm. The SYSTEM algorithm controls the interactions of the H112 with the DIS. It is responsible for providing control words to the DIS and for

determining when the processing of data may begin. This section analyzes the types of timing errors that can occur. The second section describes the PROCESSING algorithm. PROCESSING applies data analysis and computational subroutines to the raw data and links to the appropriate output routine. The PROCESSING section analyzes the types of input data errors, the relative importance of the possible types of error, and how some of these errors can be avoided. The PROCESSING section analyzes methods of computing the relative centers of the pupil and cornea and describes the method used in terms of its advantages and limitations. The third section describes the OUTPUT CONTROL algorithm and the various output routines. This section deals primarily with the problem of man-machine interaction,i.e. it explores the types of controls that the operator has over the system and the type of information output that has proven useful to the operator of the system. The fourth section describes the CALIBRATION routine. The CALIBRATION routine is designed to provide an easy means of scaling the x-y axes and shifting the origin of the system.

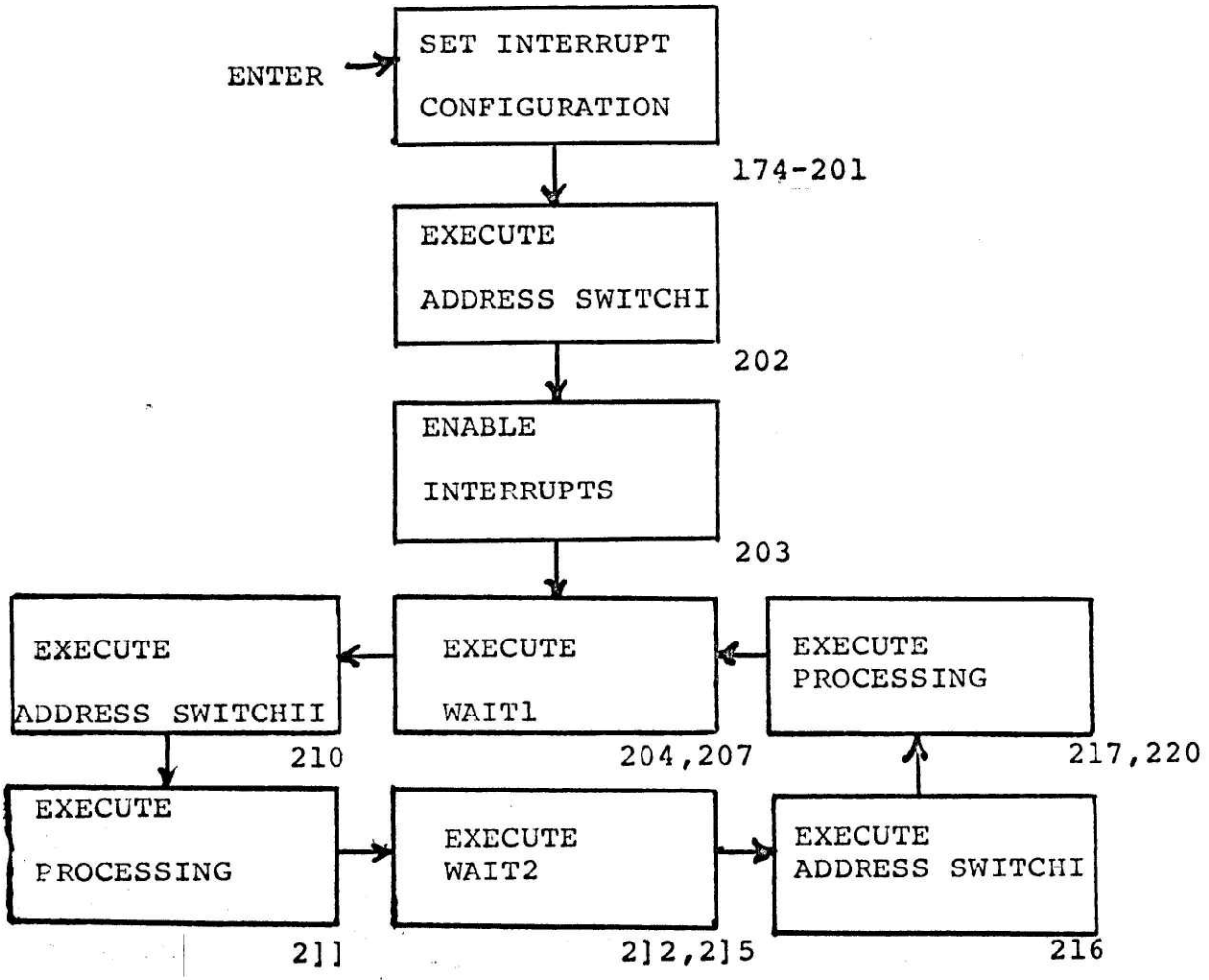
These four algorithms interact in the following manner: When the computer begins execution, it enters the OUTPUT CONTROL routine. The user may select any of three output routines or enter the CALIBRATION routine. If the CALIBRATION ROUTINE IS entered the operator will have the opportunity to computer right,left,up,down scale factors and x and y

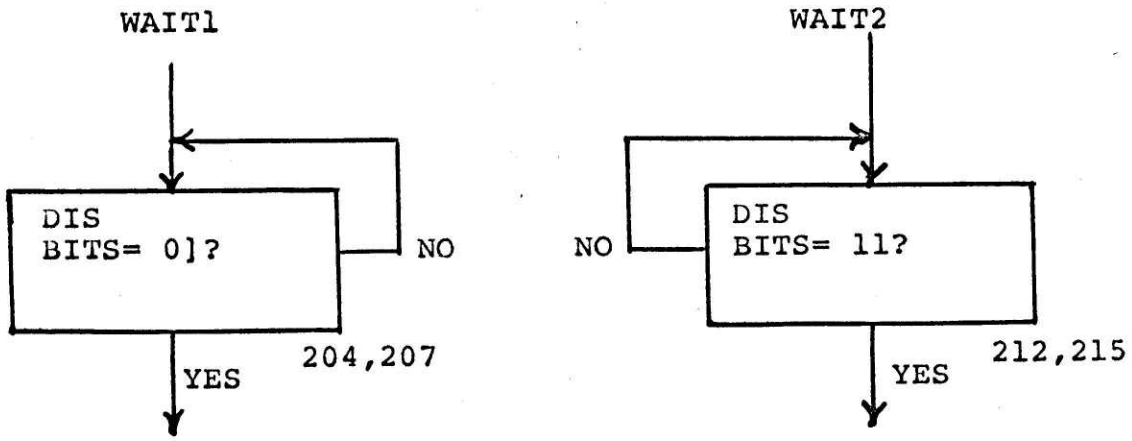
null constants. At the conclusion of the CALIBRATION routine control is returned to OUTPUT CONTROL. If the user selects an output routine, the system automatically links the appropriate output routine with the PROCESSING algorithm and initiates normal operation by entering the SYSTEM algorithm. SYSTEM synchronizes the operation of the H112 software to the DIS and controls the application of PROCESSING which processes the raw data and outputs the results using the output routine selected. At any time the user may select another output routine by pressing the teletype break key. This will cause control to be returned to the OUTPUT CONTROL routine.

## A. SYSTEM

The SYSTEM algorithm synchronizes the operation of the H112 to the second interrupt in a field. When this second interrupt is recognized, SYSTEM allocates two file start addresses to the DIS for inputting during the next field and allocates the two file start address used by the DIS during the current field to the PROCESSING routine for processing. When PROCESSING has been executed, control is returned to SYSTEM which waits for the second interrupt status of the next field to begine the cycle anew.

SYSTEM is initially entered through a series of initialization steps including: establishing interrupt configuration, such that the DIS is the only peripheral device allowed to


ENTER
SET INTERRUPT CONFIGURATION
174-201
EXECUTE ADDRESS SWITCHI
202
ENABLE INTERRUPTS
203
EXECUTE ADDRESS SWITCHII
210
EXECUTE WAIT1
204,207
EXECUTE PROCESSING
217,220
EXECUTE PROCESSING
2]]
EXECUTE WAIT2
2]2,2]5
EXECUTE ADDRESS SWITCHI
216

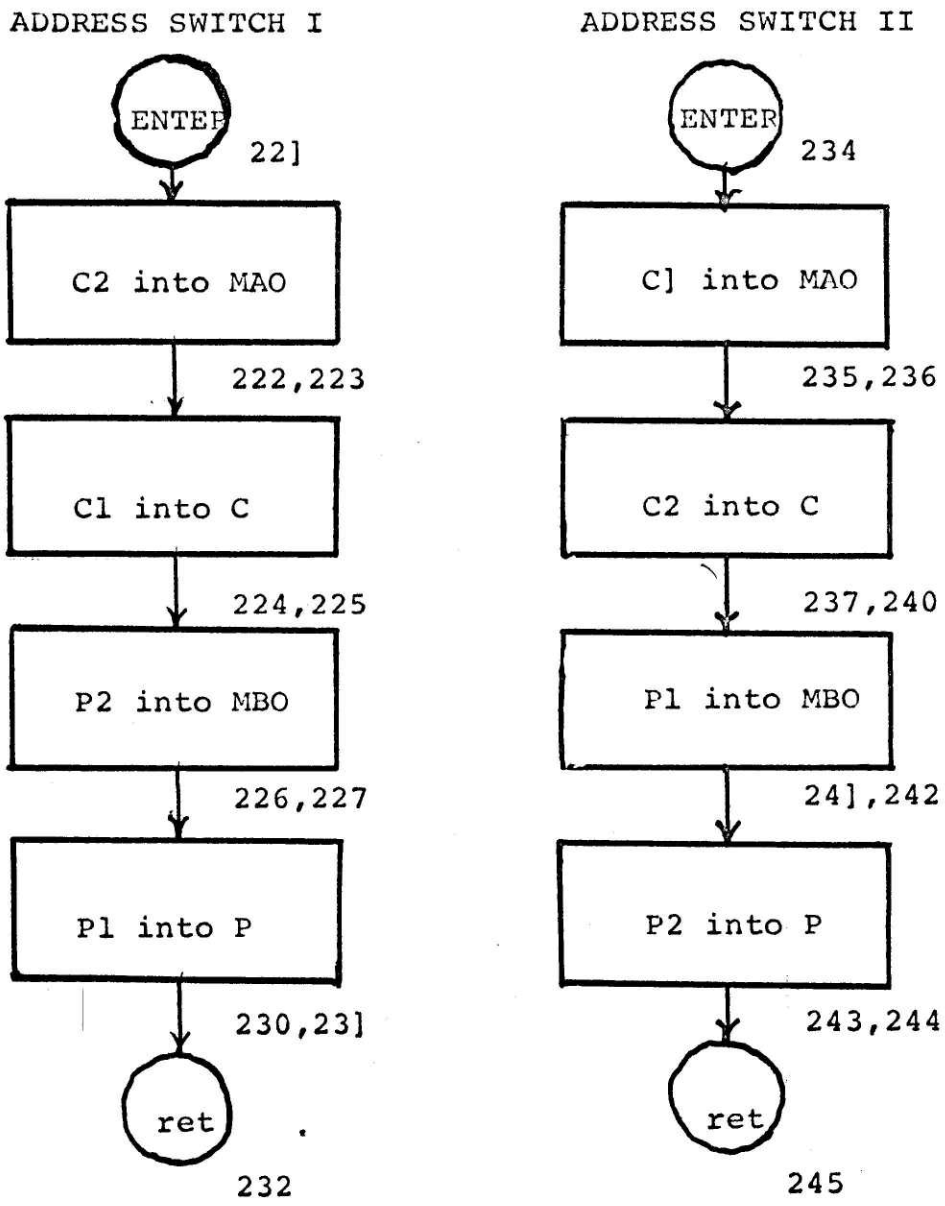

WAIT1
DIS BITS= 0]?
NO
YES
204,207
WAIT2
NO
DIS BITS= 11?
YES
212,215

interrupt; initializing the DIS status bits; initializing file start addresses; enabling interrupts; entry into WAIT1

In the actual program WAIT1 provides the necessary stall until the first field, second interrupt status is received at which time ADDRESS SWITCH1 is applied. ADDRESS SWITCH1 allocates file start addresses C1 and P1 ( C-corena,P-pupil) to the DIS and C2 and P2 to PROCESSING. PROCESSING is then executed. Control is returned to WAIT2 which waits for the second field,second interrupt status. ADDRESS SWITCH2 is applied. C2 and P2 are allocated to the DIS and C1 and P1 to PROCESSING. PROCESSING is executed and control is returned to WAIT1,etc.

The alternate field property of SYSTEM should be pointed out. SYSTEM is constructed such that is always processes a second field of a frame after a first field of a frame and a first field after a second field of a frame. This is done to insure that the output of thw system will reflect all of the data in the TV image not just the first scan or the second scan, which in fact represent different video data.

The WAIT routines have an important adjunct, the INTERRUPT routine. An interrupt is a signal from the DIS which causes the H112 to stop processing and jump to a predetermined routine. This routine is used to read in status bits from the DIS via the input/output bus. The program then resets the interrupt and restores the H112 to its machine state before the interrupt.




C1,C2 - Corneal file start addresses

P1,P2 - Pupil file start addresses

C - Cornea processing address

P - Pupil processing address

MAO - Cornea inputting address

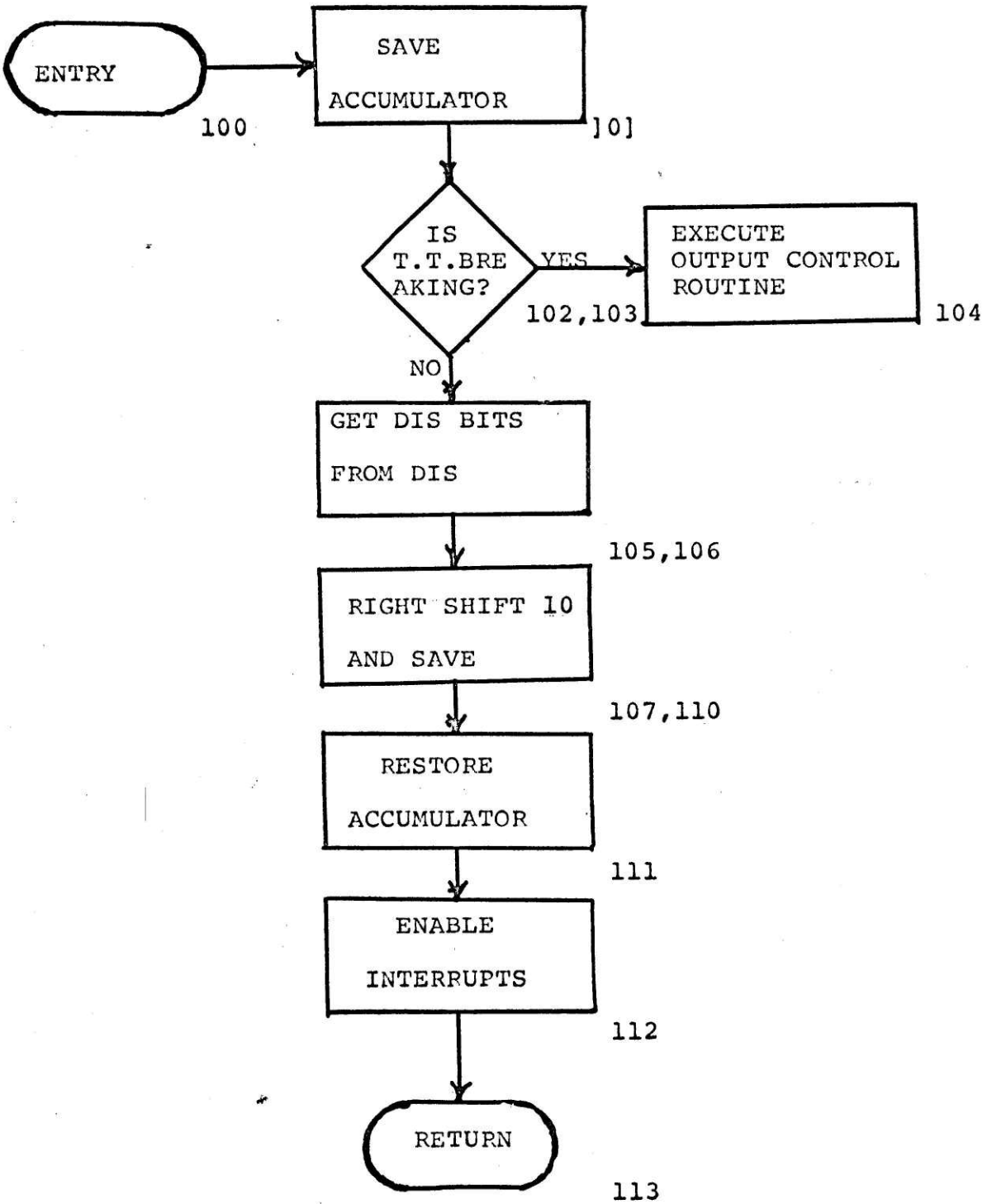
MBO - Pupil inputting address

The INTERRUPT routine is also used to determine if the teletype is breaking. If so, control is transferred to the OUTPUT CONTROL routine.

There are two reasons for the synchronization to the second interrupt of the field. First of all, the second interrupt is the point in time at which the H112 is first notified that a new data field has been totally received. Thus it is also the earliest time that the new data can be processed. Secondly, it is the last time that the H112 is notified that the next event performed by the DIS will be to read in six control words from the H112. This method of handling the control words may create serious problems if the input data files become too long.

If the processing of the data takes more than 16 milliseconds, SYSTEM may fail to manage the file start addresses correctly. Normally, processing of a field begins after the second interrupt of the field and terminates sometime before the second interrupt of the next field. Case.1. Processing terminates after the second interrupt but before the SF pulse is received by the DIS. No problem this field, but if the next field takes as long there will be problems, namely Case 2. Processing terminates after the second interrupt of the next field and just after the DIS has received the SF pulse from the camera and read in the control words for the next field. At this point the status bits still signify a second

INTERRUPT ROUTINE



interrupt status. Therefore, SYSTEM executes the file switch. PROCESSING Will process the files into which the DIS input data last field, but since the file address switch was executed after the DIS read in its control words for the next field, the DIs will input new files into the same addresses which the PROCESSING routine is using for computation. Fortunately, Case 3., the delay is so long the first interrupt occurs in the next field, will cause the SYSTEM algorithm for the appropriate second interrupt to resynchronize the system.

Unfortunately, Case 2. could have been avoided. The problem is for the SYSTEM algorithm to be able to absolutely determine the file start addresses of the DIS, no matter what point in time the PROCESSING algorithm terminates. The requirement that SYSTEM recognize the first interrupt in a field before recognizing the second interrupt would have met this requirement. ( It has been passed on to HONEYWELL.) For instance, if Case @ occurs, the new requirement would force SYSTEM to wait until it recognized the appropriate first interrupt before it could synchronize to the second, thus guaranteeing that the appropriate address switches are always executed before the DIS reads its control words. It should be noted that the appearance of a Case 2 condition usually signifies a great deal of spurious data, rather than a large amount of valid data. Thus this correction will usually result in accurate output of garbled data as opposed to garbled output of garbled data!

## B. PROCESSING

The PROCESSING algorithm proper does little actual processing. Principally, it sequences the application of data analysis, computation, and output routines to the data. It operates under no time constraints with respect to the rest of the system. It has been designed such that it will process a normal field of data in less than 16 milliseconds, thus incurring no problems for the SYSTEM algorithm.

This part of the report is written in three sections. Section 1. deals with an analysis of the data structure and the most important types of raw data error and with the algorithm used to overcome some of these data deficiencies. Section 2. analyzes methods for determining the relative position of the pupil and cornea. It also analyzes the computational routines used by PROCESSING. Sections 3. ties together the results of the previous two sections by showing how the data analysis and computation routines are sequenced to process the input data files.

### SECTION 1.

The data is structured into two files. A file is a sequential list of data, whose length is a multiple of three, terminating with three items equal to $1777_8$. One file corresponds to corneal data, the other file to pupil data. Each set of three items correspond to two thresholds crossings of

the appropriate comparator and the line counter value (x1, x2,y). The DIS is constrained to allow up to two pupils or two corneas per line.

There are several peculiarites of the eye data which must be overcome by the processing algorithm. The blink, of course, destroys the eye picture. A blink may be intercepted by a field half way down, thus distorting the image of the eye. The upper lid will sometimes obscure the eye when a person looks down. A tear sometimes can be mistaken for a cornea. It has approximately the same size and sometimes the same brilliance. If the eye is slightly out of focus, the cornea level may sink below the comparator threshold. If the eye is sl ghtly out of position, the gate will cut off a portion of the image. If the cornea is on the pupil boundary, it can cause distortion of the pupil. Defects in the camera may produce recurrent spurious data. Transients caused by switching lights can cause spurious images. The amount of pupil signal remains low except for certain angles of eyeball rotation, in which case the pupil level can become very high and appear to be corneal data.

The data analysis routine employed must be able to make distinctions between fields of data that possess the basic information necessary to compute relative postion and those fields that do not. It must further be able to minimize the effects of distortion in those fields which do possess the

requisite characteristics.

The data checking routine has these controls over the data: 1. It can change the start of file address, thus effectively losing some number of lines from the top of the picture. 2. It can change the end of file address, thus ef= fectively losing some number of lines from the bottom of the picture. 3. It can set the y value of any line to zero. This is a signal that this line of data should be ignored in further processing. 4. It can terminate the processing of the file and return control to SYSTEM. Thus, the data checking routine ( DATACHEK ) determines what data should be ignored by subsequent routines on the basis of certain assump tions about the structure of valid data.

These assumptions about valid data are embodied in DATACHEK: 1. The pupil and cornea are solid shapes. Thus valid data will be represented by a minimum number of consecutive lines. DATACHEK assumes that the pupil must have eight consecutive lines and the cornea two. 2. There is at most one valid pupil and at most one valid cornea per line. In all cases, when two pupils or corneas are encountered on any line, one is in error. Since choosing proved to be a time consuming and difficult task, the only option was to ignore both items.

As a consequence, the operation of DATACHEK is really an inspection of the y line numbers. In the pupil file, data is checked until the first eight consecutive lines are encountered.

The beginning of these eight lines becomes the beginning of the file. The first nonconsecutive line after the first eight signals the end of file. Duplicate line numbers are set to zero. (Duplicate line numbers do not affect the sequentiality requirement.) The corneal file analysis is analagous. Note that encountering the end of file code before the sequentiality requirement is fulfilled necessitates the termination of processing.

This type of error checking works quite well in the pupil case. This is because spurious data in the pupil file is almost totally of a spot nature. Large spurious pupil formations have never been encountered. Spurious data thus has a negligible effect since it is is above the pupil it is rejected, if it is even with the pupil only a small amount of valid data is rejected, and if it is below the pupil it is rejected. Even small spots will be accepted if they appear on lines that are consecutive to the true pupil. Note that this analysis is independent of the various distortional sources of error.

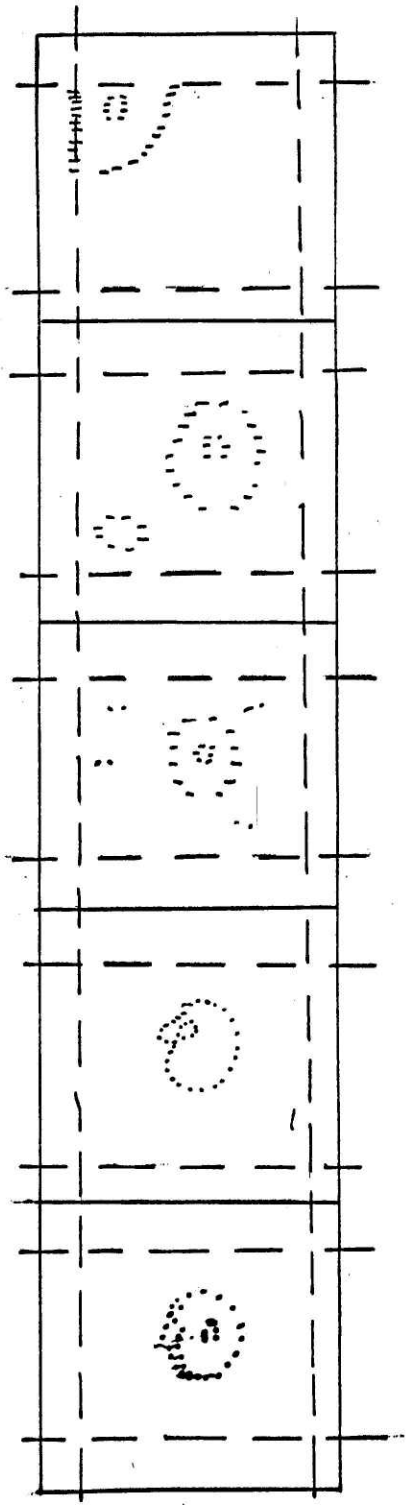
Corneal data checking is much less well-defined. Since the cornea has such a small size and is not limitted to any particular position relative to the pupil, it proved impossible to distinguish between a cornea and a tear. The basis for checking the cornea is that most tears will be below the cornea. They will therefore be nonsequential and rejected. A tear

even with the cornea will zero out the duplicate lines, leaving the non-duplicate data, which will be unreliable. A tear above the cornea will be accepted as the cornea. If one is willing to restrict the use of the oculometer to small angles, one could require that the corenea be inside the pupil, which would eliminate the tear problem.

Further data checking is done only to the extent that computational routines check to see if data lines have zero y's. The pupil x average does utilize a guard against the blink. It averages sixteen values at the bottom of the data file to calculate the x center. In this way even if the top of the pupil is distorted by the eyelid coming down, the x center will remain stable since it is calculated from data at the bottom of the pupil.

Note that some data problems have not been overcome. Certain errors can only be overcome by the operator by insuring correct positioning of the subject's eye. The most serious remaining data problem is distortion. Distortion is a readily repeatable effect due to the relative position of the cornea and pupil. The effect is due to the corneal position affecting the time that the pupil comparator fires when the cornea is near the edge of the pupil. In all cases the pupil is made larger near the cornea. The only solution to this problem is to exploit the curvature of the pupil to detect localized distortions. This is unfortunately a most

TYPICAL IMAGES OF INVALID DATA FIELDS



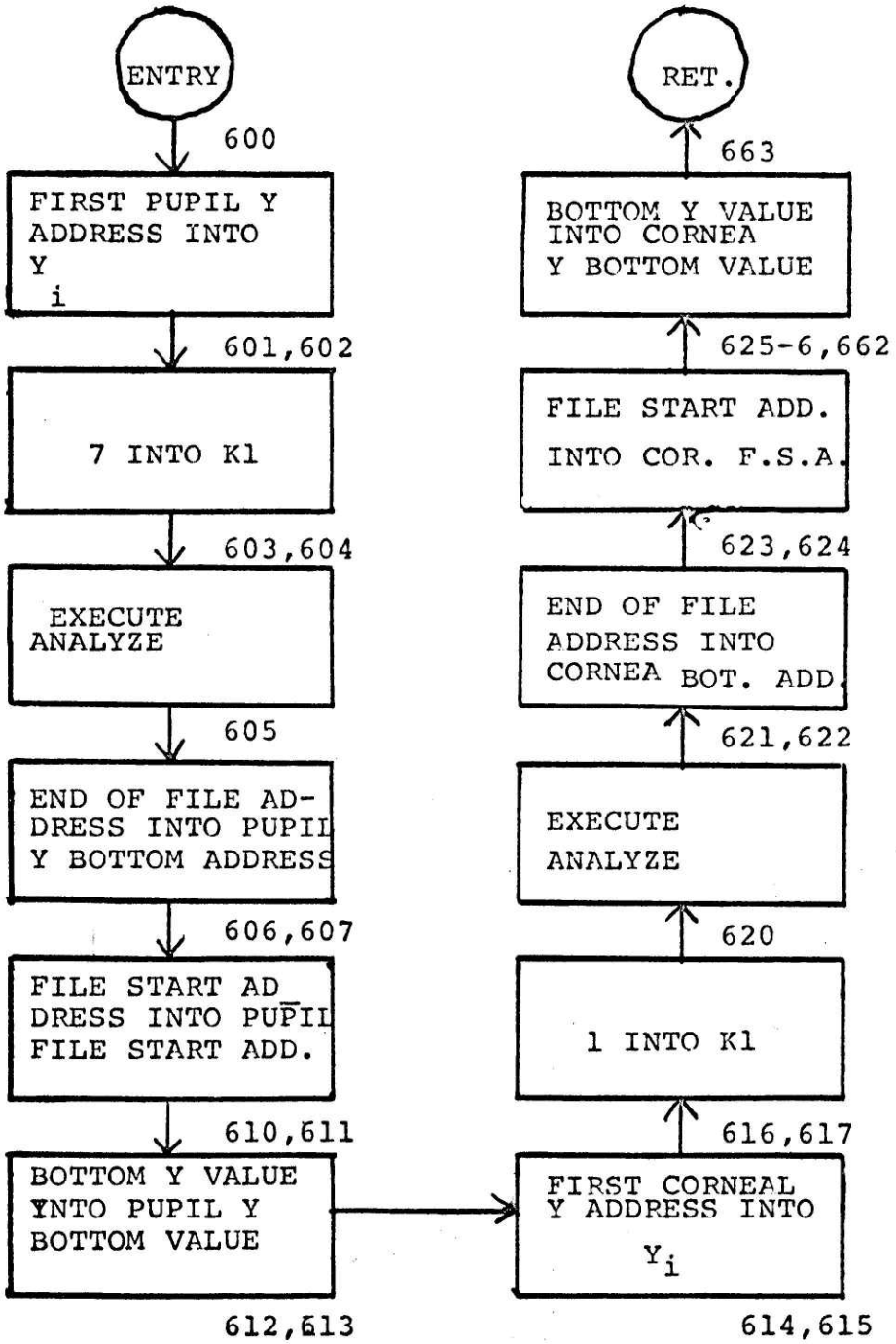
EYE IS OUT OF THE GATE

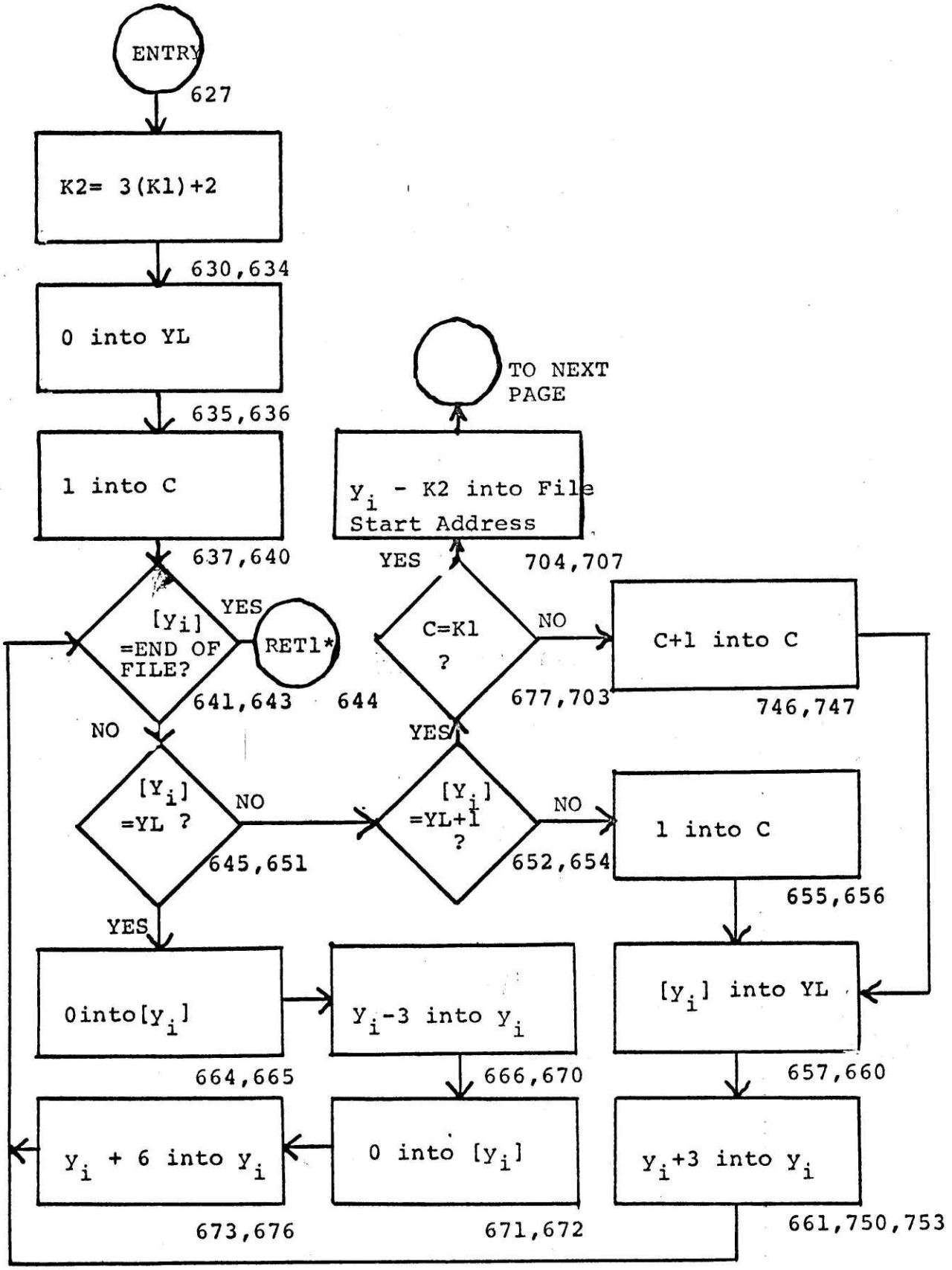
A TEAR

SPOTS  EITHER TRANSIENTS OR CAMERA DEFECT

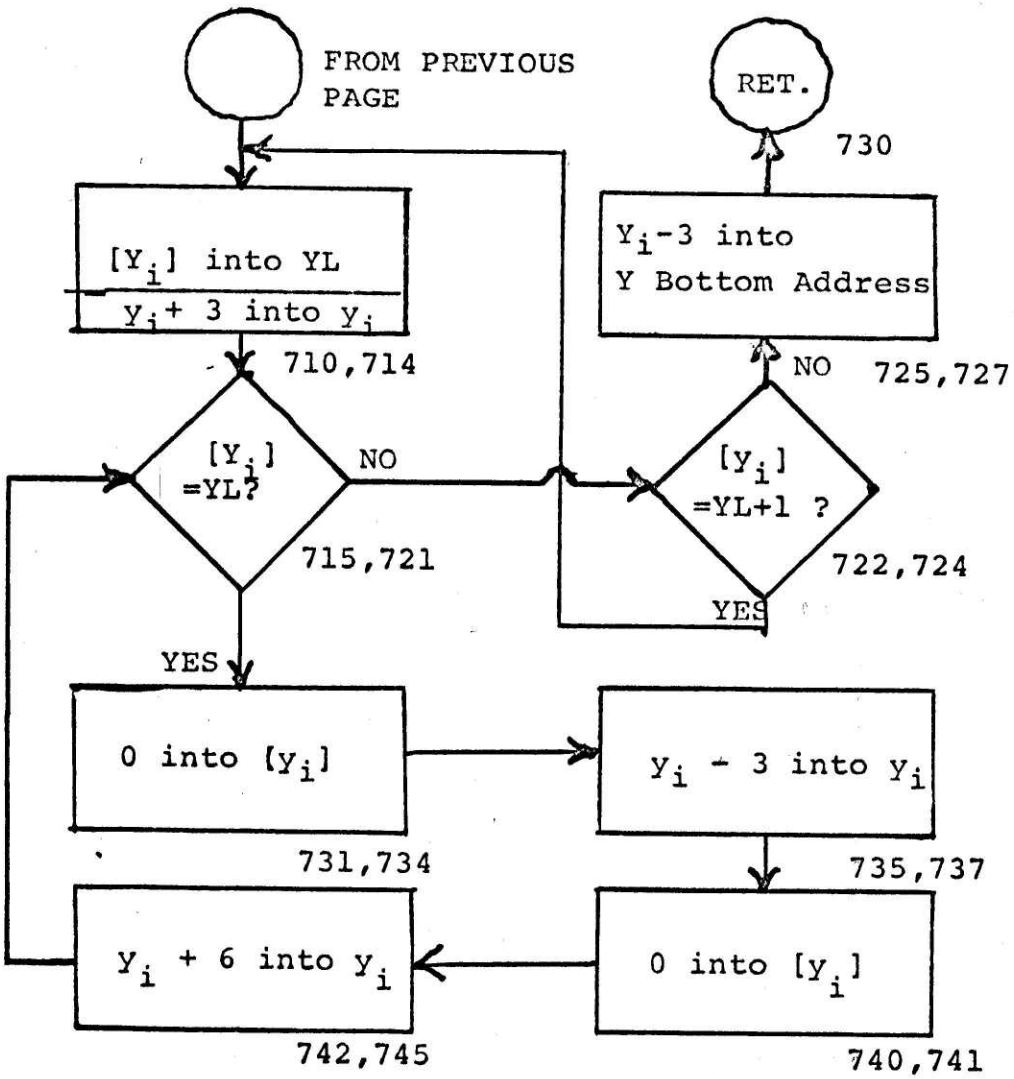
DISTORTION CORNEA NEAR PUPIL EDGE

LEFT EDGE INDISTINCT OUT OF FOCUS


ENTRY
600
FIRST PUPIL Y ADDRESS INTO Y i
601,602
7 INTO K1
603,604
EXECUTE ANALYZE
605
END OF FILE ADDRESS INTO PUPIL Y BOTTOM ADDRESS
606,607
FILE START ADDRESS INTO PUPIL FILE START ADD.
610,611
BOTTOM Y VALUE INTO PUPIL Y BOTTOM VALUE
612,613
FIRST CORNEAL Y ADDRESS INTO Yi
614,615
616,617
1 INTO K1
620
EXECUTE ANALYZE
621,622
END OF FILE ADDRESS INTO CORNEA BOT. ADD.
623,624
FILE START ADD. INTO COR. F.S.A.
625-6,662
BOTTOM Y VALUE INTO CORNEA Y BOTTOM VALUE
663
RET.




* Retl returns control to SYSTEM ( ABORTS PROCESSING)


FROM PREVIOUS PAGE
RET.
730
[Y_i] into YL
y_i + 3 into y_i
710,714
Y_i-3 into
Y Bottom Address
NO
725,727
[Y_i] =YL?
NO
715,721
[y_i] =YL+1 ?
722,724
YES
YES
0 into [y_i]
y_i - 3 into y_i
731,734
735,737
y_i + 6 into y_i
0 into [y_i]
742,745
740,741

complex problem to code without a high level compiler. It is also very time consuming from a processing time standpoint.

Note that the method of rejecting duplicate lines (Setting the y values of the line to zero.) produces an added dividend if the EYE IMAGE routine is used for outputting. One can tell at a glance if duplicate data is being input simply by the line of dots along the bottom of the oscilloscope

SECTION 2.

Consider how to compute the relative position of the pupil and cornea. Clearly, the first task is to determine the centers of these objects. Exploiting the symmetry of the x coordinates about the vertical line passing through the center of the pupil/cornea, an average of several lines of x value pairs should give an accurate indication of x center. The y case is different in that in the vertical direction we know where the shape begins and ends, but not that these lines are symmetric about a horizontal line through the y center of the shape. In other words, the x data is input into the memory in x-pairs, symmetric about the x center, while it is not immediately obvious which y-lines form symmetric pairs about the y center. Consider further that the shape is being intercepted by different y-lines every field, thus a mean y may be different each field. Several schemes suggest themselves for the y problem, the most accurate of which would be to do a center of mass calculation, that is add up the lengths of the x chords

starting from the top of the shape down and to start a similar summation from the bottom up, stopping at the y value at which the two summations are most nearly equal. Unfortunately, this method requires too much computation time. I finally settled on a scheme which made use of the x diameter of the shape, which can be determined quite accurately. The method involves determining the bottom y value of the shape and subtracting from it a number representing the y radius of the shape, determined from the x diameter of the shape. The result is the y center of the shape.

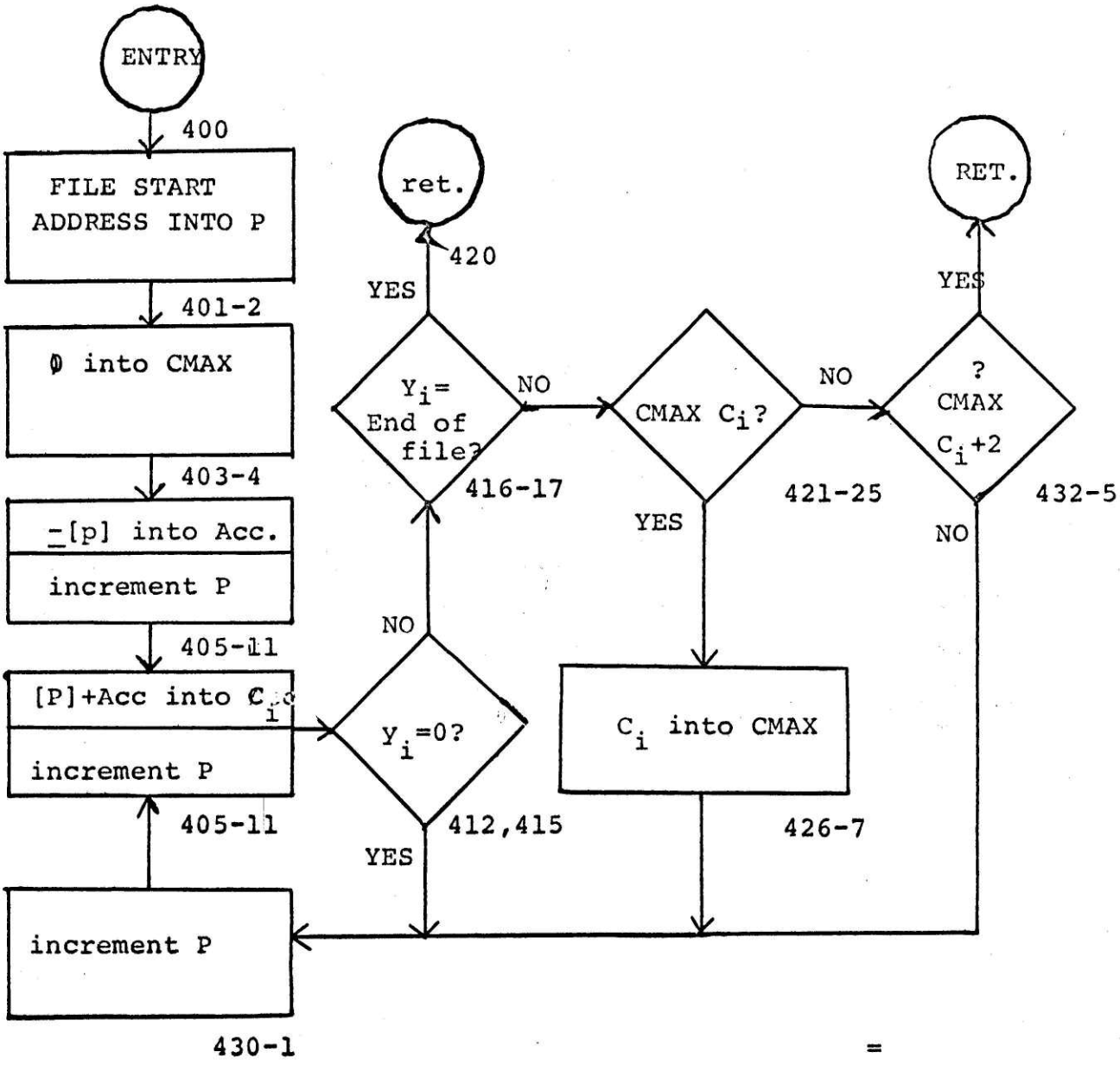
Once the center coordinates are obtained it is a straightforward operation to calculate the relative displacement of the cornea and pupil.

This method of calculation requires four computational routines: LONG CHORD determines the longest x chord of a file; CXAV determines the x center of the cornea; PXAV determines the x center of the pupil; and CONVERT which converts the x diameter of a file into the y radius of the file.

One additional computational routine is employed to ensure the quality of the output. The values of the x and y relative positions are smoothed using DIGITAL FILTERS.

LONG CHORD

This routine examines input data files by calculating the difference of the x coordinates of each data line and comparing that difference to the previous largest difference encountered. The search is initiated with the top line of the file and continues until the current relative position is more




=

$C_i$ represents current chord of search.
CMAX represents maximum chord encountered.

than two counts less than the maximum chord previously encountered. Since the pupil is nominally a circle, every comparison will normally result in the current chord becoming the maximum chord until the search goes past the center of the circle. The factor of two counts is used to guarantee that a bit of noise on any one chord won't terminate the search prematurely.

In practice, this routine has worked quite well except in two cases: 1. When the cornea is very near the pupil boundary the x diamter there will become larger. This may cause the x diameter to become too large which will eventually cause the y center of the pupil to move up, which will cause the EYE DIRECTION output to bob down. 2. If the eye is out of focus the assumption that the pupil is a circle is no longer valid. Thus the search for the longest chord of the file may be terminated prematurely.

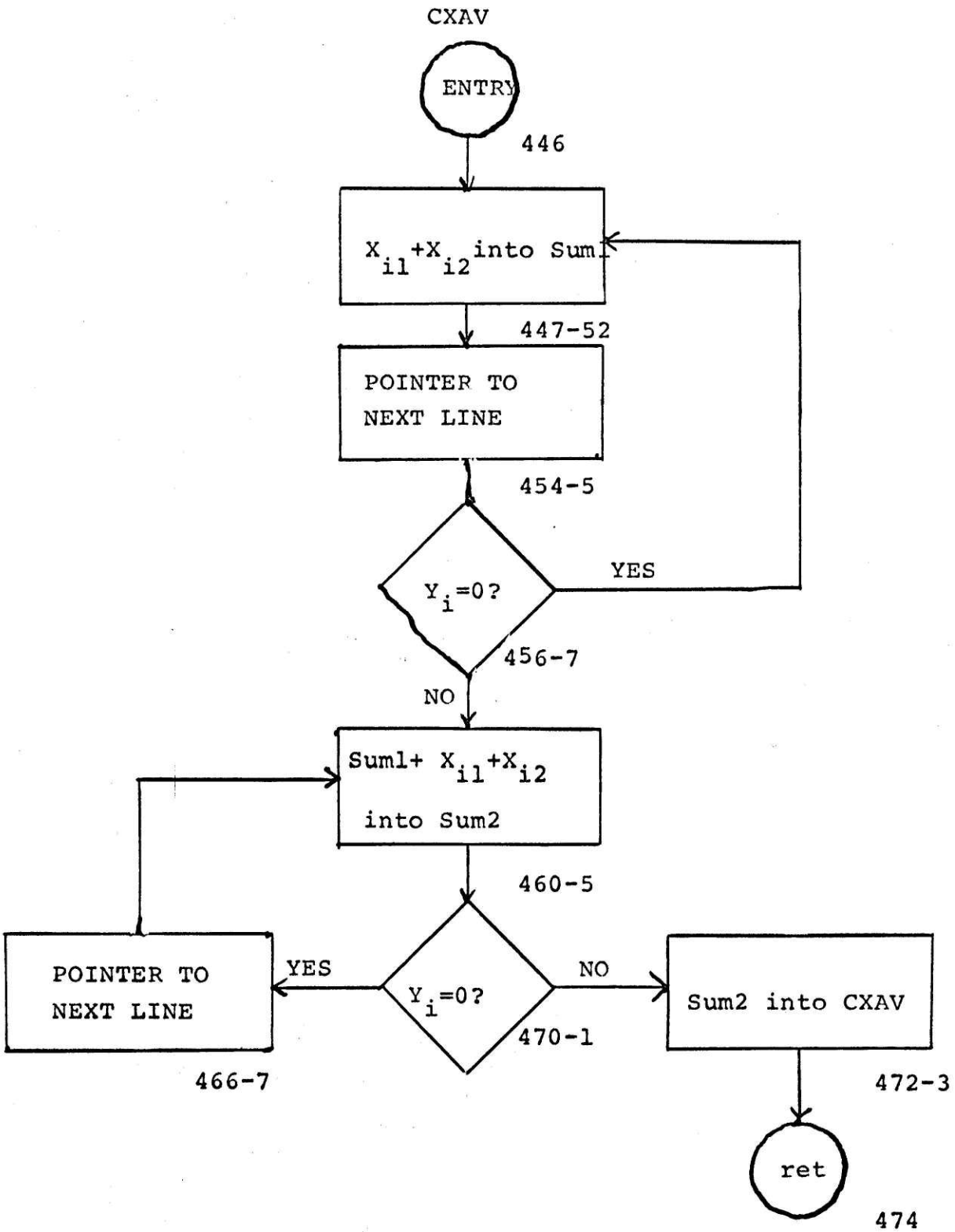
The input to LONG CHORD is the starting address of the data file to be searched. The output is the value of the longest x chord of that file. Lines whose y values are zero are ignored.

CXAV

CXAV computes the average x position of the cornea file. The input to the routine is the address of the start of the cornea file. The output is the x average to ¼ of an x count.

The cornea file is normally very short, usually only 3 or 4 lines. To avoid having to divide, the routine was written to add the first four x values of the file. This result is

CXa


CXAV
ENTRY
446
$X_{i1}+X_{i2}$ into Sum1
447-52
POINTER TO NEXT LINE
454-5
$Y_i=0$?
YES
456-7
NO
Sum1+ $X_{i1}+X_{i2}$ into Sum2
460-5
$Y_i=0$?
YES
NO
POINTER TO NEXT LINE
466-7
470-1
Sum2 into CXAV
472-3
ret
474

CXAV with two fractional bits. The H112 is only a twelve bit machine, but no overflows can occur since the maximum value that any x could have is less than $1000_8$ (This is the maximum Gate setting.). Therefore the sum of four x's must be less than $4000_8$, i.e. no overflows.

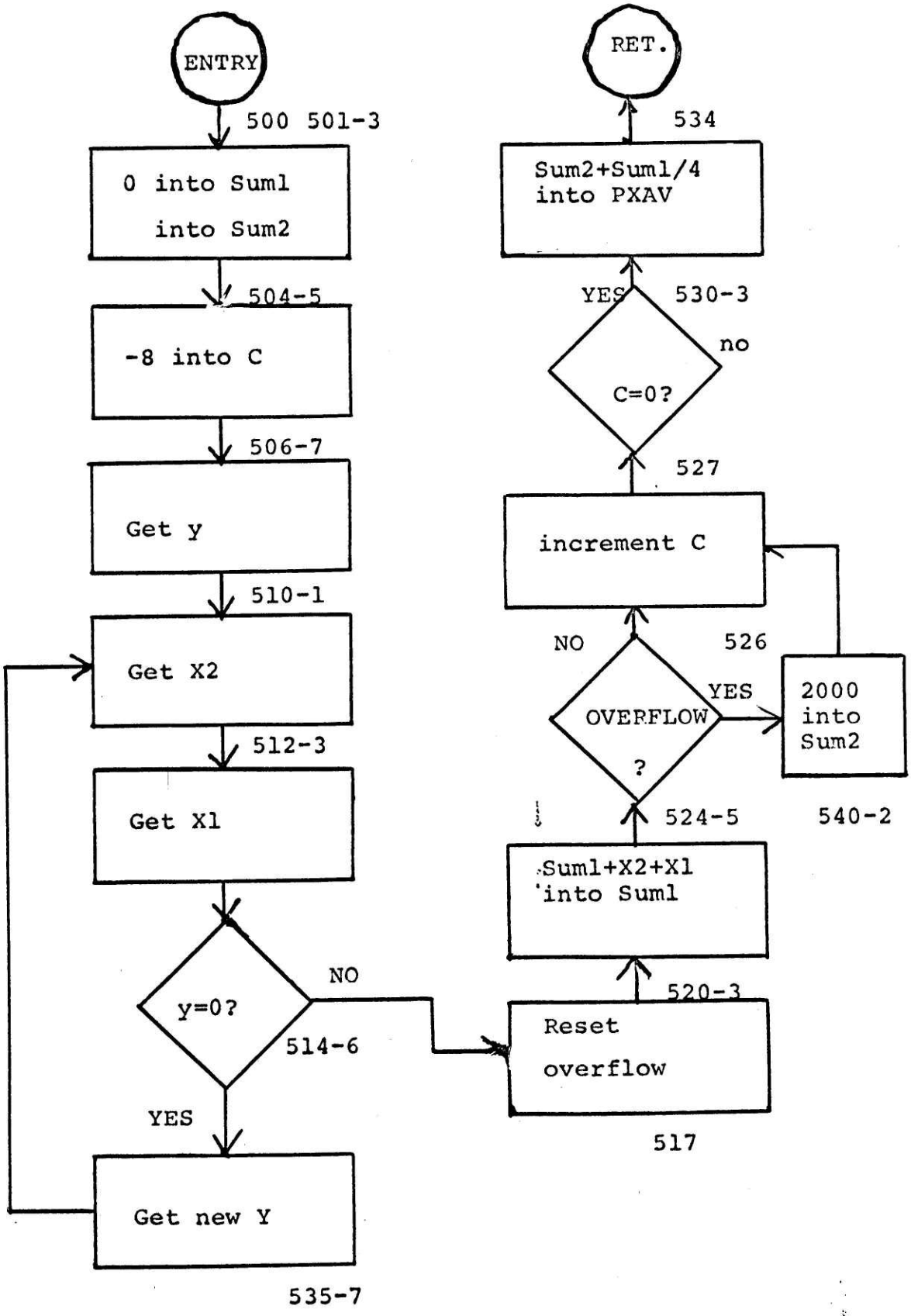
Adding four values necessitates two valid cornea lines. The DATACHEK routine will terminate processing of the data if less than two valid cornea lines are encountered. Lines which have zero y values are ignored.

PXAV

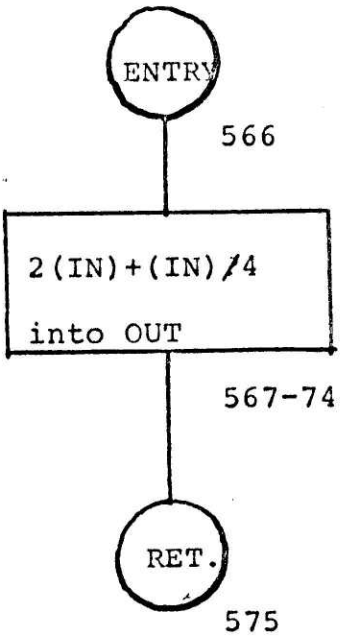
PXAV averages the bottom 16 valid x values of the pupil file. Sixteen values are used to provide an accurate average and to avoid division. Bottom lines are used to avoid blink distortional problems.

Each line, starting with the bottom line, is checked to see if the y value is zero. If it is,the line is skipped. If it is not zero, the x values of that line are added into a running sum of the x values. Note that an overflow of a 12 bit machine can occur. Also note that it can occur at most one for 16 values constrained to be less than 1000 octal. To conform to CXAV it is necessary to have two fractional bits. This necessitates dividing the sum of 16 values by 4 or right shifting two places. If an overflow occurred this result will be $2000_8$ less than the true average. The algortihm is thus: set Sum1=Sum2=0; Let Sum1 be the sum of 16 valid x values; set Sum2=2000 if an overflow occurs; right shift Sum1 two places; Sum1+Sum2= PXAV.

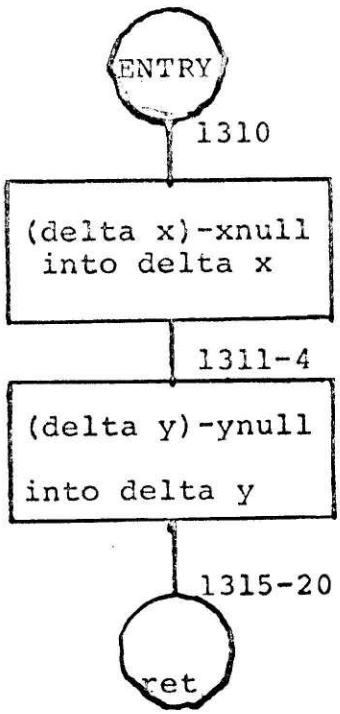
PXAV


ENTRY
500 501-3
0 into Sum1
into Sum2
504-5
-8 into C
506-7
Get y
510-1
Get X2
512-3
Get X1
y=0?
NO
514-6
YES
Get new Y
535-7
Reset
overflow
517
520-3
Sum1+X2+X1
into Sum1
524-5
OVERFLOW
?
NO
526
YES
2000
into
Sum2
540-2
increment C
527
C=0?
YES
530-3
no
Sum2+Sum1/4
into PXAV
534
RET.

CONVERT




NULL CONSTANT



Note that DATACHEK guarantees that the pupil will have eight valid lines and thus 16 valid x values.

CONVERT

The CONVERT routine exploits the geometry of the camera and the shape of the x-y grid to convert x counts into y counts divided by two,i.e. x diameters into y radii.

The formula for conversion is:

$(\frac{1}{2})$(X counts)(262.5 y counts/3 units)(4units/635 x counts) = Y counts

or

$$\frac{(\frac{1}{2})(262.5)(4)}{(635)(3)} \text{ ( X diameter) = ( Y radius)}$$

$$.275 \text{ ( X diameter) = ( Y radius)}$$

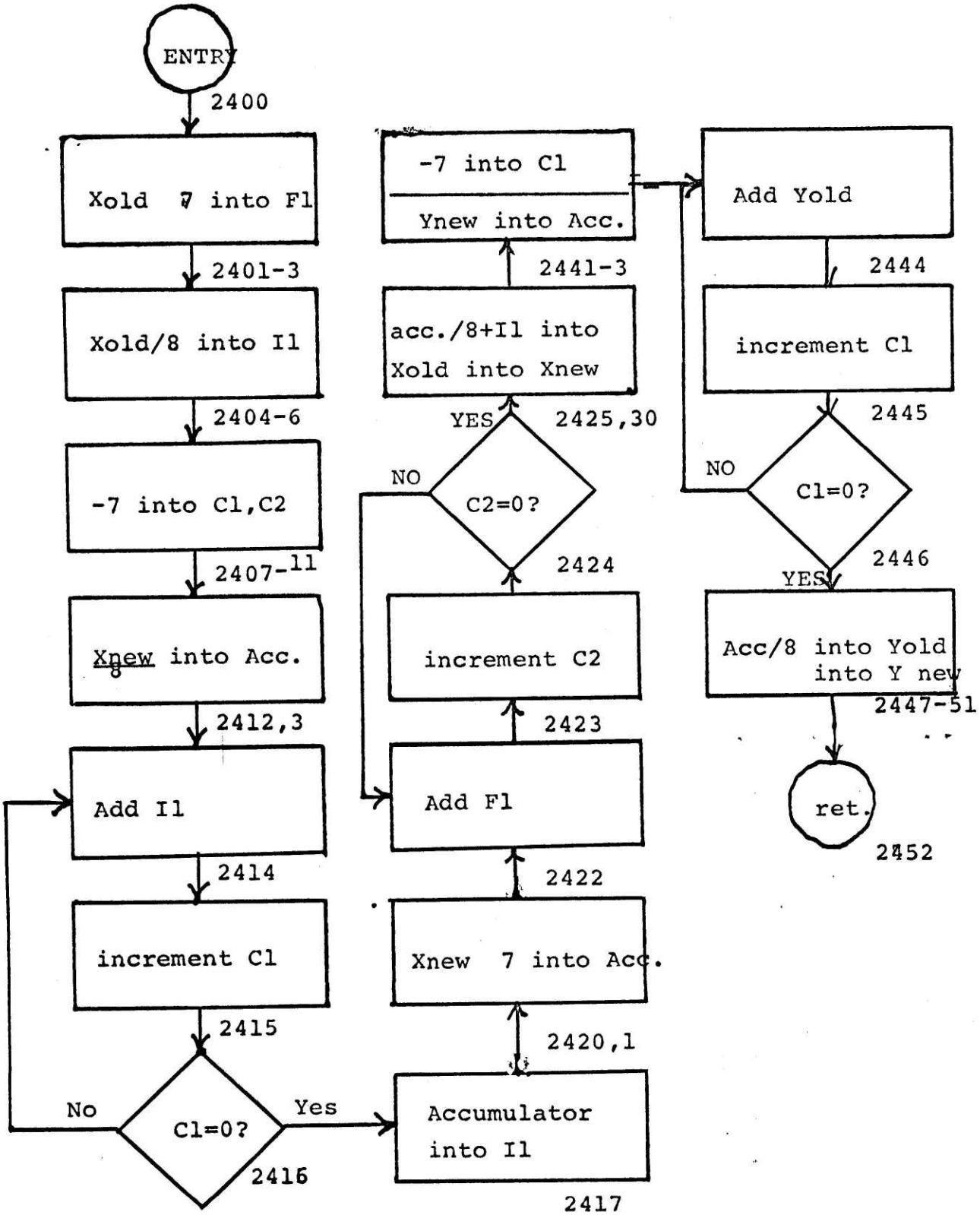
(Note that 4/3 is the aspect ratio of the TV camera.)

The CONVERT routine approximates this by:

$$2.25\ x = y$$

This is approximately equivalent to the conversion formula with three bits beyond the decimal point retained. ( To check- divide by eight- .28125x=y)

If the input is less than $3434_8$ ( an x diameter of $707_8$ ) no overflows will occur.


ENTRY
2400
Xold 7 into F1
2401-3
Xold/8 into I1
2404-6
-7 into C1,C2
2407-11
Xnew into Acc.
8
2412,3
Add I1
2414
increment C1
2415
No
C1=0?
Yes
2416
Accumulator into I1
2417
2420,1
Xnew 7 into Acc.
2422
Add F1
2423
increment C2
2424
NO
C2=0?
YES
2425,30
acc./8+I1 into Xold into Xnew
2441-3
-7 into C1
Ynew into Acc.
Add Yold
2444
increment C1
2445
NO
C1=0?
YES
2446
Acc/8 into Yold into Y new
2447-51
ret.
2452

## DIGITAL FILTERS

The function of the digital filters is to smooth the eye direction output. Each field the output value is determined by the following formula:

(7/8)(Old Value)(1/8)(New value)= Output Value.

It takes approximately 17 iterations of the digital filter to achieve .9 of a unit step at time zero. Since there are about 60 iterations per second, the time constant for the filter is a little under ¼ of a second. This has proven more than adequate to achieve a well behaved, responsive trace in the EYE DIRECTION routine.

The Filters must cope with the contingency that overflows are likely to occur since the summation must be carried out before the division by eight to attain maximum accuracy. The filters overcome this problem by splitting the values, performing the summations separately and recombining the results.

## SECTION 3.

The first action done by PROCESSING is to execute the data analysis routine, DATACHEK. The input to DATACHEK resides in a common area ( The zero sector of the h112.) Before calling DATACHEK, PROCESSING must first initialize two variables to the respective addresses of the firsty't in the cornea and pupil files. This is done by incrementing each file start address by two. The output of DATACHEK is: the starting addresses

of the data files;the addresses of the last valid item in each file; and the value of the bottom valid y in each file.

Next, the longest chords of the cornea and pupil files are found. The LONG CHORD routine has as its input a file address in common and it returns the x value of the diamter to another common location. PROCESSING applies LONG CHORD to the corneal file and them to the pupil file.

CXAV is executed, then PXAV. These routines manage there I/O completely, making use of variables stored in the common area. The output of these two routines is the x center of the cornea and the x center of the pupil.

PROCESSING next computes the relative x position of the cornea and pupil.

The CONVERT routine is then applied to the x diameter of the corneal file, yeilding the y radius of the cornea (within 1/8 of a y count).

The bottom y values of the pupil and cornea are multiplied by eight ( left shifted three) to align the decimal points.

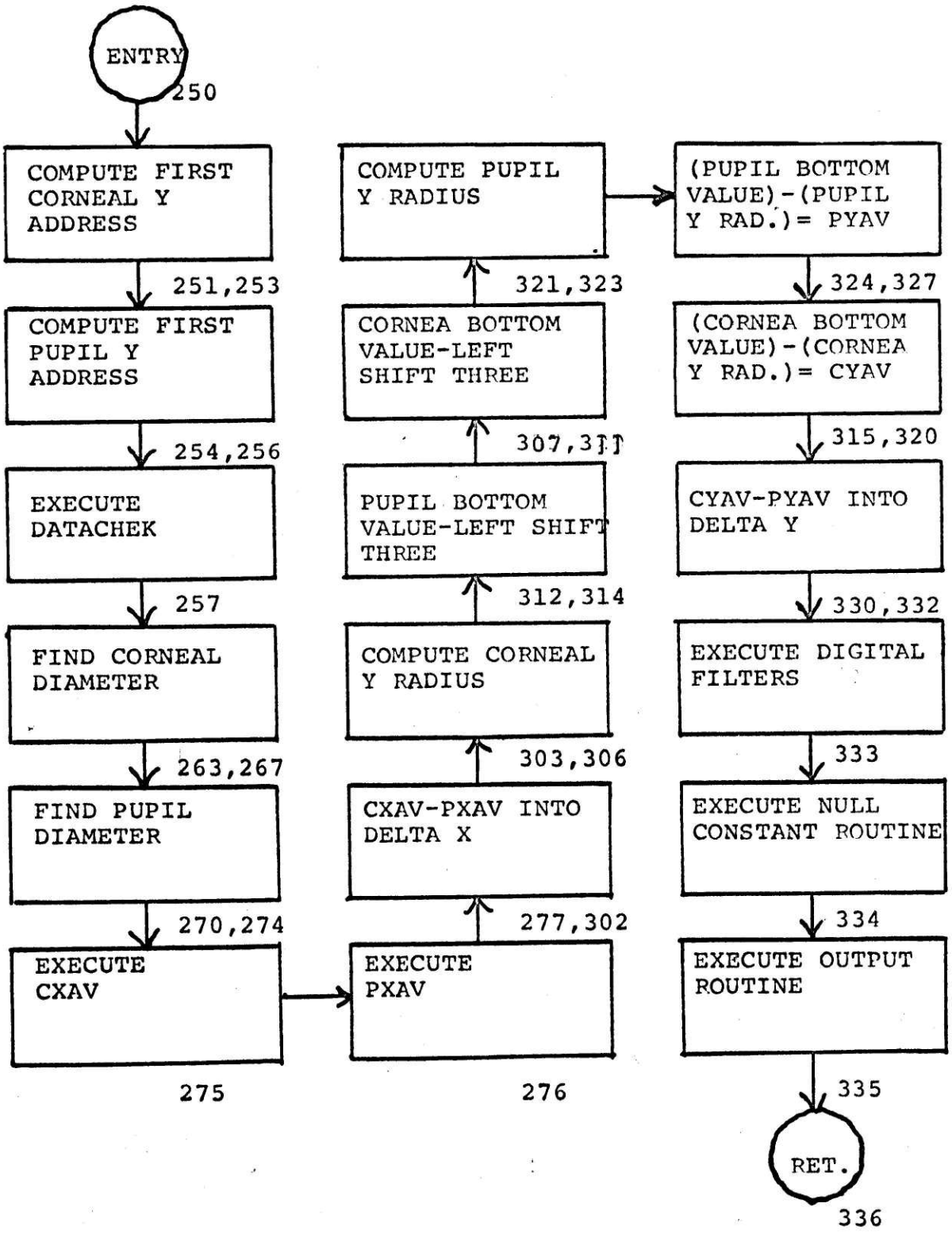
The corneal y average,CYAV, is computed.

The CONVERT routine is applied to the x diameter of the pupil file,yeilding the y radius of the pupil ( within 1/8 of a count). The pupil y average is computed (PYAV).

PROCESSING computes the relative y position.

The DIGITAL filters are applied to delta x and delta y.

PROCESSING


ENTRY
250
COMPUTE FIRST CORNEAL Y ADDRESS
251,253
COMPUTE FIRST PUPIL Y ADDRESS
254,256
EXECUTE DATACHEK
257
FIND CORNEAL DIAMETER
263,267
FIND PUPIL DIAMETER
270,274
EXECUTE CXAV
275
277,302
EXECUTE PXAV
276
303,306
CXAV-PXAV INTO DELTA X
312,314
COMPUTE CORNEAL Y RADIUS
307,311
PUPIL BOTTOM VALUE-LEFT SHIFT THREE
321,323
CORNEA BOTTOM VALUE-LEFT SHIFT THREE
COMPUTE PUPIL Y RADIUS
(PUPIL BOTTOM VALUE)-(PUPIL Y RAD.)= PYAV
324,327
(CORNEA BOTTOM VALUE)-(CORNEA Y RAD.)= CYAV
315,320
CYAV-PYAV INTO DELTA Y
330,332
EXECUTE DIGITAL FILTERS
333
EXECUTE NULL CONSTANT ROUTINE
334
EXECUTE OUTPUT ROUTINE
335
RET.
336

PROCESSING lastly calls the output routine which was selected during execution of OUTPUT CONTROL.

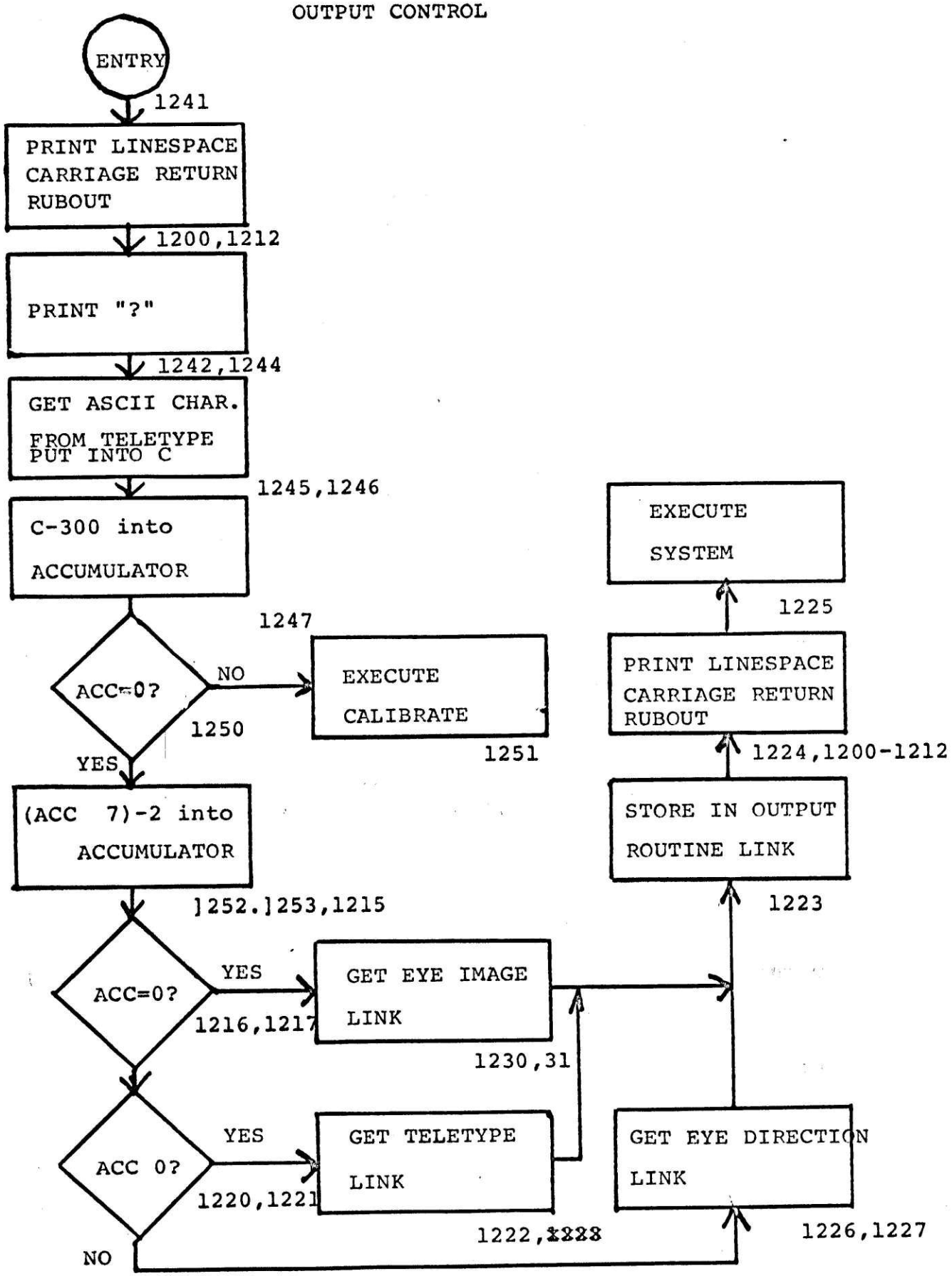
Control is returned to SYSTEM. This concludes the processing of the current field.

## C. OUTPUT CONTROL

During the course of my work it became very readily apparent that the operation of the software could be irrevocably damaged through an inexperienced computer operator. It was also obvious that most people operating this system would like to be able to switch from one form of output to another. ( For example, if the analog output of eye position was noisy, a person would like to switch to a picture of the eye to see if the input data to the computer was stable.). It was apparent thay any method of changing output relying upon the user to change the Instruction Location Counter or to access control words manually was fraught with risk for the software system.

For these reasons, I developed a method of program control over the computer involving the minimum of actions by the user: master clear; press start. Initially, the computer responds, through the teletype with "?". The user may respond "1","2", or "3". This signifies TELETYPE output, EYE IMAGE output, or EYE DIRECTION output respectively. Optionally, the user may specify "c", causing the computer to enter the CALIBRATION routine. The computer automatically establishes the output

OUTPUT CONTROL



configuration and enters the SYSTEM algorithm. At any time during the operation of the system the operator may change the output by depressing the break key for a few seconds. The computer will respond "?" as above.

The output routines have undergone more changes than any other part of this software system. At one point, for example, the EYE IMAGE routine would output not only the raw eye data on analog channels, but a box representing the coordinates of the Gate.

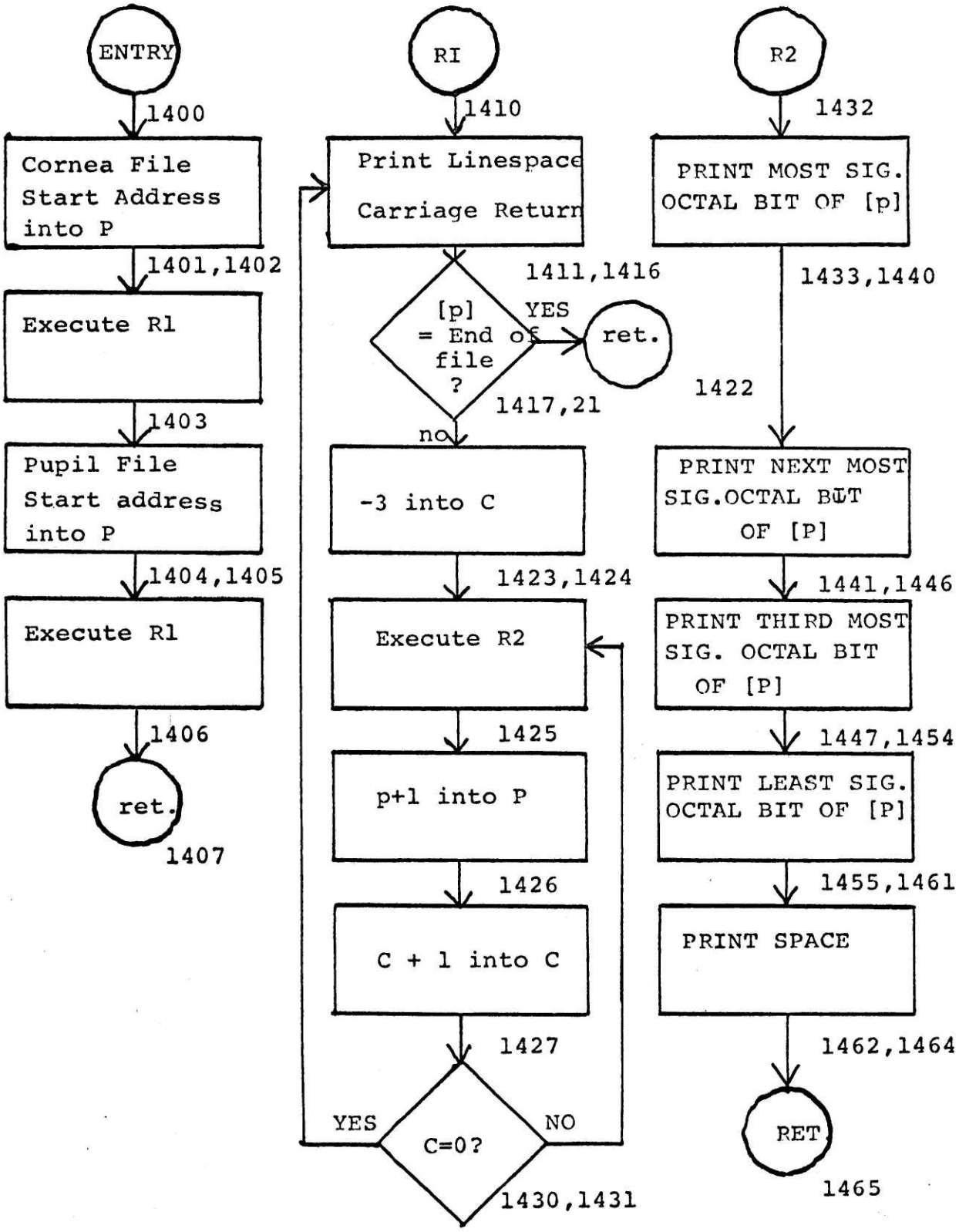
These changes have reflected the differing needs of the people who have used the system. For example, the people who are selling the system are concerned that eye direction be represented by a steady,responsive dot; they would like a system that is easy to use; they would like to be able to scale outputs. The people who designed and built the system had a greater need for precise data output forms. They needed to be able to examine the state of the data at different points in the system.

Three routines are currently operational. They are EYE IMAGE, EYE DIRECTION, AND TELETYPT output. EYE DIRECTION provides an analog signals on two channels representing relative x and y eye direction. EYE IMAGE provides analog voltages which trace the pupil and corneal shape. TELETYPE lists the input data files on the teletype.

TELETYPE

The teletype output algorithm has existed in several forms

TELETYPE ROUTINE



during the course of the testing of the system. In its most useful form the program would print the cornea and pupil files for a field and a vector of computed values from the PROCESSING algorithm, thus allowing one to pinpoint computational errors. In its current form the vector of addresses has been discarded since they are not useful to the people operating the system.

I chose to output the raw data in a form which preserved the right,left,up,down structure of the eye. This is accomplished by printing one line per each line received from the DIS, in the same order as they were input from the DIS. See next page.

The operation of the algorithm is accomplished through the use of a pointer which is set to the current address of the value to be output. The pointer is incremented after each word is output. The program automatically spaces after every word and linespaces and carriage returns after every three words (x1,x2,y). The program is terminated when the value of a word to be output is 1777.

Output is octal. Since all output is numeric the conversion from binary to ASCII is trivial.(The ASCII equivalent of an octal bit is just 260 plus the octal bit.)

8

EYE IMAGE ALGORITHM

The primary requirement of this algorithm is to display a picture of the eye upon an oscilloscope, with a mimimum of

TELETYPE OUTPUT ROUTINE SAMPLE

(Cornea)
233 234 ]3]

233 235 132

232 236 133

233 235 134

Each line contains X1 x2 y.

(Pupil)
240 241 120

240 246 121

237 255 122

230 267 123

225 270 124

222 272 125

222 273 126

222 272 127

222 272 130

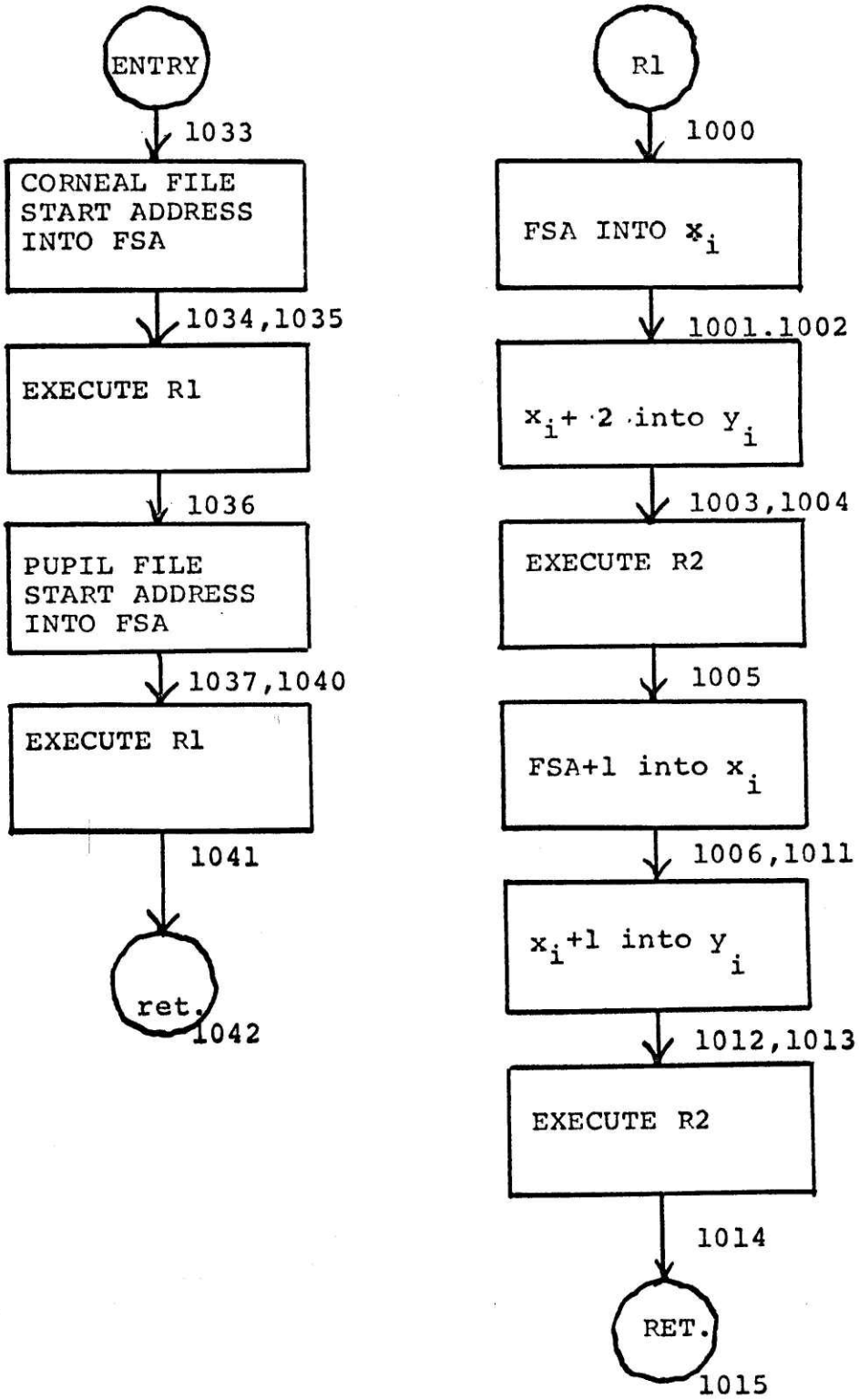
225 265 131

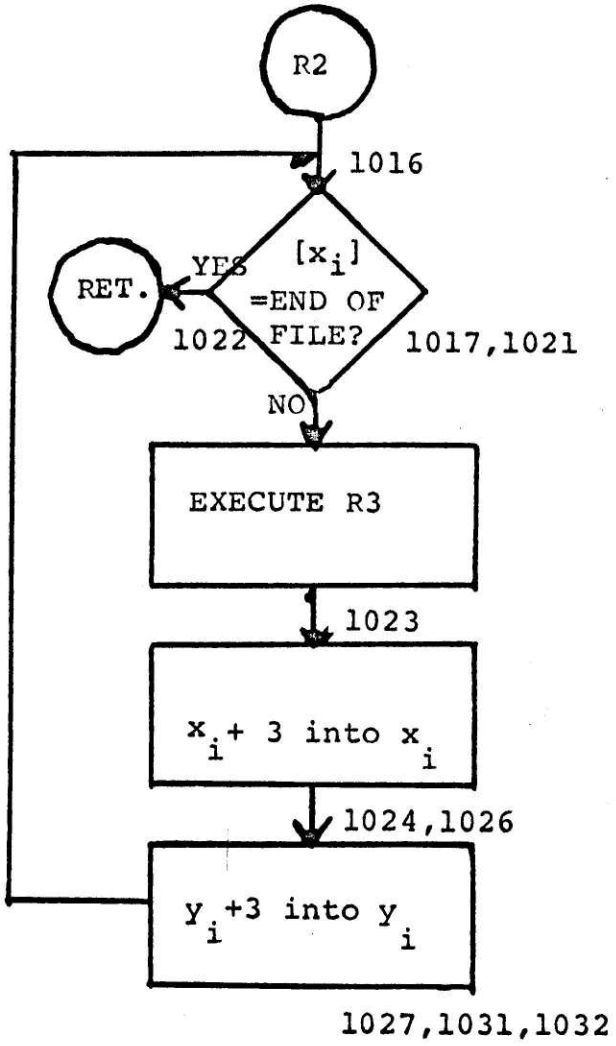
230 260 132

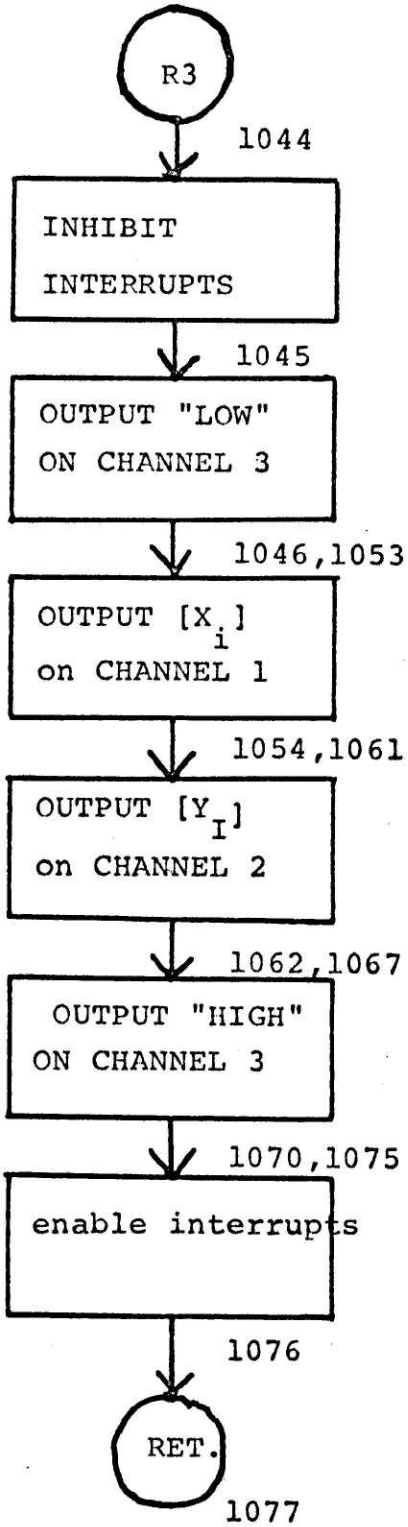
237 250 133

240 247 134

EYE IMAGE ROUTINE




R2
1016
RET.
YES
$[x_i]$ =END OF FILE?
1022
1017,1021
NO
EXECUTE R3
1023
$x_i$ + 3 into $x_i$
1024,1026
$y_i$ +3 into $y_i$
1027,1031,1032


R3
1044
INHIBIT INTERRUPTS
1045
OUTPUT "LOW" ON CHANNEL 3
1046,1053
OUTPUT $[X_i]$ on CHANNEL 1
1054,1061
OUTPUT $[Y_I]$ on CHANNEL 2
1062,1067
OUTPUT "HIGH" ON CHANNEL 3
1070,1075
enable interrupts
1076
RET.
1077

spurious images due to the process of moving the trace from one spot to the next. The algorithm accomplishes this with one spurious spot, except that the image is upside down. If the output medium has provision for intensity modulation no spurious images are produced.

The upside down characteristic is a consequence of using an oscilloscope in which a positive voltage must be above a a less positive voltage, and a D/A converter which converts a "larger" two's complement number into a larger voltage than the voltage corresponding to a smaller two's complement number. The problem is that the y coordinate system used goes from zero at the top of the picture to 262 at the bottom of the picture. An inverter must be used somewhere in the system to have an absolutely accurate analog image output. This defect has been ignored.

This algorithm essentially starts at the top of the cornea file outputting $x_{il}, y_i$, incrementing i until the end of file marker is encountered. Next $x_{i2}, y_i$ is output from i=1 to the end of file. The pupil is then output in the same manner.

A spurious dot appears in the typical output because when the output changes from the cornea file to the pupil file the trace makes a transition in x and then a transition in y. The point x',y shows up clearly on the output. Note that these tran-

sition point occur during the outputting of a single file. They are not discernible because of their proximity to the boundaries.

EYE DIRECTION I.

This version of the eye direction output is currently not in use. This version is compatible with the CALIBRATION Routine.

This routine takes the filtered values of delta x and delta y ( the relative positon of the cornea and pupil) and shifts them by the appropriate null constants. The results are scaled by the appropriate scale factors. A simple output routine than outputs them onto analog channels. For a discussion of these constants see the CALIBRATION routine description.

EYE DIRECTION II.

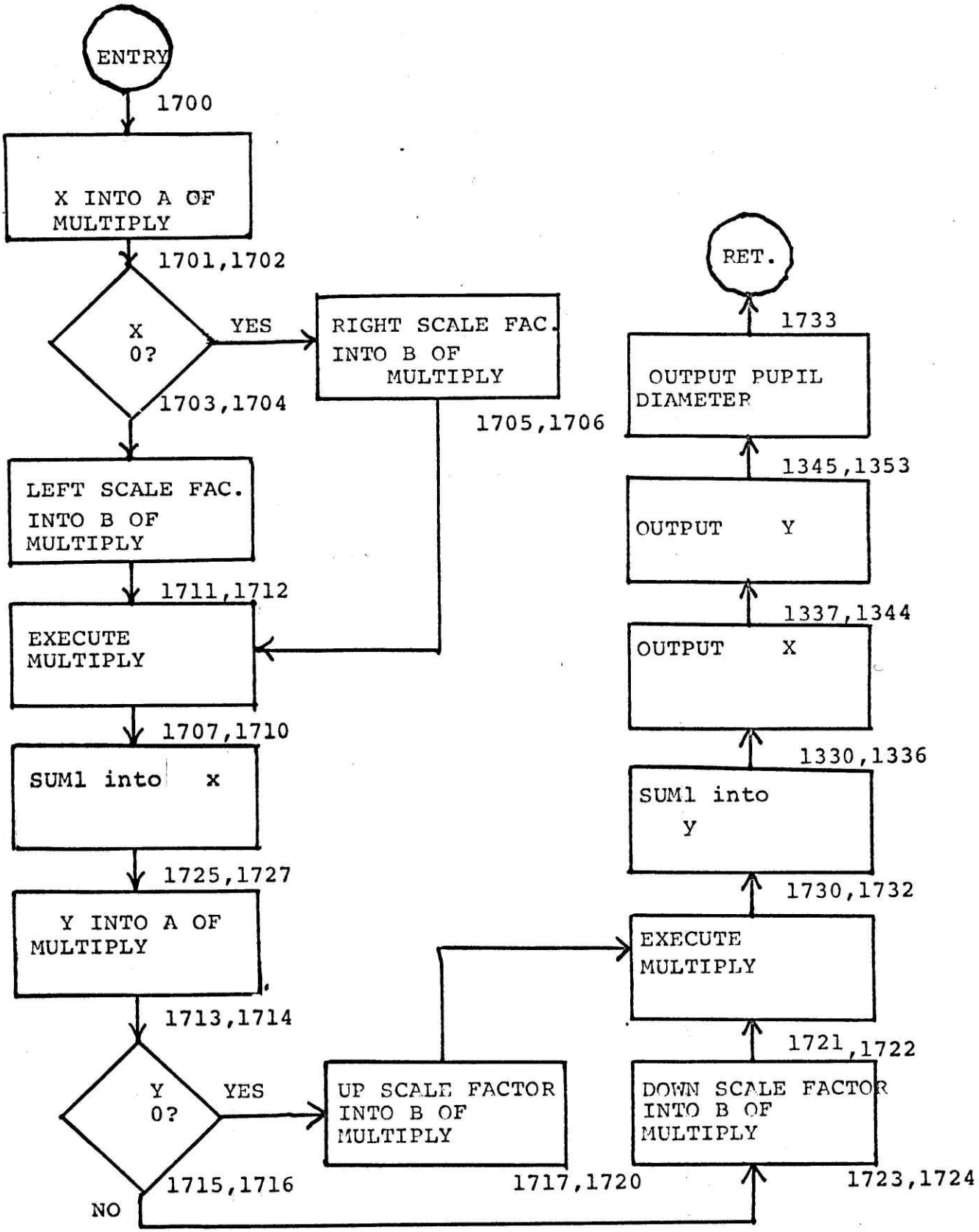
This routine, currently implemented, simply outputs the filtered values of delta x and delta y onto analog channels.

## D. CALIBRATION

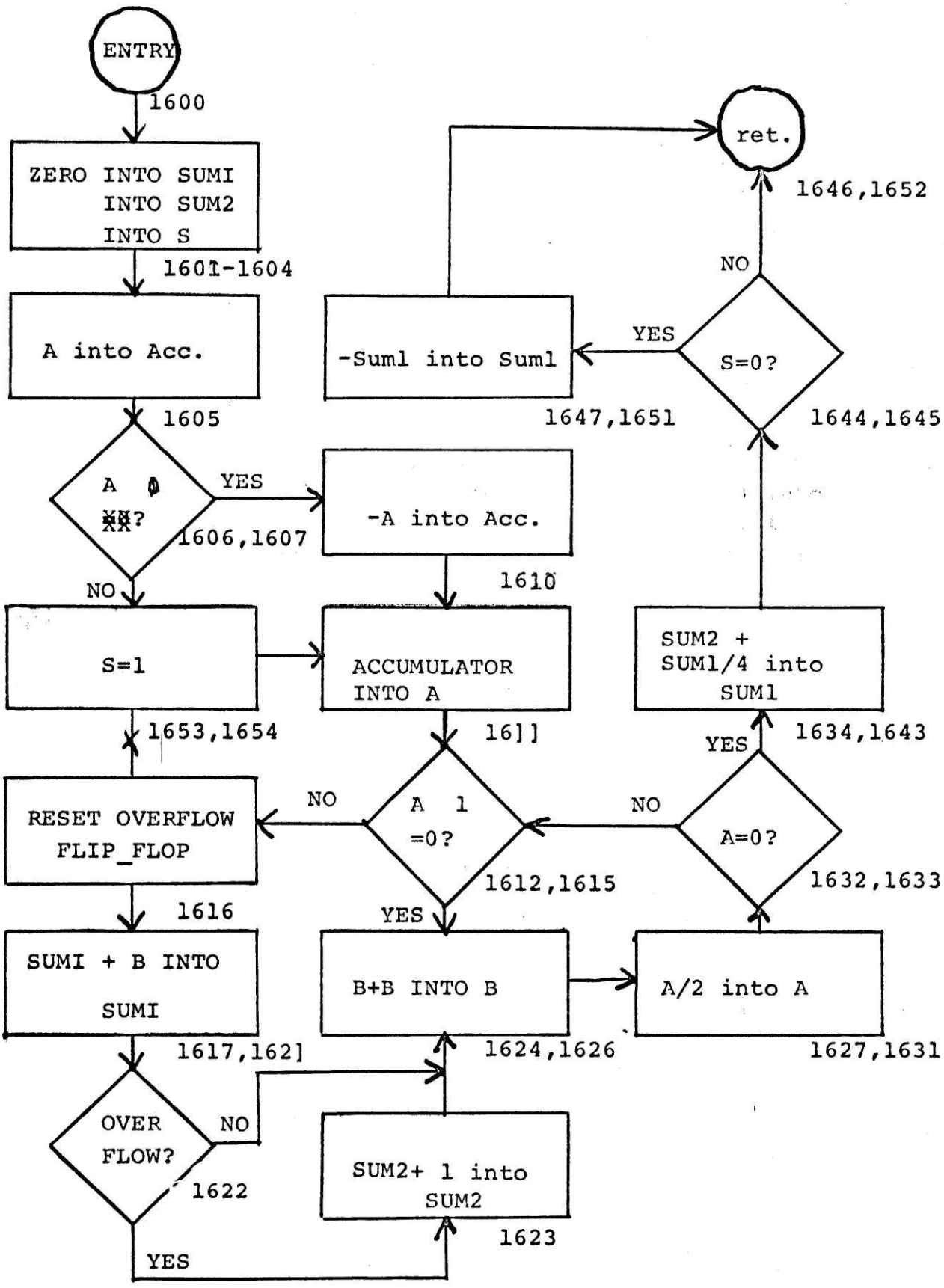
If the operator types "c" during the OUTPUT CONTROL routine control will be transferred to CALIBRATION. CALIBRATION Will rewspond "--CAL?" The user may type "L","R","U","D",".", signifying left,right,up,down, or quit respectively. He may also type "N", signifying null. After each subroutine has been executed, control will be returned to CALIBRATION ( except quit)

With the subject looking towards a 0,0 point, the operator should first type "N". The computer will store the actual coordinates of the subject's direction, to be used to shift

EYE DIRECTION



MULTIPLY

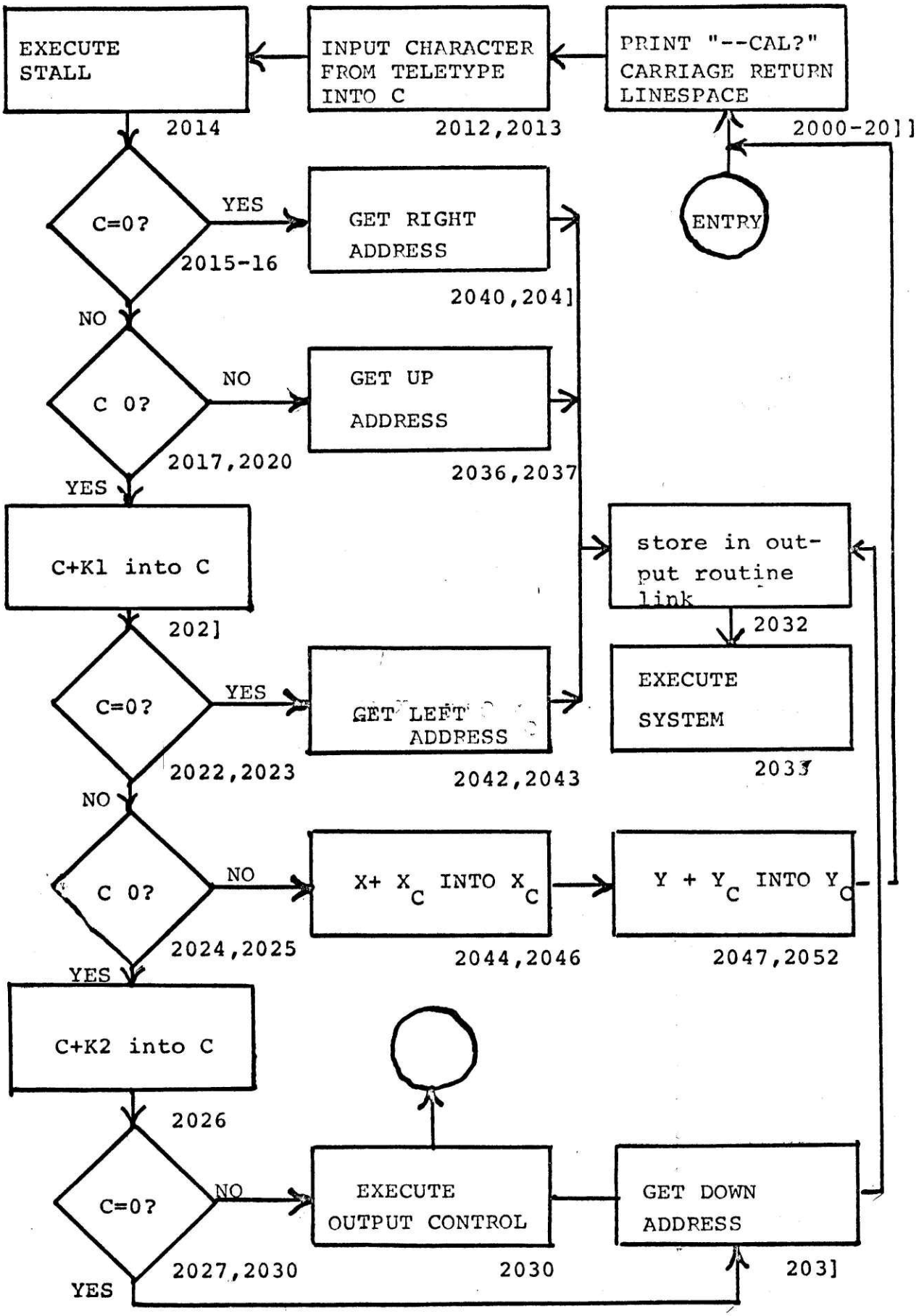

ENTRY
1600
ZERO INTO SUMI
INTO SUM2
INTO S
1601-1604
A into Acc.
1605
A 0
XX?
YES
1606,1607
-A into Acc.
1610
NO
S=1
1653,1654
ACCUMULATOR
INTO A
16]]
RESET OVERFLOW
FLIP_FLOP
1616
SUMI + B INTO
SUMI
1617,162]
OVER
FLOW?
NO
1622
SUM2+ 1 into
SUM2
1623
YES
A 1
=0?
NO
1612,1615
YES
B+B INTO B
1624,1626
A=0?
NO
1632,1633
YES
A/2 into A
1627,1631
SUM2 +
SUM1/4 into
SUM1
YES
1634,1643
S=0?
YES
1644,1645
-Sum1 into Sum1
1647,1651
NO
ret.
1646,1652

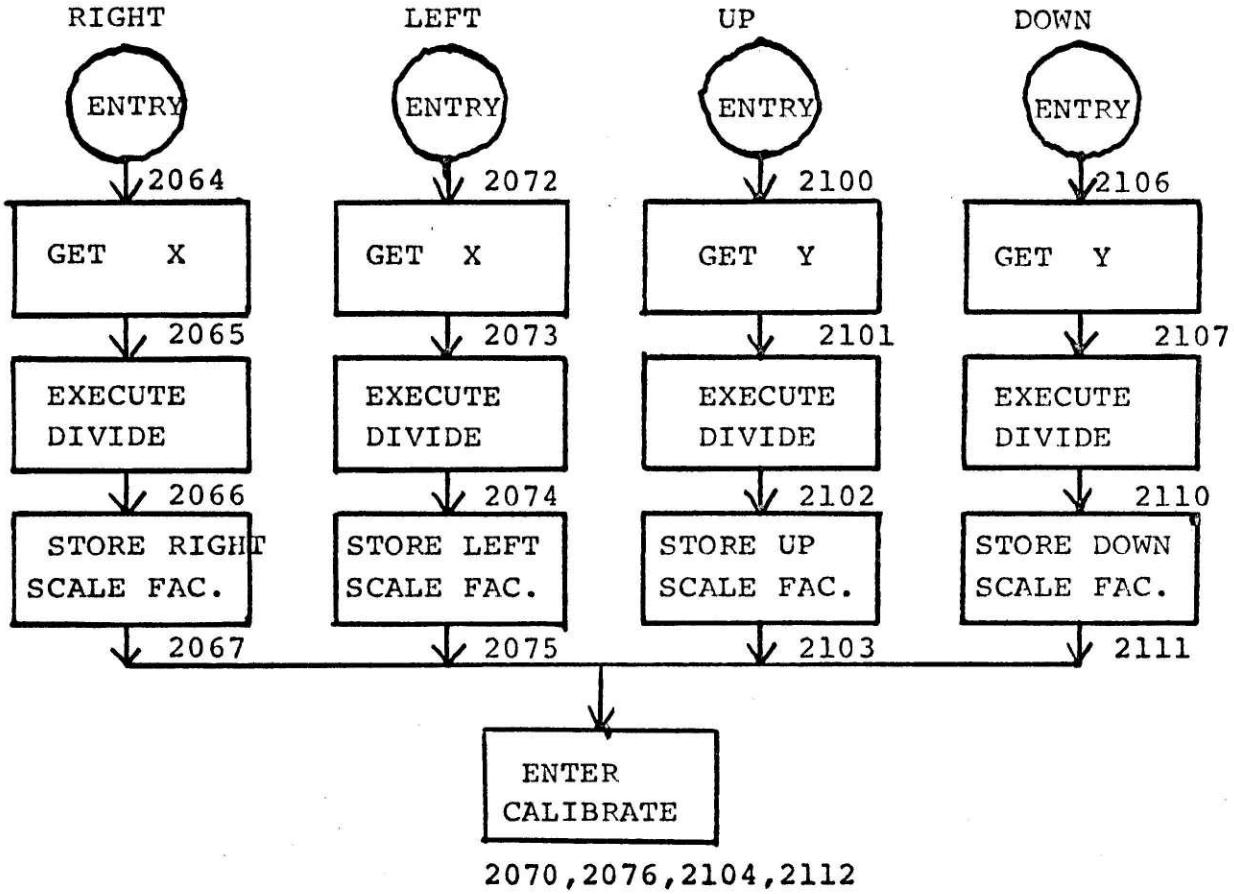
all subsequent readings by that amount. Next the right,left, up, or down scale factors may be computed. If the operator would like to relate a $20^{o}$ displacement with a 5 volt output, then he simply should have the subject look $20^{o}$ left,or right or up or down and press the appropriate key. The computer will store as the scale factor (5 volt digital equivalent/(Actual reading minus appropriate null constant)).

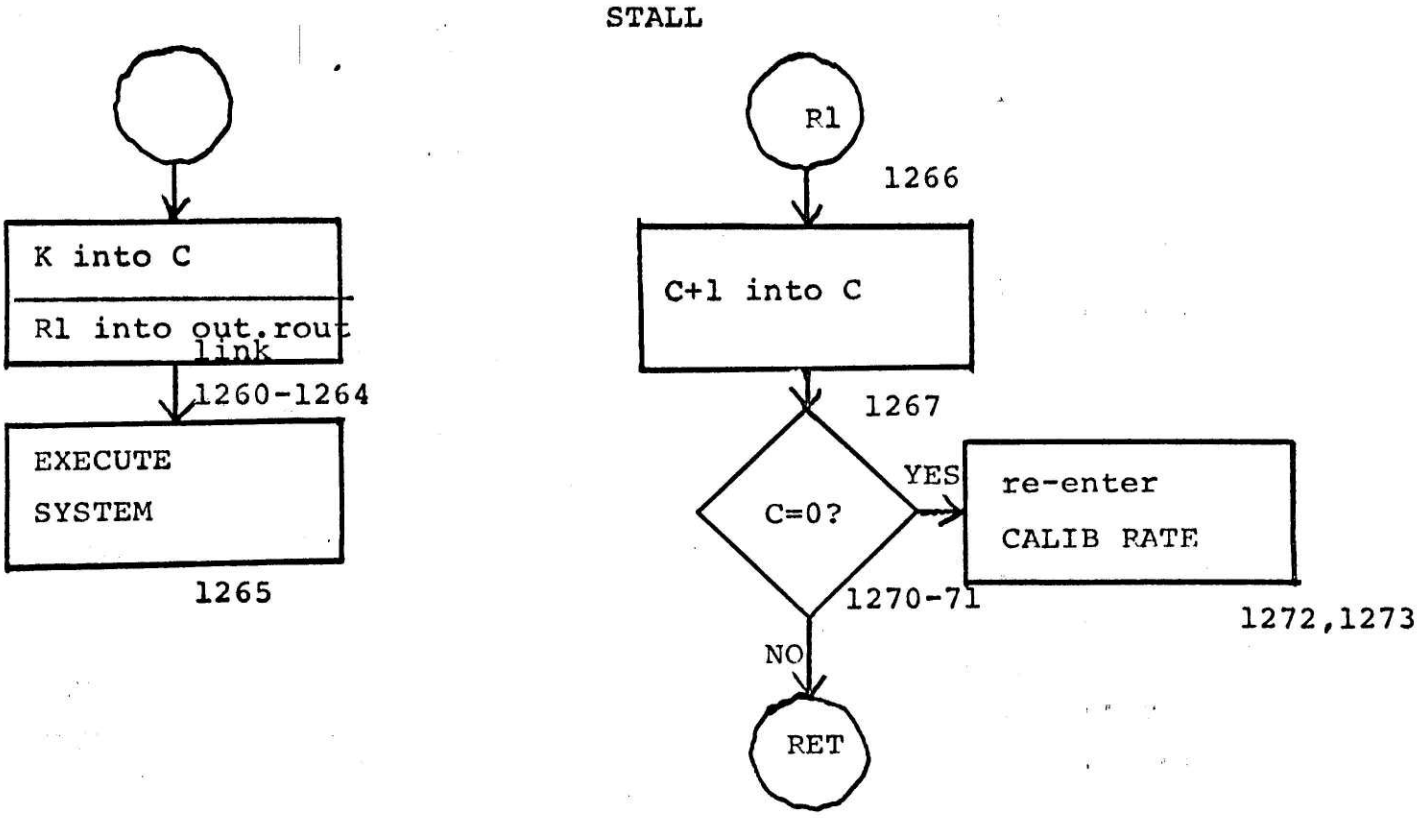
After the appropriate key is depressed the computer first stalls about 20 frames. This is done to give the DIGITAL FILTERS time to settle upon a value.

The CALIBRATION routine has never really worked satisfactorily. The principal problems seem to be that the output spot cannot be made to behave steadily without additional filtering. Yet more filtering slows down the responsiveness of the system.

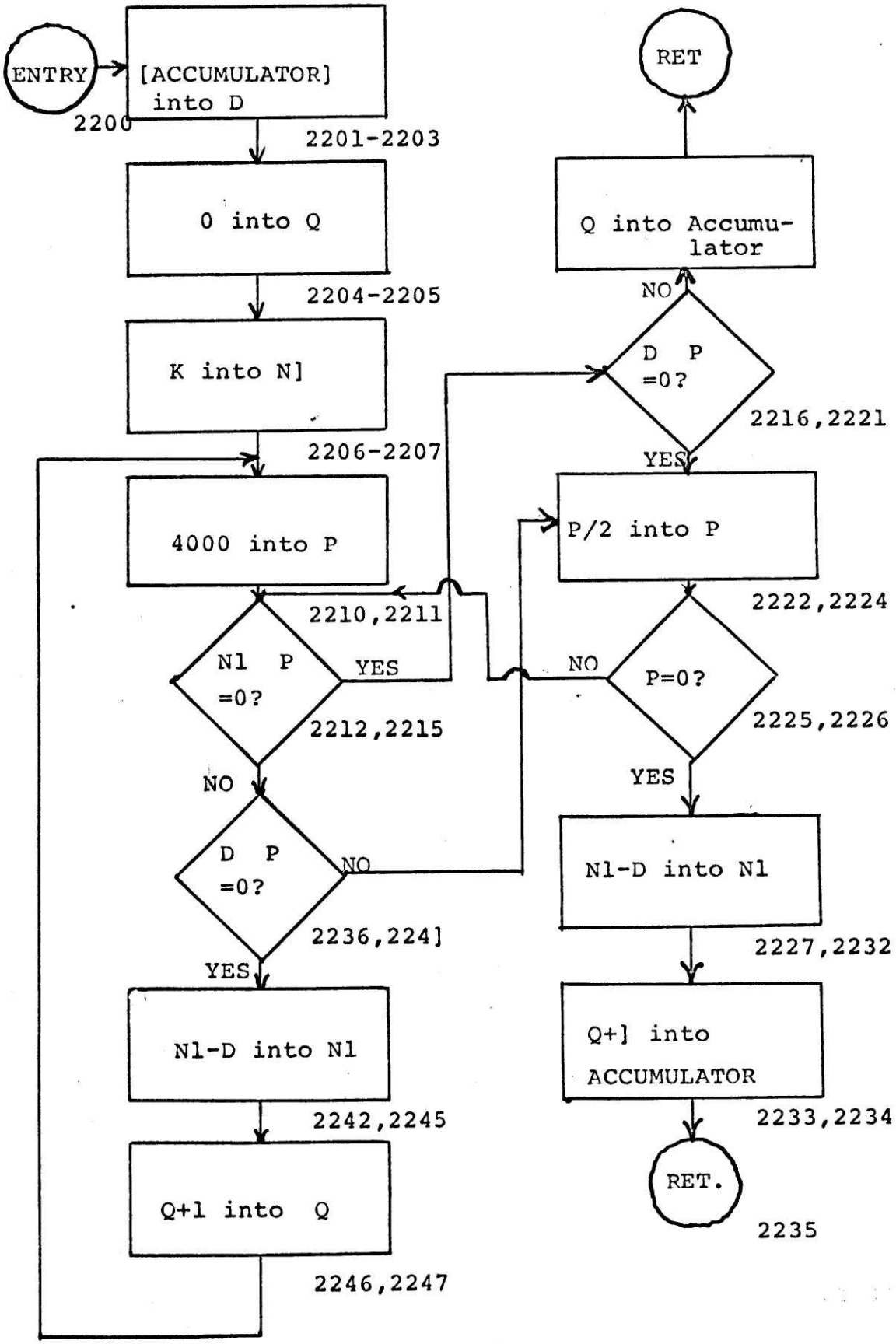
CALIBRATE


EXECUTE STALL
2014
INPUT CHARACTER FROM TELETYPE INTO C
2012,2013
PRINT "--CAL?" CARRIAGE RETURN LINESPACE
2000-20]]
ENTRY
C=0?
YES
2015-16
GET RIGHT ADDRESS
2040,204]
NO
C 0?
NO
2017,2020
GET UP ADDRESS
2036,2037
YES
C+K1 into C
202]
store in output routine link
2032
C=0?
YES
2022,2023
GET LEFT ADDRESS
2042,2043
EXECUTE SYSTEM
2033
NO
C 0?
NO
2024,2025
X+ $X_C$ INTO $X_C$
2044,2046
Y + $Y_C$ INTO $Y_C$
2047,2052
YES
C+K2 into C
2026
C=0?
NO
2027,2030
EXECUTE OUTPUT CONTROL
2030
GET DOWN ADDRESS
203]
YES


RIGHT
ENTRY
2064
GET X
2065
EXECUTE DIVIDE
2066
STORE RIGHT SCALE FAC.
2067
LEFT
ENTRY
2072
GET X
2073
EXECUTE DIVIDE
2074
STORE LEFT SCALE FAC.
2075
UP
ENTRY
2100
GET Y
2101
EXECUTE DIVIDE
2102
STORE UP SCALE FAC.
2103
DOWN
ENTRY
2106
GET Y
2107
EXECUTE DIVIDE
2110
STORE DOWN SCALE FAC.
2111
ENTER CALIBRATE
2070,2076,2104,2112


STALL
K into C
R1 into out.rout
link
1260-1264
EXECUTE SYSTEM
1265
R1
1266
C+1 into C
1267
C=0?
YES
re-enter CALIB RATE
1270-71
1272,1273
NO
RET

DIVIDE (K/ACCUMULATOR)



# APPENDIX A.

The following pages contain a complete listing of the software system used in the H112 for the Oculometer. Each line of code is written in the following format:

#1 #2 A1 A2 #3-#4 - A3

where

#1- core location of this particular instruction or constant

#2- Contents of location

A1- mnemonic for instruction; from H112 assembler

A2- Either blank or *; blank signifies direct memory reference
* signifies that the memory reference is to the contents of an address which is the contents of the memory location referenced

#3- either 0 or ]. 0 signifes that the memory reference is to the 0 sector. A 1 signifies that the reference is to a location in the same sector of memory as the instruction

#4- location of memory referred to by instruction

A3- a description of the instruction

The coding is organized in the following format:


I. SYSTEM 63

ADDRESS SWITCH] 64

ADDRESS SWITCH2 64

INTERRUPT 65

II. PROCESSING 66

DATACHEK 68

LONG CHORD 72

CXAV 73

PXAV 74

CONVERT 76

DIGITAL FILTERS 77

NULL CONSTANT 78

III. OUTPUT CONTROL 79

EYE IMAGE 80

TELETYPE OUTPUT 83

EYE DIRECTION 86

IV. CALIBRATE 90

DIVIDE 92

STALL 94

V. COMMON 95

The following instructions perform the initialization operations for the SYSTEM algorithm.

```
174    7707              -    mask constant
175     774  lda  0-74    -    get mask constant
176    4301  smk      0   -    set mask
177    4300  smk      1   -    set mask
200      60  cra          -    get zero
201    1077  sta  0-77    -    store in DIS bits
202    3621  jst  1-21    -    execute ADDRESS SWITCH1
203      44  enb          -    enable interrupts
```

The following instructions perform WAIT1.

```
204     477  lda  0-77    -    get DIS bits
205    2376  add  1-176   -    add -1
206     202  sze          -    is it zero?
207    1604  jmp  1-4     -    no- repeat loop
```

When WAIT1 is satisfied ADDRESS SWITCH1 and PROCESSING ARE applied.

```
210    3634  jst  1-34    -    yes- execute ADDRSS SWITCH1
211    3650  jst  1-50    -    execute PROCESSING
```

The following instructions perform WAIT2.

```
212     477  lda  0-77    -    get DIS bits
213    2377  add  1-177   -    add -3
214     202  sze          -    is it zero?
215    1612  jmp  1-12    -    no- repeat loop
```

When WAIT2 is satisfied ADDRESS SWITCH2 and PROCESSING are applied.

```
216     3621  jst  1-21 -     execute ADDRESS SWITCH2
217     3650  jst  1-50 -     execute PROCESSING
220     1604  jmp  104  -     enter WAIT1

                    ADDRESS SWITCH1
221                       -   entry (Store return address.)
222      746  lda  1-146-     get C2
223     5350  sta* 1-150-     store in MAO
224      744  lda  1-144-     get C1
225     1023  sta  0-23 -     store for PROCESSING
226      747  lda  1-147-     get P2
227     5351  sta* 1-151-     store in MBO
230      745  lda  1-145-     get P1
231     1024  sta  0-24 -     store for PROCESSING
232     5621  jmp* 1-21 -     return
                    ADDRESS SWITCH2
234                       -   entry ( Store return address.)
235      744  lda  1-144-     get C1
236     5350  sta* 1-150-     store in MAO
237      745  lda  1-145-     get P1
240     5351  sta* 1-151-     store in MBO
241      746  lda  1-146-     get C2
242     1023  sta  0-23 -     store for PROCESSING
243      747  lda  1-147-     get P2
244     1024  sta  0-24 -     store for PROCESSING
245     5634  jmp* 1-34 -     return
```

INTERRUPT ROUTINE

```
100                                 -
101   1314 sta  1-114   -  save accumulator
102   4101 sks      1   -  is teletype breaking?
103   1705 jmp  1-105   -  no- goto 105
104   5676 jmp* 1-76    -  yes- goto OUTPUT CONTROL routine
105   4060 ina  60      -  get bits from DIS
106   1705 jmp  1-105   -  go to 105
107    112 lgr    12    -  right shift bits ]2 ( decimal ]0)
110   1277 sta  1-77    -  store DIS bits
111    714 lda  1-114   -  restore accumulator
112     44    enb       -  enable interrupts
113   5700    jmp* 1-100- return
```

```
250                     -    entry ( Store return address.)
251    423  lda  0-23  -    get start of cornea file
252   2343  add  1-143-    add 2
253   1023  sta  0-23  -    store first cornea y address
254    424  lda  0-24  -    get start of pupil file
255   2343  add  1-143-    add 2
256   1024  sta  0-24  -    store first pupil y address
257   7431  jst* 0-31  -    execute DATACHEK
260   1663  jmp  1-63  -

263    423  lda  0-23  -    get start of cornea file
264   1011  sta  0-11  -    store for LONG CHORD routine
265   7404  jst* 0-4   -    execute LONG CHORD
266    412  lda  0-12  -    get longest cornea chord
267   1361  sta  1-161-    save it
270    424  lda  0-24  -    get start of pupil file
271   1011  sta  0-11  -    store for LONG CHORD routine
272   7407  jst* 0-4   -    execute LONG CHORD
273    412  lda  0-12  -    get longest pupil chord
274   1022  sta  0-22  -    save it
275   7405  jst* 0-5   -    execute CXAV
276   7406  jst* 0-6   -    execute PXAV
277    417  lda  0-17  -    get PXAV
300      5  tca        -    minus it
301   2015  add  0-15  -    add CXAV
302   1020  sta  0-20  -    store delta x
```

```
303     761  lda  1-161-    get longest corneal chord
304    1011  sta  0-11 -    store for CONVERT routine
305    7407  jst* 0-7  -    execute CONVERT
306     200  nop       -    no operation
307     425  lda  0-25 -    get cornea bottom value
310     151  rar    11 -    left shift three
311    1370  sta  1-170-    save it
312     426  lda  0-26 -    get pupil bottom value
313     151  rar    11 -    left shift three
314    1371  sta  1-171-    save it
315     412  lda  0-12 -    get cornea y radius from CONVERT
316       5  tca       -    minus it
317    2370  add  1-170-    add cornea bottom value
320    1366  sta  1-166-    store CYAV
321     422  lda  0-22 -    get pupil long chord
322    1011  sta  0-11 -    store for CONVERT routine
323    7407  jst* 0-7  -    execute CONVERT
324     412  lda  0-12 -    get pupil y radius
325       5  tca       -    minus it
326    2371  add  1-171-    add pupil bottom value
327    1375  sta  1-175-    store PYAV
330       5  tca       -    minus it
331    2366  add  1-166-    add CYAV
332    1021  sta  0-21 -    store delta y
333    7432  jst* 0-32 -    execute FILTERS
```

```
334     7416   jst* 0-16 -     execute NULL CONSTANT
335     7410   jst* 0-10 -     execute (OUTPUT) routine
336     5650   jmp* 1-50 -     return
```

The following routines are called by PROCESSING.

1. DATACHEK

```
600                    -       entry (Store return address.)
601     423    lda   0-23 -     get first corneal y address
602     1356   sta   1-156-    store in y_i
603      774   lda   1-174-    load consecutive line requirement (8)
604     1357   sta 1-157 -     store in kl
605     3637   jst   1-27 -    execute ANALYZE
606      770   lda   1-170-    get end of file address
607     1027   sta   0-27 -    store in corneal y bottom address
610      771   lda   1-171-    get new start of file address
611     1023   sta   0-23 -    store as corneal file start address
612      767   lda   1-167-    get y bottom value
613     1025   sta   0-25 -    store as corneal y bottom value
614      424   lda   0-24 -    get pupil first y address
615     1356   sta   1-156-    store in y_i
616      773   lda   1-173-    load consecutive line requirement (2)
617     1357   sta   1-157-    store in kl
620     3627   jst   1-27 -    execute ANALYZE
621      770   lda   1-170-    get end of file address
622     1030   sta   0-30 -    store in pupil y bottom address
623      771   lda   1-171-    get new start of file address
624     1024   sta   0-24 -    store in pupil start of file address
625      767   lda   1-167-    get bottom y value
```

```
626    1662  jmp   1-62  -
662    1026  sta   0-26  -     store pupil y bottom value
663    5600  jmp*  1-0   -     return
```

ANALYZE

```
627                      -     entry ( Store return address.)
630     757  lda   1-157-      get k1
631    2357  add   1-157-      + k1
632    2357  add   1-157-      + k1
633    2375  add   1-175-      + 2
634    1360  sta   1-160-      store as k2
635      60  cra       -       clear accumulator
636    1367  sta   1-167-      store in ylast
637     762  lda   1-162-      get 1
640    1361  sta   1-161-      store in C (Comparison counter)
641    4756  lda*  1-156-      get y  value
                                   i
642    2366  add   1-166-      add end of file checking constant
643     302  snz       -       is result zero?
644    5414  jmp*  0-14  -     yes- terminate PROCESSING
645    4756  lda*  1-156-      no-get y  value
                                      i
646       5  tca       -       minus it
647    2367  add   1-167-      add ylast value
650     302  snz       -       is result nonzero?
651    1664  jmp   1-64  -     no- jump to 664
652    2362  add   1-162-      add 1
653     302  snz       -       is result zero?
```

```
654    1677  jmp  1-77 -    yes jump to 677
655     762  lda  1-162-    no-load 1
656    1361  sta  1-161-    store in C
657    4756  lda* 1-156-    get y  value
                                 i
660    1367  sta  1-167-    store in ylast
661    1750  jmp  1-150-    goto 750
662    1026  sta  0-26 -    store pupil y bottom value
663    5600  jmp* 1-0  -    return
664      60  cra       -    zero into accumulator
665    5356  sta* 1-156-    store in y  value
                                      i
666     756  lda  1-156-    get y  address
                                 i
667    2363  add  1-163-    add -3
670    1356  sta  1-156-    store y  address
                                   i
671      60  cra       -    zero into accumulator
672    5356  sta* 1-156-    store in y  value
                                      i
673     756  lda  1-156-    get y  address
                                 i
674    2365  add  1-165-    add 6
675    1356  sta  1-156-    store y  address
                                   i
676    1641  jmp  1-41 -    goto 640
677     757  lda  1-157-    get k1
700       5  tca       -    minus it
701    2361  add  1-161-    add C
702     202  sze       -    is result zero?
703    1746  jmp  1-146-    no- jmp to 746
704     760  lda  1-160-    yes- get k2
705       5  tca       -    minus it
```

```
706    2356  add   1-156-    add y  address
                              i
707    1371  sta   1-171-    store new file start address
710    4756  lda*  1-156-    get y  value
                              i
711    1367  sta   1-167-    store in ylast value
712     756  lda   1-156-    get y  address
                              i
713    2364  add   1-164-    + 3
714    1356  sta   1-156-    store in y  address
                                       i
715    4756  lda*  1-156-    get y  value
                              i
716       5  tca         -   minus it
717    2367  add   1-167-    add the value of ylast
720     302  snz         -   is the result zero?
721    1731  jmp   1-131-    yes-jump to 731
722    2362  add   1-162-    + 1
723     302  snz         -   is the result zero?
724    1710  jmp   1-110-    yes- go to 710
725     756  lda   1-156-    get y  address (no)
                              i
726    2363  add   1-163-    + (-3)
727    1370  sta   1-170-    store in y bottom address
730    5627  jmp*  1-27  -   return
731
732
733      60  cra         -   zero into accumulator
734    5356  sta*  1-156-    store in y  value
                                       i
735     756  lda   1-156-    get y  address
                              i
736    2363  add   1-163-    + (-3)
737    1356  sta   1-156-    store in y  address
                                       i
740      60  cra         -   zero into accumulator
```

| | | | | |
|---|---|---|---|---|
| 741 | 5356 | sta* | 1-156- | store in $y_i$ value |
| 742 | 756 | lda | 1-156- | get $y_i$ address |
| 743 | 2365 | add | 1-165- | +6 |
| 744 | 1356 | sta | 1-156- | store $y_i$ address |
| 745 | 1715 | jmp | 1-115- | goto 715 |
| 746 | 3361 | irs | 1-161- | C=C+1 |
| 747 | 1657 | jmp | 1-57 - | goto 657 |
| 750 | 756 | lda | 1-156- | get $y_i$ address |
| 751 | 2364 | add | 1-164- | +3 |
| 752 | 1356 | sta | 1-156- | store $y_i$ address |
| 753 | 1641 | jmp | 1-41 - | goto 641 |

754-761 Temporary Storage

762-765 Constants

773-774 Constants

## 2. LONG CHORD

| | | | | |
|---|---|---|---|---|
| 400 | | | - | entry ( Store return address.) |
| 401 | 411 | lda | 0-11 - | get data file starting address |
| 402 | 1344 | sta | 1-144- | store pointer (P) |
| 403 | 60 | cra | - | zero into accumulator |
| 404 | 1012 | sta | 0-12 - | set max chord to zero (MC) |
| 405 | 4744 | lda* | 1-144- | get value of pointer (x1) |
| 406 | 5 | tca | - | minus it |
| 407 | 3344 | irs | 1-144- | increment P |
| 410 | 6344 | add* | 1-144- | add value of P |
| 411 | 1345 | sta | 1-145- | store current chord |
| 412 | 3344 | irs | 1-144- | increment P |
| 413 | 4744 | lda* | 1-144- | get value of P ($y_i$) |

```
414     302   snz        -    is it zero?
415    1630   jmp   1-30  -    yes- goto430
416    2365   add   1-165-    no-add end of file check constant
417     302   snz        -    is result zero?
420    5600   jmp*  1-0   -    yes- return
421     745   lda   1-145-    no- getcurrent chord
422       5   tca        -    minus it
423    2012   add   0-12  -    add MC
424     203   smz        -    is result less than or equal to zero?
425    1632   jmp   1-32  -    no- goto 432
426     745   lda   1-145-    yes- get current chord
427    1012   sta   0-12  -    store in MC
430    3344   irs   1-144-    increment P
431    1605   jmp   1-5   -    next line
432    2347   add   1-147-    add -2
433     203   smz        -    is result less than or equal to zero?
434    5600   jmp*  1-0   -    return(yes)
435    1630   jmp   1-30  -    no- get next line
```

3. CXAV

```
446                       -    entry ( Store return address.)
447    4423   lda*  0-23  -    get x1
450    3023   irs   0-23  -    increment pointer
451    6023   add*  0-23  -    add x2
```

```
452    1345  sta   1-145-    store sum
453    3023  irs   0-23 -    increment pointer
454    4423  lda*  0-23 -    get y value
455    3023  irs   0-23 -    increment pointer
456     302  snz         -   is y value zero?
457    1647  jmp   1-47 -    yes- go to 447
460     745  lda   1-145-    no- load sum
461    6023  add*  0-23 -    add x1
462    3023  irs   0-23 -    increment pointer
463    6023  add*  0-23 -    add x2
464    1360  sta   1-160-    store sum1
465    3023  irs   0-23 -    increment pointer
466    4423  lda*  0-23 -    get y value
467    3023  irs   0-23 -    increment pointer
470     302  snz         -   is it  zero?
471    1660  jmp   1-60 -    yes- go to 460
472     760  lda   1-160-    no- get sum1
473    1015  sta   0-15 -    store CXAV
474    5646 jmp*  1-46 -    return
```

4.PXAV

```
500                    -    entry ( Store return address.)
501      60  cra       -    zero into accumulator
502    1363  sta   1-163-    store sum1
503    1364  sta   1-164-    store sum1
504     752  lda   1-152-    initialize line counter to -8
```

```
505    1353  sta  1-153-    store in k
506    430   lda  0-30 -    get bottom y address
507    1360  sta  1-160-    store in k1
510    2351  add  1-151-    add -1
511    1362  sta  1-162-    store in x2
512    2351  add  1-151-    add -1
513    1361  sta  1-161-    store in x1
514    4760  lda* 1-160-    get k1
515     302  snz       -    is it zero?
516    1735  jmp  1-135-    yes- goto 535
517      30  toa       -    clear overflow bit
520    4762  lda* 1-162-    get x2 value
521    6361  add* 1-161-    add x1 value
522    2363  add  1-163-    add sum1 value
523    1363  sta  1-163-    store result in sum1
524     304  sno       -    was there an overflow?
525    1740  jmp  1-140-    yes goto 541
526    3353  irs 1-153 -    no-increment line counter
527    1735  jmp  1-135-    goto 535 ( if line counter is neg.)
530     763  lda  1-163-    get sum1
531     102  lgr       2-   divide by 4
532    2364  add  1-164-    add sum2
533    1017  sta  0-17 -    store PXAV
534    5700  jmp* 1-100-    return
535     761  lda  1-161-    get x1 address
536    2351  add  1-151-    add -1
```

```
537     1707   jmp   1-107-     goto 507
540      757   lda   1-157-     get 2000
541     1364   sta   1-164-     store in sum2
542     1726   jmp   1-126-     goto 526
543-565 constants
```

5. CONVERT

```
566                      -      entry ( Store return address.)
567     411    lda   0-11 -     get x counts
570     153    rar     13 -     left shift 1
571     1344   sta   1-144-     store 2x
572      103   lgr      3 -     right shift 3
573     2344   add   1-144-     add 2x
574     1012 sta 0-12 -          store result (2x+x/4)
575     5766   jmp*  1-166-     return
```

```
2400                    -    entry
2401    636   lda  1-36 -    get Xold
2402   2637   ana  1-37 -    logical and with 7
                                                8
2403   1235   sta  1-35 -    store in F1
2404    636   lda  1-36 -    get Xold
2405    123   ars    3  -    divide by eight
2406   1234   sta  1-34 -    store in  I1
2407    640   lda  1-40 -    get -7
2410   1232   sta  1-32 -    store in C1
2411   1233   sta  1-33 -    store in C2
2412    420   lda  0-20 -    get delta X
2413    123   ars    3  -    divide by eight
2414   2234   add  1-34 -    add I1
2415   3232   irs  1-32 -    increment C1
2416   1614   jmp  1-14 -    if C1 is negative go to 2414
2417   1234   sta  1-34 -    if C1 is zero store acc.in I1
2420    420   lda  0-20 -    get . delta x
2421   2637   ana  1-37 -    logical and with 7
                                                8
2422   2235   add  1-35 -    add F1
2423   3233   irs  1-33 -    increment C2
2424   1622   jmp  1-22 -    if C2 is negative go to 2422
2425    123   ars    3  -    if C2 is zero divide acc. by eight.
2426   2234   add  1-34 -    add I1
2427   1236   sta  1-36 -    store in Xold
2430   1020   sta  0-20 -    store in delta x
```

```
2431    1641    jmp  1-41 -    goto 2441

2441     640    lda  1-40 -    get -7
2442    1232    sta  1-32 -    store in C1
2443     421    lda  0-21 -    get delta y
2444    2253    add  1-53 -    add Yold
2445    3232    irs  1-32 -    increment C1
2446    1644    jmp  1-44 -    if C1 is negative go to 2444
2447     123    ars     3 -    if C1 is zero divide by eight
2450    1253    sta  1-53 -    store in Yold
2451    1021    sta  0-21 -    store in delta y
2452    5600    jmp* 1-0  -    return
2432-2440       Constants and Temporary Storage
```

NULL CONSTANT ROUTINE

```
1310                      -    entry
1311     437    lda  0-37 -    get X null constant
1312       5    tca       -    minus it
1313    2020    add  0-20 -    add delta x
1314    1020    sta  0-20 -    store in delta x
1315     440    lda  0-40 -    get Y null constant
1316       5    tca       -    minus it
1317    2021    add  0-21 -    add delta y
1320    1021    sta  0-21 -    store in delta y
1321    5710    jmp* 1-110-    return
```

OUTPUT CONTROL

```
1200                -    entry ( Store return address.)
1201    :632  lda   1-32 -    get linespace character
1202    4202  ota   2    -    output to teletype
1203    1602  jmp   1-2  -
1204     633  lda   1-33 -    get carriage return character
1205    4202  ota   2    -    output to teletype
1206    1605  jmp   1-5  -
1207     657  lda   1-57 -    get rubout character
1210    4202  ota   2    -    output to teletype
1211    1610  jmp   1-10 -
1212    5600  jmp*  1-0  -    return

1215    2235  add   1-35 -    add -2
1216     302  snz        -    is result zero?
1217    1630  jmp   1-30 -    yes- go to 1230
1220     201  smi        -    no- is result minus?
1221    1626  jmp   1-26 -    no- go to 1226
1222     640  lda   1-40 -    yes-get teletype routine link
1223    1010  sta   0-10 -    store in output routine link
1224    3600  jst   1-0  -    execute carriage control rout.(1200)
1225    5414  jmp*  0-14 -    begin SYSTEM algorithm
1226     636  lda   1-36 -    get eye direction link
1227    1623  jmp   1-23 -    go to 1223
1230     637  lda   1-37 -    get eye image link
1231    1623  jmp   1-23 =    go to 1223
```

1232-1240 Constants and Temporary Storage

```
1241   3600  jst  1-0  -    execute carriage control subrout.
1242    655  lda  1-55 -    get question mark character
1243   4202  ota  2    -    output to teletype
1244   1643  jmp  1-43 -    go to 1243
1245   4001  ina  1    -    input character from teletype
1246   1645  jmp  1-45 -    go to 1245
1247   2256  add  1-56 -    add -300
1250    201  smi       -    is result negative?
1251   5654  jmp* 1-54 -    no- execute CALIBRATE
1252   2634  ana  1-34 -    yes- select units digit
1253   1615  jmp  1-15 -    go to 1215
1254-1257 Constants
```

1. EYE IMAGE

This routine is entered at location 1033.

```
1000                  -    entry
1001   774   lda  1-174-    get file start address
1002   1344  sta  1-144-    store in x_i
1003   2350  add  1-150-    add 2
1004   1345  sta  1-145-    store in y_i
1005   3616  jst  1-16 -    execute subroutine
1006    774  lda  1-174-    get file start address
1007   2347  add  1-147-    add 1
1010    200  nop       -    no operation
1011   1344  sta  1-144-    store in x_i
1012   2347  add  1-147-    add 1
1013   1345  sta  1-145-    store in y_i
1014   3616  jst  1-16 -    execute subroutine
```

```
1015   5600   jmp* 1-0  -    return
1016                   -    entry
1017   4744   lda* 1-144-    get value of x
                                          i
1020   2352   add  1-152-    add end of file check constant
1021    302   snz         -  is result zero?
1022   5616 jmp*  1-16  -    yes- return
1023   3644   jst  1-44  -    no-execute subroutine
1024    744   lda  1-144-    get x  address
                                 i
1025   2351   add  1-151-    add 3
1026   1344   sta  1-144-    store in x  address
                                       i
1027    745   lda  1-145-    get y  address
                                 i
1030   2351   add  1-151-    add 3
1031   1345   sta  1-145-    store in y  address
                                       i
1032   1617   jmp  1-17  -    go to 1017

1033                   -    entry ( Store return address.)
1034    423   lda  0-23  -    get corneal file start address
1035   1374   sta  1-174-    store in file start address
1036   3600   jst  1-0   -    execute subroutine
1037    424   lda  0-24  -    get pupil file start address
1040   1374   sta  1-174-    store in file start address
1041   3600   jst  1-0   -    execute subroutine
1042   5633   jmp* 1-33  -    return
1044                   -    entry
1045     50   inh         -  inhibit interrupts
1046    755   lda  1-155-    get channel 3 address
1047   4216   ota  16    -    open d/a channel 3
1050    200   nop         -  no operation
```

```
1051    746  lda  1-146-    output "low" on channel 3
1052   4217  ota  17   -
1053    200  nop       -
1054    753  lda  1-153-    get channel 1 address
1055   4216  ota  16   -    open channel 1
1056    200  nop       -
1057   1715  jmp  1-115-    go to 1115
1060   4217  ota  17   -    output on channel 1
1061    200  nop       -
1062    754  lda  1-154-    get channel 2 address
1063   4216  ota  16   -    open channel 2
1064    200  nop       -
1065   1720  jmp  1-120-    go to 1120
1066   4217  ota  17   -    output on channel 2
1067    200  nop       -
1070    755  lda  1-155-    get channel 3 address
1071   4216  ota  16   -    open channel 3
1072    200  nop       -
1073    756  lda  1-156-    get "high" signal
1074   4217  ota  17   -    output on channel 3
1075    200  nop       -
1076     44  enb       -    enable interrupts
1077   5644  jst* 1-44 -    return
1115   4744  lda* 1-144-    get value of x
                                          i
1116    152  rar  12   -    left shift two
1117  16660  jmp  1-60 -    go to 1060
```

```
1120    4745  lda* 1-145-    get y  value
                                  i
1121     152  rar  12   -    left shift two
1122    1666  jmp  1-66 -    go to 1066
```

1144-1177 Constants and Temporary Storage

## 2.TELETYPE OUTPUT

```
1400                   -    entry ( Store return address.)
1401     423  lda  0-23 -    get cornea file start address
1402    1301  sta  1-101-    store in current address pointer(P)
1403    3610  jst  1-10 -    execute subroutine1
1404     424  lda  0-24 -    get pupil file start address
1405    1301  sta  1-101-    store in current address pointer(P)
1406    3610  jst  1-10 -    execute subroutine
1407    5600  jmp* 1-0  -    return
1410                   -    entry1
1411     676  lda  1-76 -    get carriage return character
1412    4202  ota  2    -    output to teletype
1413    1613  jmp  1-13 -
1414     677  lda  1-77 -    get linespace character
1415    4202  ota  2    -    output to teletype
1416    1615  jmp  1-15 -
1417    4701  lda* 1-101-    get value of P
1420    2274  add  1-74 -    add end of file constant check
1421     302  snz       -    is result zero?
1422    5610  jmp* 1-10 -    yes-return
1423     675  lda  1-75 -    get -3
```

```
1424   1300   sta   1-100-    store in word counter (C)
1425   3632   jst   1-32 -    execute subroutine3
1426   3301   irs   1-101-    increment P
1427   3300   irs   1-100-    increment C
1430   1625   jmp   1-25 -    if C is negative go to 1425
1431   1611   jmp   1-11 -    go to 1411
1432                  -       entry3
1433   4701   lda*  1-101-    get value of P
1434   2673   ana   1-73 -    select most sig. octal bit
1435    111   lgr   9    -    right justify
1436   2267   add   1-67 -    convert to ASCII
1437   4202   ota   2    -    output to teletype
1440   1637   jmp   1-37 -
1441   4701   lda*  1-101-    get value of P
1442   2672   ana   1-72 -    select next most sig. octal bit
1443    106   lgr   6    -    right justify
1444   2267   add   1-67 -    convert to ASCII
1445   4202   ota   2    -    output to teletype
1446   1645   jmp   1-45 -
1447   4701   lda*  1-101-    get value of P
1450   2671   ana   1-71 -    select third most sig. bit
1451    103   lgr   3    -    right justify
1452   2267   add   1-67 -    convert to ASCII
1453   4202   ota   2    -    output to teletype
1454   1653   jmp   1-53 -
1455   4701   lda*  1-101-    get value of P
```

```
1456    2670   ana   1-70 -     select lest sig. octal bit
1457    2267   add   1-67 -     convert to ASCII
1460    4202   ota   2    -     output to teletype
1461    1660   jmp   1-60 -
1462     666   lda   1-66 -     get space character
1463    4202   ota   2    -     output to teletype
1464    1663   jmp   1-63 -
1465    5632   jmp*  1-32 -     return3
```

1466-1501 Constants and Temporary Storage

## 3. EYE DIRECTION

```
1700                    -    entry
1701   420   lda  0-20  -    get delta x
1702  1260   sta  1-60  -    store for MULTIPLY
1703   201   smi        -    is accumulator less than zero?
1704  1711   jmp  1-111-    no- go to 1711
1705   433   lda  0-33  -    get right scale factor (yes)
1706  1257   sta  1-57  -    store for MULTIPLY
1707  3600   jst  1-0   -    execute MULITPLY
1710  1725   jmp  1-125-    go to 1725
1711   434   lda  0-34  -    get left scale factor
1712  1706   jmp  1-106-    go to 1706
1713   42]   lda  0-2]  -    get delta y
1714  1260   sta  1-60  -    store for MULTIPLY
1715   201   smi        -    is accumulator less than zero?
1716  1723   jmp  1-123-    go to 1723
1717   436   lda  0-36  -    get up scale factor
1720  1257   sta  1-57  -    store for MULTIPLY
1721  3600   jst  1-0   -    execute MULTIPLY
1722  1730   jmp  1-130-    go to 1730
1723   435   lda  0-35  -    get down scale factor
1724  1720   jmp  1-120-    go to 1720
1725   661   lda  1-61  -    get result of x mulitply
1726  1020   sta  0-20  -    store in delta x
1727  1713   jmp  1-113-    go to 1713
1730   661   lda  1-61  -    get result of y multiply
```

```
1731  1021   sta  0-21 -    store result of y multiply
1732  7764   jst* 1-164-    execute ANALOG OUT
1733  5700   jmp* 1-100-    return

                    ANALOG OUT

1330                  -     entry
1331   754   lda  1-154-    get channel 1 address
1332  4216   ota  ]6   -    open channel 1
1333  1732   jmp  1-132-    go to 1332
1334   420   lda  0-20 -    get delta x
1335  4217   ota  17   -    output on channel 1
1336  1735   jmp  1-135-    go to 1335
1337   755   lda  1-155-    get  channel 2 address
1340  4216   ota  16   -    open channel 2
1341  1740   jmp  1-140-    go to 1340
1342   421   lda  0-2] -    get delta y
1343  4217   ota  17   -    output on channel 2
1344  1743   jmp  1-143-    go to ]343
1345   756   lda  1-156-    get channel 3 address
1346  4216   ota  16   -    open channel 3
1347  1746   jmp  1-146-    go to 1346
1350   422   lda  0-22 -    get pupil diameter
1351  4217   ota  17   -    output on channel 3
1352  1751   jmp  1-151-    go to 1351
1353  5730   jmp* 1-130-    return
]354-]356 Constants and Temporary Storage
```

MULTIPLY

```
1600                -          entry
1601    60    cra       -      zero into accumulator
1602  1261    sta  1-61 -      zero into suml
1603  1262    sta  1-62 -      zero into sum2
1604  1263    sta  1-63 -      zero into S
1605   660    lda  1-60 -      get A
1606   201    smi       -      is A negative?
1607  1653    jmp  1-53 -      no-go to 1653
1610     5    tca       -      yes- minus it
1611  1260    sta  1-60 -      store int A
1612   660    lda  1-60 -      get A
1613  2655    ana  1-55 -      select l.s.b.
1614   303    spn       -      is it one?
1615  1624    jmp 1-24  -      go to 1624
1616    30    toa       -      zero in overflow flop
1617   657    lda  1-57 -      get B
1620  2261    add  1-61 -      add suml
1621  1261    sta  1-61 -      store suml
1622   304    sno       -      overflow?
1623  3262    irs  1-62 -      yes- sum2+1 into sum2
1624   657    lda  1-57 -      get B  (no)
1625  2257    add  1-57 -      add B
1626  1257    sta  1-57 -      store B
1627   660    lda  1-60 -      get A
1630   101    lgr  1    -      divide by 2 (right shift one)
```

```
1631  1260   sta  1-60 -    store A
1632   202   sze       -    is A zero?
1633  1613   jmp  1-13 -    no-go to 1613
1634   661   lda  1-61 -    yes- get suml
1635   102   lgr  2    -    divide by 4
1636  1261   sta  1-61 -    store suml
1637   662   lda  1-62 -    get sum2
1640  2656   ana  1-56 -    select two least significant bits
]64]   ]42   rar  2    -    left shift 10(10)
1642  2261   add  1-61 -    add suml
1643  1261   sta  1-61 -    store in suml
1644   663   lda  1-63 -    get S
1645   202   sze       -    is S zero?
1646  5600   jmp* 1-0  -    no-return
1647   661   lda  1-61 -    yes-  get suml
1650     5   tca       -    minus it
]651  1261   sta  1-61 -    store suml
1652  5600   jmp* 1-0  -    return
1653  3263   irs  1-63 -    incremtn S
1654  1612   jmp  1-12 -    goto 1612
```

1655-1663 Constants and temporary storage

CALIBRATE

```
2000    737  lda  1-137-    get number of characters
2001   1363  sta  1-163-    store in character counter
2002    746  lda  1-146-    get start of message address
2003   1364  sta  1-164-    store in message address counter
2004   4764  lda* 1-164-    get character
2005   4202  ota  2     -   output to teletype
2006   1605  jmp  1-5   -   go to 2005
2007   3364  irs  1-164-    next character
2010   3363  irs  1-163-    increment character counter
2011   1604  jmp  1-4   -   if char. counter is neg go to 2004
2012   4001  ina  1     -   receive character from teletype
2013   1612  jmp  1-12  -   go to 2012
2014   5770  jmp* 1-170-    go to STALL routine
2015    302  snz        -   is result zero?
2016   1640  jmp  1-40  -   yes- go to 2040
2017    201  smi        -   is result minus?
2020   1636  jmp  1-36  -   no- go to 2036
2021   2342  add  1-142-    yes-add constant for next check
2022    302  snz        -   is result zero?
2023   1642  jmp  1-42  -   yes- go to 2042
2024    201  smi        -   no- is result negative?
2025   1644  jmp  1-44  -   no- go to 2044
2026   2340  add  1-140-    yes- add constant for next check
2027    202  sze        -   is result zero?
```

```
2030    5634   jmp*  1-34  -     no-execute OUTPUT CONTROL
2031     762   lda   1-162-     yes-get DOWN routine address
2032    1010   sta   0-10  -     store in output routine address
2033    5414   jmp*  0-14  -     execute SYSTEM algorithm
2034    1241                -    OUTPUT CONTROL entry address
2036     761   lda   1-161-     get UP routine address
2037    1632   jmp   1-32  -     go to 2032
2040     757   lda   1-157-     get RIGHT routine address
2041    1632   jmp   1-32  -     go to 2032
2042     760   lda   1-160-     get LEFT routine address
2043    1632   jmp   1-32  -     go to 2032
2044     420   lda   0-20  -     get delta x
2045    2037   add   0-37  -     add old correction constant
2046    1037   sta   0-37  -     store new x correction constant
2047     421   lda   0-21  -     get delta y
2050    2040   add   0-40  -     add old y correction constant
2051    1040   sta   0-40  -     store new y correction constant
2052    1600   jmp   1-0   -     go to 2000

2064                       -     entry to RIGHT
2065     420   lda   0-20  -     get delta x
2066    7716   jst*  1-116-     execute DIVIDE
2067    1033   sta   0-33  -     store right scale factor
2070    1600   jmp   1-0   -     go to 2000

2072                       -     entry to LEFT routine
2073     420   lda   0-20  -     get delta x
```

```
2074    7716   jst* 1-116-    execute DIVIDE
2075    1034   sta  0-34 -    store left scale factor
2076    1600   jmp  1-0  -     go to 2000

2100                  -       entry to UP routine
2101     421   lda  0-21 -    get delta y
2102    7716   jst* 1-116-    execute DIVIDE
2103    1035   sta  0-35 -    store up scale factor
2104    1600   jmp  1-0  -    go to2000

2106                  -       entry to DOWN routine
2107     421   lda  0-21 -    get delta y
2110    7716   jst* 1-116-    execute DIVIDE
2111    1036   sta  0-36 -    store down scale factor
2112    1600   jmp  1-0  -    go to 2000
2113-2117 Constants and Temporary Storage
                        1. DIVIDE
2200                  -       entry ( Store return address
2201     301   spl      -      is accumulator positive?
2202       5   tca      -      no- minus it
2203    1251   sta  1-51 -    yes- store denominator(D)
2204      60   cra      -      zero into accumulator
2205    1254   sta  1-54 -    store quotient (Q)
2206     653   lda  1-53 -    get numerator (N)
2207    1250   sta  1-50 -    store N1
2210     653   lda  1-53 -    get constant
2211    1252   sta  1-52 -    store in pointer(P)
```

```
2212    650   lda   1-50 -    get N1
2213   2652   ana   1-52 -    select bit P
2214    202   sze        -    is result zero?
2215   1636   jmp   1-36 -    go to 2236 (no)
2216    651   lda   1-51 -    get D (yes)
2217   2652   ana   1-52 -    select bit P
2220    202   sze        -    is result zero?
2221   1634   jmp   1-34 -    no- go to 2234
2222    652   lda   1-52 -    get P
2223    101   lgr   1    -    right shift one(Select next m.s.b.)
2224   1252   sta   1-52 -    store P
2225    202   sze        -    is result zero?
2226   1612   jmp   1-12 -    go to 2212 (no)
2227    651   lda   1-51 -    get D (yes)
2230      5   tca        -    minus it
2231   2250   add   1-50 -    add N1
2232   1250   sta   1-50 -    store in N1
2233   3254   irs   1-54 -    increment Q
2234    654   lda   1-54 -    get Q
2235   5600   jmp*  1-0  -    return
2236    651   lda   1-51 -    get D
2237   2652   ana   1-52 -    select bit P
2240    202   sze        -    is result zero?
2241   1622   jmp   1-22 -    go to 2222 (no)
2242    651   lda   1-51 -    get D (yes)
2243      5   tca        -    minus it
2244   2250   add   1-50 -    add N1
```

```
2245    1250   sta   1-50 -     store in N1
2246    3254   irs   1-54 -     incremtn Q
2247    1610   jmp   1-10 -     go to 2210
2250-2254  Constants and Temporary Storage
```

2. STALL

```
1260    1274   sta   1-74 -     store accumulator
1261     675   lda   1-75 -     get number of fields to wait
1262    1276   sta   1-76 -     store in counter (C)
1263     677   lda   1-77 -     get address for output routine
1264    1010   sta   0-10 -     store for output
1265    5414   jmp*  0-14 -     enter SYSTEM algorithm
1266                  -         entry
1267    3276   irs   1-76 -     increment C
1270    5666   jmp*  1-66 -     C≠0  return
1271     674   lda   1-74 -     if C is zero get accumulator
1272    2300   add   1-100-     add constant
1273    5701   jmp*  1-101-     return
1275-1301 Constants and Temporary Storage
```

COMMON ( ZERO SECTOR)

| | | | | |
|---|---|---|---|---|
| 0 | 560] | jmp* 1-1 | - | execute OUTPUT CONTROL |
| ] | 1241 | sta 1-41 | - | output control routine link |
| 2 | 3700 | jst 1-100 | - | execute INTERRUPT |
| 4 | 400 | | - | LONG CHORD LINK |
| 5 | 446 | | - | CXAV LINK |
| 6 | 500 | | - | PXAV LINK |
| 7 | 566 | | - | CONVERT LINK |
| 10 | | | - | output routine link location |
| 14 | 175 | | - | SYSTEM LINK |
| 15 | | | - | CXAV XXN VALUE |
| 16 | | | | |
| 17 | | | - | PXAV VALUE |
| 20 | | | - | DELTA X |
| 21 | | | - | DELTA Y |
| 22 | | | - | PUPIL DIAMETER |
| 23 | | | - | CORNEA FILE START ADDRESS |
| 24 | | | - | PUPIL FILE START ADDRESS |
| 25 | | | - | CORNEA BOTTOM VALUE |
| 26 | | | - | PUPIL BOTTOM VALUE |
| 27 | | | - | CORNEA BOTTOM ADDRESS |
| 30 | | | - | PUPIL BOTTOM ADDRESS |
| 33 | | | - | RIGHT SCALE FACTOR |
| 34 | | | - | LEFT SCALE FACTOR |
| 35 | | | - | UP SCALE FACTOR |
| 36 | | | - | DOWN SCALE FACTOR |
| 37,40 | | | - | NULL CORRECTION CONSTANTS 95. |