A COMPUTER BASED ALARM HANDLING SYSTEM

FOR PROCESS PLANT

by

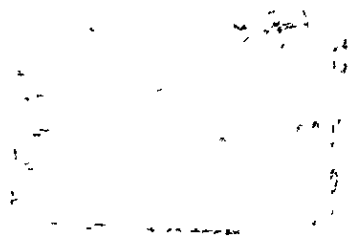
GARY HOENIG

VOL II

APPENDICES

APPENDIX A

INTRODUCTION TO THE
OLDMAN ON-LINE CRT DISPLAY PACKAGE



APPENDIX A

TABLE OF CONTENTS

A.1. INTRODUCTION

Recently the author in conjunction with the Department of Industry undertook a research and development program to study and perfect cathode ray tube (CRT) based display techniques used for displaying process information to plant operating staff. Surveys by the Department of Industry have shown that all the major manufacturers of process control equipment supply CRT's with standard display formats as the primary interface between the process and the process operator. Since the range of display formats provided by most instrument manufacturers is limited, operators are unable to investigate alternative ways of displaying plant information. The project endevoured to develop a display facility whereby operating staff could specify and assess a variety of display formats while the system was on-line.

Although the intention of the exercise was to provide information of the types of display formats preferred by operators, the implementation of the project yielded a variety of technical difficulties primarily related to real time microprocessor applications.

A.2. HARDWARE

Due to limited resources the most suitable equipment available for implementing the system consisted of:

1) Intecolor 8001 VDU with BASIC programing support for the 8080 microprocessor with a maximum capacity or 24K Bytes for display and software implementation. A dual drive floppy disc storage system used for program and data storage was interfaced to the Intecolor.

2) Solartron Compact Logger for sampling,

conditioning, and logging plant data from up to 30 analogue inputs.

3) A purpose built dedicated touch control operator's keyboard for display selection.

A.3. SYSTEM DESCRIPTION

The on-line display management package, referred to as OLDMAN, is intended to be a self-contained system which can be attached to an existing process plant. Process variables are measured directly from the existing plant instrument sensors and control system using the data logger. Data is sent to the Intecolor display via an RS-232 communication link. The Intecolor converts the data into a displayable form. See Figure A.1. Individual picture formats are stored on the floppy disc. The operator selects the appropriate display using the custom built touch control keyboard. The system then locates the correct picture overlay stored on disc. As incoming data from the data logger is received, the system stores a limited amount of data for historical type display formats such as trend type displays. All relevent data is then displayed on the VDU in a format described in the picture format which is stored on disc.

Figure A.2 shows the OLDMAN display system in use by an operator. The touch keyboard located in front of the display is adequately sensitive to be used even with heavy gloves. The keyboard construction is also resistant to environmental abuse.

The display system provides a means of specifying both the variant and invariant information content of a display. The design procedure is to draw the invariant parts of the

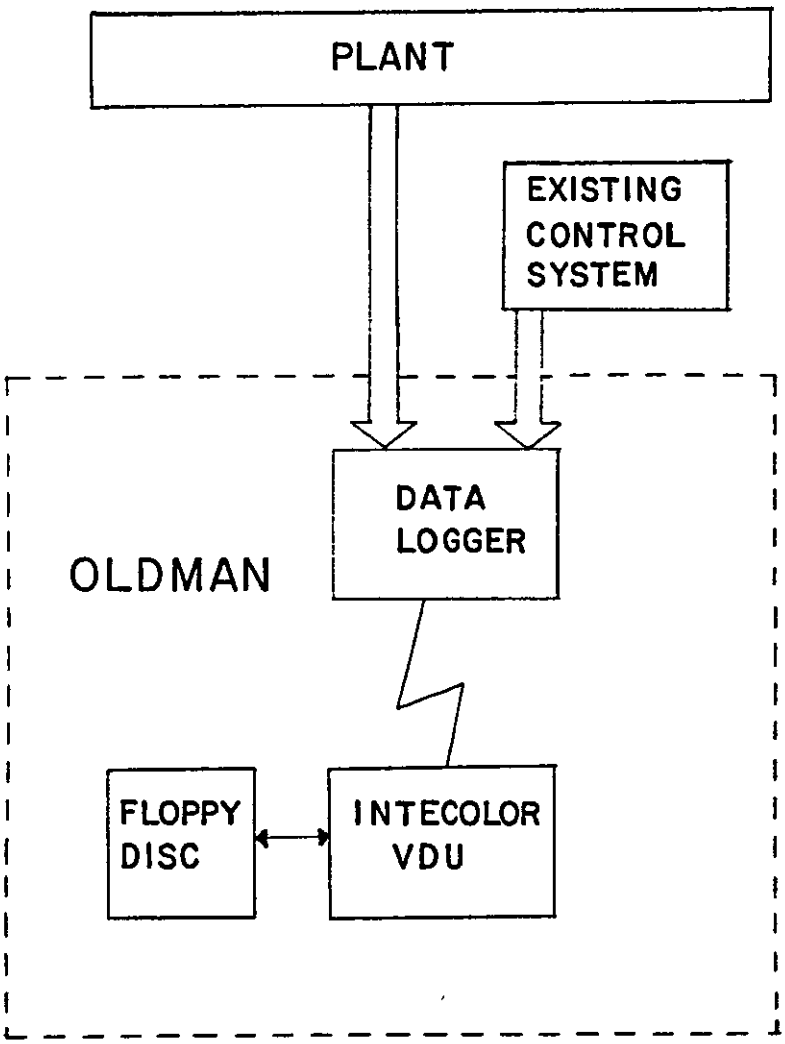


FIG. A.I. OLDMAN Information Display Package

Figure A-2 The OLDMAN Display Package in Use

display overlay such as titles, legends, or mimics using special graphic and semigraphic characters. Then how and where the variable information is to be displayed is specified. The two step procedure generates a 'picture blank', the invariant picture content, and the 'activated picture'. The 'activated picture' is the 'picture blank' with variant display locations inserted. All of this data is stored on the floppy disc medium. The display can be comprised of an invariant picture and variant process information in the form of alphanumerics, analogue representation, or graphical plots. Figures A.3 through A.7 illustrate some of the display formats which can be built.

The on-line display management package consists of two major sections:

1) On-Line Mode where displays containing real time information are accessable to the operator on demand.

2) Off-Line Mode where, using the same hardware, operators and/or designers can build display formats as required.

Formats are developed, editted, and stored in the off-line mode.

In the on-line mode the system operates in real time, collecting process data and displaying the information acccording to one of the operated selected formats developed off-line. The two operational modes can not both be running simultaneously. Further details of the OLDMAN display package are described in reference [A1]. Some operational difficulties of the system are discussed by the author in reference [A2].

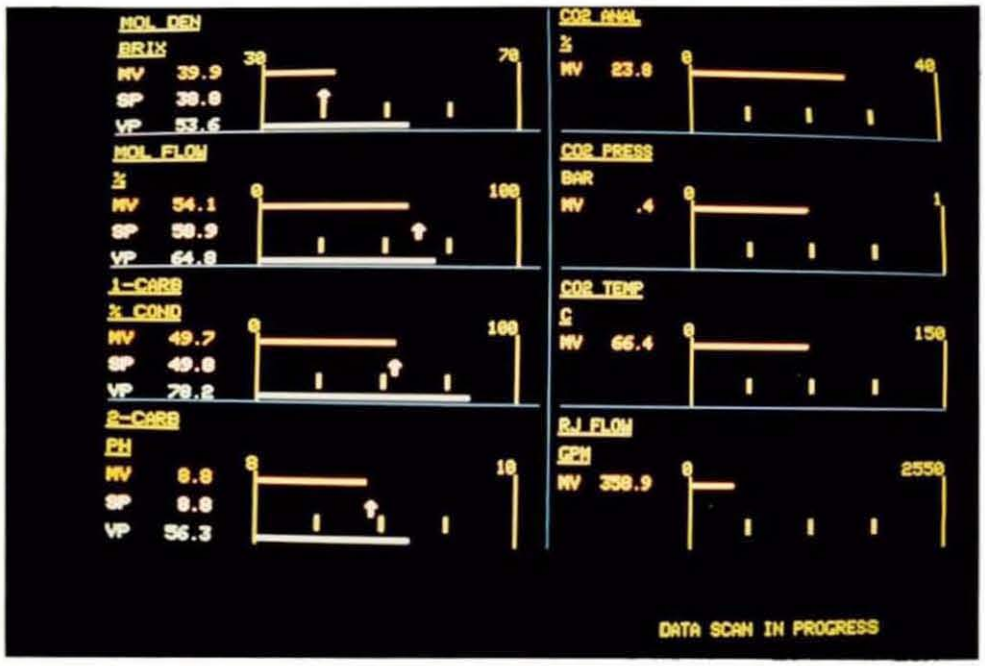


Figure A.3 Horizontal Loop Display Format

Figure A.4 Vertical Loop Display Format

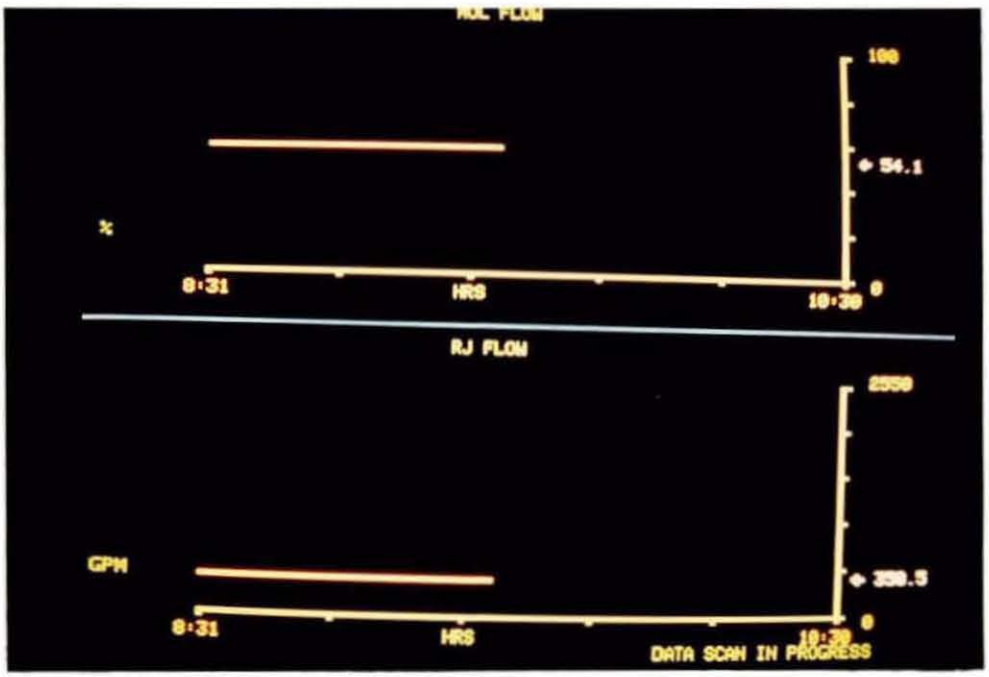


Figure A.5 Historical Trend Display x 2

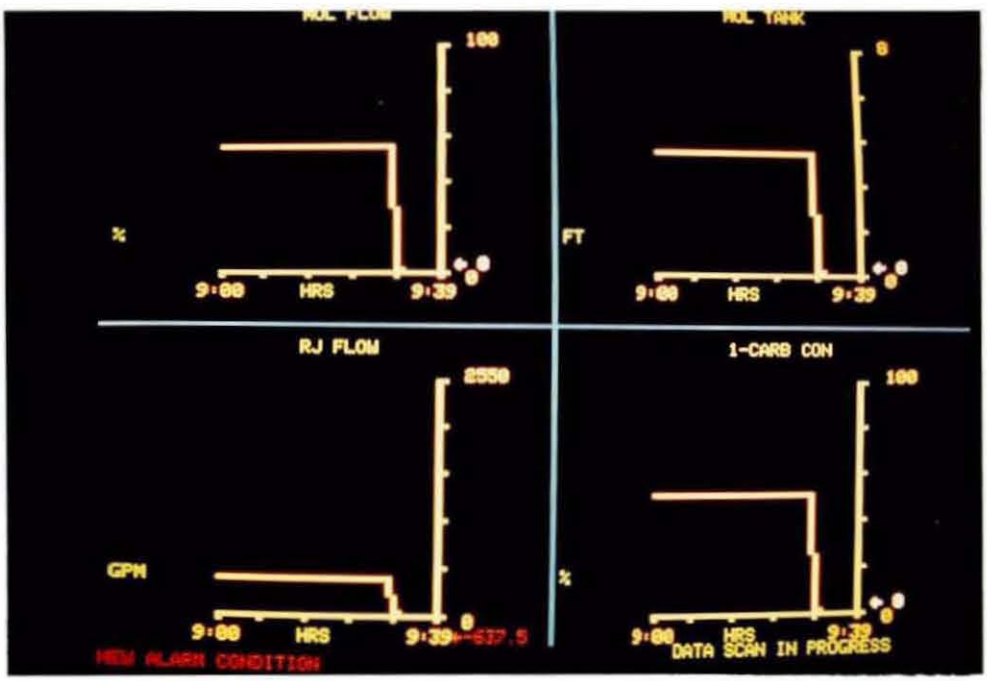


Figure A.6 Historical Trend Display x 4

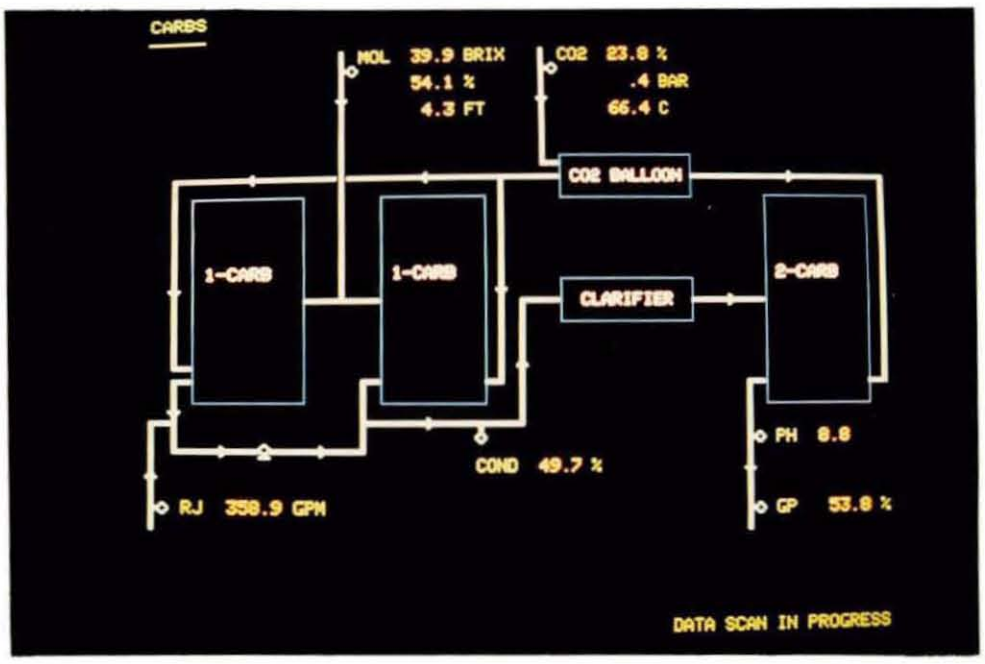


Figure A.7 Example Mimic Display Format

A.4 REFERENCES

A1. Umbers, I.G., Mark, S.J., and Hoenig, G., 1981, "A User's Guide for an On-Line Display Management Package", PRC2(CON), Department of Industry, Warren Spring Laboratory, Stevenage, England.

A2. Hoenig, G., 1981, "Real-Time Design Considerations for an On-Line Display Package", internal report, Department of Industry, Warren Spring Laboratory, Stevenage, England, June, 1981.

APPENDIX B

USER'S GUIDE FOR THE OFF-LINE COMPONENT OF THE ALARM HANDLING SYSTEM

APPENDIX B

TABLE OF CONTENTS

B.1. SYSTEM SPECIFICATION

B.1.1 Introduction

Computer based process control systems are becoming common place in the process industry. With the switch to automated processing many features of process computer control have not been adequately examined. The alarm subsystem is such a feature. A recent survey and report[1] has shown that all major manufacturers of process control equipment supply inadequate alarm handling facilities. The system described in this report has been designed as a generalized alarm handling package with a high degree of system flexibility. The handling package is intended to be an add on feature to existing process control systems. An exercise of this nature will provide information on the types of alarm requirements preferred by operators and designers and will also provide an insight into how adequate alarm handling facilities should be incorporated in existing process control systems.

B.1.2 Equipment Description

It is usually the best policy when designing a computer based system to concentrate on the functional specification before considering the implementation of the system. In this way the design approach is not limited by the physical entities which are available, but instead, is directed towards satisfying the overall system objectives. However, where resources are limited some account must be taken of the hardware that can be provided by the available

resources.

Prototype research and development often requires changes to be made in the overall system objectives. The possible changes in system requirements necessitates the choice of hardware which not only meets preliminary requirements, but which is also capable of serving future needs. The most suitable equipment available for implementing the off-line component of the alarm handling system considered here is outlined below.

The off-line computer used for prototype development of the alarm handling system is the Chromatics CG 1999 Colour Graphics Computer. This Z-80 processor based unit contains 32K of memory, a high resolution colour graphics display, and a single 8 inch, 250K single sided, single density floppy disk drive. A Teletype T43 printing terminal is available for connection to the Chromatics via an RS-232 serial port. The printer is used for obtaining listings of the loaded data. The serial line is also used as the down line link to the on-line computer for transfering the alarm data base to the on-line system. The Chromatics contains Micro Soft BASIC Ver 3.0 language which operates in conjunction with a Chromatics operating system and disk operating system.

B.1.3 System Requirements

The off-line alarm handling system is an independent, stand-alone computer system which is used to build an alarm handling data base for subsequent use in the on-line alarm handling system. The off-line system allows the designer to load alarm and other alarm related information into the system at his leisure. This raw input data is stored in files for future use. File storage of the raw data also

allows the designer to review and make changes to the raw data with relative ease. Once the raw data is stored, the off-line system retrieves the raw data from the storage files, examines the data, and converts the data into a condensed numerically coded data base. This alarm data base contains all of the information in the raw data files plus other information required by the on-line computer system. Converting the raw data into a data base structure ensures that the on-line computer program can run at maximum efficiency and speed. The alarm data base is stored by the off-line system ready for future transfer to the on-line system.

B.1.4 Organisation of the Report

The remainder of the report presents a user's guide to the facilities provided by the off-line alarm handling system package. A second report describes and documents the OFLAD software.

B.2. USER'S GUIDE

B.2.1 Introduction to OFLAD

The OFf-Line Alarm Data (OFLAD) program structure is restricted by the available memory in the off-line computer, i.e., the Chromatics CG 1999. The resulting structure is comprised of three separate BASIC programs which interact without operator intervention. This approach maximizes the memory space available for data handling. The three program modules are:

1) System Monitor and Data Input Routine (OFLAD)

2) The Data Compiler (COMP)

3) The Data Base Loader (TRANSFER)

Each of the three modules are linked via common data files stored on the disk unit. The System Monitor provides the necessary coding for loading and running the other modules as appropriate.

B.2.2 Conventions

Certain conventions are observed in operating the system:

a) The type of response required to questions asked by the software are as far as possible self-explanatory. The response to a question requiring a yes or no reply is either "Y" or "N". Occasionally special response formats are required by the software. In these instances special instructions are given in the user's

guide and the reference is given in the software prompt.

b) Where more than one item is required by a question the items supplied must be separated by a space.

c) User responses are entered by a carriage return.

d) There are four types of files used by the system:

i) DA Files: Data Acquisition Files contain coded ASCII information required when building an alarm data base.

ii) EP Files: Event Processor Files contain coded ASCII information required when building an alarm data base.

iii) AG Files: Alarm Generation Files contain coded ASCII information required when building an alarm data base.

iv) ADB Files: Alarm Data Base Files contain an image of an alarm data base as generated by the COMP module. The ADB is transferred to the on-line computer by the TRANSFER module.

e) The permitted ranges of file identifiers are as follows:

File Type	Range
DA	1 - 25
EP	26 - 50
AG	51 - 75
ADB	76 - 100

B.2.3 Start-Up Procedure

1. Ensure that the T43 printer is connected to port SIO-0 on the Chromatics.

2. Switch on the Chromatics, disk drive unit, and T43 printer.

3. Insert a suitably initialised floppy disk into the disk drive unit (check that the label is uppermost and nearest to the drive door.)

4. On the Chromatics keyboard: (see Note: 1)

   i) Press the (RESET) and the (BOOT) keys.
   ii) Press the (BASIC) key.
   iii) The Chromatics will respond with a request for memory size:

```
MEMORY SIZE?
```

5. Type: &HB000 (RETURN)

6. Type: DOS"LOAD OFLAD" (RETURN)

7. Type: RUN (RETURN)

OFLAD will now be loaded from the floppy disk and will commence to run.

Note 1: The ( ) around a letter or a series of letters indicates that this is one key on the keyboard.

B.2.4 OFf-Line Alarm Data System Monitor

The OFLAD Alarm Data System Monitor comprises an operating system and alarm data input routines. The operating system recognizes commands used to evoke the various functions available in the software package. The data input routines are essentially data file editors which allow the user to perform editing functions on any of three data base file areas called working files. These working files correspond to the three major operational sections of the Alarm Handling System; Data Acquisition, Event Processing and Alarm Generation. Data files may be built, examined, and stored. Similarly previously developed data files can be examined, modified, deleted, etc. Once files are completed the Compiler is used to build a coded Alarm Data Base which can be interpreted by the on-line Alarm Handling System. Compilation checks the validity of the Data Acquistion (DA), Event Processor (EP) and Alarm Generation (AG) files, building them into a numerically coded Alarm Data Base (ADB). Finally, the data base can be transferred to the on-line Alarm Handling System.

B.2.4.1 Functions Available

OFLAD provides facilities for activating Data Acquisition, Event Processor and Alarm Generation files. (Activate refers to the procedures used to arrange for dynamic information to be used for alarm data base compilation.) In addition, facilities are provided for housekeeping tasks such as deleting DA, EP, and AG files and listing disk directory.

When OFLAD has been loaded and is waiting for a command it will prompt the user by printing:

COMMAND:

The user responds by typing a command shown in the list below. If the user does not know the command to use then a list of commands and functions can be displayed by entering an incorrect command or the command HElp. Only the first two letters of the command need be entered.

Command	Function
DIrectory	List disk directory
LOad	Load data files into working file space
STore	Store or kill working files
COmp	Goto COMP module (Module 2)
FIles	List working files loaded
ENter	Enter data in working file
CHange	Edit a working file
LIst	List a working file
DElete	Delete a disk file
TRansfer	Goto to TRANSFER module (Module 3)
HElp	List commands

B.2.4.2 Housekeeping Functions

List Disk Directory - DI. The contents of the disk currently loaded in the disk drive unit are listed with the following example format:

FILE NO. - NO. OF ELEMENTS

DA

```
1 - 10     5 - 20     ,etc.

EP

30 - 10    ,etc.

AG

60 - 10    ,etc.

ADB

100 - 250 ,etc.
```

The first number displayed is the file reference number followed by the number of elements in the file. The number of elements is directly proportional to the file size.

Delete Files - DE. This command allows unwanted DA, EP, AG and ADB files to be removed from the currently loaded disk. The command format is as follows:

```
INPUT FILE NUMBER:
```

The required file identifier number is obtained from the directory listing. This value is entered. The program now asks

```
SURE?
```

which requires confirmation of the deletion ("Y" or "N").

Go to COMP Module - CO. This command is entered when all the currently required activation or file maintenance

operations have been performed. It loads and starts the second program (Module 2: COMP), which is described elsewhere. In brief the COMP Module is used to compile and build an alarm data base from activated DA, EP and AG files.

*** WARNING *** All working files must be secured by using the STore command, otherwise the information in the working files is lost.

Go to TRANSFER Module - TR. This command is entered when all the currently required activation or file maintenance operations have been performed. It loads and starts the third program (Module 3: TRANSFER), which is described elsewhere. In brief the TRANSFER module is used to transfer a compiled alarm data base to the on-line alarm handling computer.

*** WARNING *** All working files must be secured by using the STore command, otherwise the information in the working files is lost.

HElp - HE. This command lists the available system commands. No response with a (RETURN) is equivalent to the HElp command.

B.2.4.3 File Manipulation and Data Entry

The OFLAD program module contains three working file areas allocated to the DA, EP and AG sections respectively. There are two modes of operating the working files.

1) The working files can be 'loaded' with files of the same type from the floppy disk drive unit. Once loaded

using the LOad command, the data in the respective working file may be altered using the CHange command. New data is added to the file by using the ENter command. Data is deleted from the working file through the use of the CHange command and inputing a space in place of the first data string in the data group (see CHange command). A working file can be listed at any time using the LIst command. The file is then stored on disk using the STore command.

2) Data can be immediately entered into a working file which has not been loaded from the disk. A new working file space is loaded by using the LOad command and responding with a file identifier number which does not exist on the currently loaded disk. The CHange, ENter, LIst, and STore commands function in the same manner as described above. When the loaded files are STored, the currently loaded file identifier number is used to store the file on disk unless instructed otherwise.

Either method may be used. The FIle command will list the working files currently present in the system. Note: The working files are not stored until a STore command is issued. If other program modules are entered, the current working files are lost.

Load Working Files - LO. This command allows the user to copy the contents of files stored on the currently loaded floppy disk into the appropriate working file space. The command also allows the creation of a new file. There are three working files; DA, EP and AG respectively.

The program responds:

DA FILE NUMBER:

The number (1 - 25) of the DA file to be loaded is entered. Next the program asks:

EP FILE NUMBER:

The number (26 - 50) of the EP file to be loaded is entered. Next the program asks:

AG FILE NUMBER:

The number (51 - 75) of the AG file to be loaded is entered.

If a file is not present on the currently loaded disk, the program responds

NEW FILE

If no file is to be loaded then press RETURN .

List the Loaded Working Files - FI. This command lists the file identifier numbers of the files which have been loaded into the respective working files. If a working file has not been loaded, no response will be given for that particular working file.

Store or Kill Working Files - ST. This command allows the user to delete a working file or store the file on the currently loaded disk with either the same file identifier number or a new one. The STore command is the same for any of the three working files. The user is then asked whether or not each individual working file is to be stored, etc. If a file is loaded in the working file the program asks:

STORE DA (file number)?

STORE EP (file number)?

STORE AG (file number)?

In each case the response to the prompt is as follows:

RETURN = Do not store, leave status quo

"Y" = Store, no change in file number and delete working file

"NF" = Store with new file number and delete working file

"K" = Kill working file

If the response is "NF" the program then asks:

ENTER NEW NUMBER?

The new file identifier number can be any legal file number for the respective working file.

Enter New Data - EN. This command allows new data to be added to a loaded working file. The working file is specified by the prompt:

ITEM:

The acceptable responses to the ITEM: prompt are:

Da - Data Acquisition working file

Ep - Event Processor working file

Ag - Alarm Generation working file

Help - Help, list ITEM commands

Only the first letter D, E, A, or H respectively need be entered. No response to the ITEM: prompt, i.e., (RETURN) only, exits the ENter command.

DA Unit - D. If the response to the ITEM: prompt is "D" then the program responds:

INPUT DATA ACQUISITION INFORMATION

This indicates that the program is entering the routine for inputing information relating to data acquisition. This is the first working file. Each measured variable used by the alarm handling system must be assigned in order that the system knows where and how to obtain process information about the variable. Each question must be answered to ensure proper compilation of the alarm data base. However, a (RETURN) with no response leaves the item unchanged. This is useful when using the CHange command.

The program responds:

PLANT CODE []:

This is the identifying code for the measured variable. Whenever the particular measured variable is referenced the Plant Code must be used. Any alphanumeric code is acceptable.

Next the program asks:

NAME []:

This is the English description of the Plant Code. It is recommended that the name be as brief as possible. The maximum length should not be more than 10 characters.

The program continues:

INPUT DEVICE []:

The input device for the measured variable is now assigned. The device assignment is the device from which data is acquired on the on-line computer system. The acceptable devices are:

0 - Device #0 Media Plant I/O

1 - Device #1 TT3: Host Link

2 - Device #2 TT1: T43 Terminal

3 - Device #3 Not Assigned

4 - Device #4 Not Assigned

M - Memory location when using DMA equipment.

If "M" is entered the program asks:

ADDRESS []:

Any decimal address is acceptable.

The program next asks:

DATA TYPE []:

The data type is either Analogue or Binary, so enter either "A" or "B". The data type is analogue if the measured variable is represented by a continuous parameter. The data type is binary if the measured variable has only two states, i.e., ON-OFF or 0-1. If the response is "A" the program asks:

CONVERSION ALGORITHM #[]:

There are five conversion algorithms available on the on-line computer. The conversion number identifies how the measured variable is to be converted into values used by the event processor. Enter a value 1 to 5. (See Table B-1.)

If the measured variable is a binary type then the program asks if data inversion is required.

DATA INVERSION (Y/N):

A "Y" response indicates that the binary value is changed or inverted from either a 0 to a 1 or a 1 to a 0 before it is stored in the DA on-line data base.

The program continues:

RANGE []:

Enter the minimum and maximum values that the measured variable will reach. These values must be in units identical to those seen by the on-line alarm handling sytem. The format of the response is:

value min value max RETURN

These values are not engineering units, they are in the measured parameter units. The program can not deal with engineering values.

The program next asks for the significant change:

SIGNIFICANT CHANGE []:

Enter in measured parameter units the minimum change during one scan period which will initiate event processing. Data value changes that occur during the data acquisition scan period are compared with the significant change. No action is taken if the data value change is less.

The next question asked is:

SCAN RATE []:

The Scan Rate is the time between data acquisition samplings. The Alarm Handling System has four levels of scanning.

Scan Rate	Time Interval
1	1 Second
2	5 Seconds
3	15 Seconds
4	1 Minute

Enter the number 1, 2, 3, or 4 and (RETURN). The program now asks:

SCAN PRIORITY []:

Within each scan group the measured variables are sampled one after another in the order of their priority. Any integer number 1 - 100 can be entered with 1 being the highest (first up) priority. If more than one variable has the same priority in a scan group then the variables with the same priority are sampled in alphanumeric order of the Plant Code. With a large number of variables in a scan group, in some cases, the time between sampling may not always be consistent for low priority variables.

The program responds

END OF DA INPUT

and returns to the COMMAND prompt, ready for further instructions.

Event Processor List - E. With the response to the ITEM: prompt "E" the program prints:

INPUT EVENT STATUS INFORMATION

This indicates that the program is entering the routine for inputing information relating to event status assignments. This is the second working file. The event processor in the on-line computer retrieves processed measured variable information from the DA on-line data base. These values are compared with the conditions assigned in the event list corresponding to each Plant Code. The event processor sets a flag in the event status image if the conditions for the event are true. Each question must be answered to ensure proper compilation of the alarm handling system data base. However, if no response is required or

the question does not apply, then press (RETURN) . This results in no change to the data in the working file and is useful when using the CHange command.

The program asks:

PLANT CODE []:

The alphanumeric response to this question must be identical to the plant code given in the DA unit section in order to assign the correct event status conditions. If any Plant Codes are entered which are not in the DA file, the alarm data base will fail to compile. Any number of events can be assigned to a Plant Code, however each must have a unique event name.

Next the program asks:

EVENT NAME []:

This is the English description of the event. It is highly recommended that the name be as brief as possible. The maximum recommended length should be not more than 10 characters. Spaces should also be avoided since the event name is entered in the alarm file which has difficulties dealing with spaces. Spaces are permitted, but special formats when using spaces are required in the alarm file.

Now the program continues:

EVENT TYPE []:

The event type describes the class of event as follows:

XLO

LO

HI

XHI

TREND (Change in DA)/(Scan Interval)

ON

OFF

DEVI

TDEVI

The event type determines how the alarm handling system examines the DA data before deciding if an event has occurred. For example, if an event is named TANKHI then a possible event type would be HI, i.e., the variable is then examined in this context. If the type is TREND then the variable is examined on the basis of rate of change and so on. The limits are entered which represent the limits for the selected event type. These event condition parameters are entered next. Analogue event types require limits of the alarm band. For analogue event types the response must be a numeric value in units related to those generated by the DA processor. In other words if the values are modified by the DA processor conversion algorithms, the limit values are applied to the converted values. Analogue bands can also implement hysteresis on the band limits.

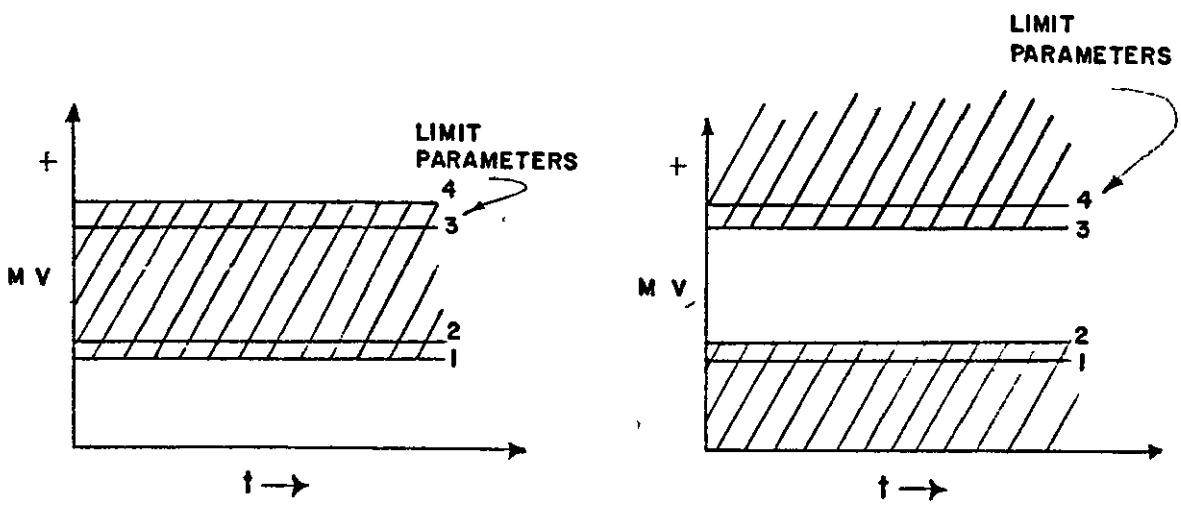
ON/OFF binary events do not require event conditioning. If the event corresponds to a binary variable press (RETURN)

The program continues:

ENTER PARAMETERS

1 []:
2 []:
3 []:
4 []:

Referring to the figure below the shaded areas represent the plant data value range after conversion in which an event will be considered to have occurred, i.e. TRUE. The limits of these ranges are represented by the parameters 1, 2, 3 and 4. Parameters 1 and 2 describe the lower limit of the range while 3 and 4, the upper limit. There are two parameters for each range limit to accommodate hysteresis.



1a Applies to most event types

1b Applies to DEVI and TDEVI only

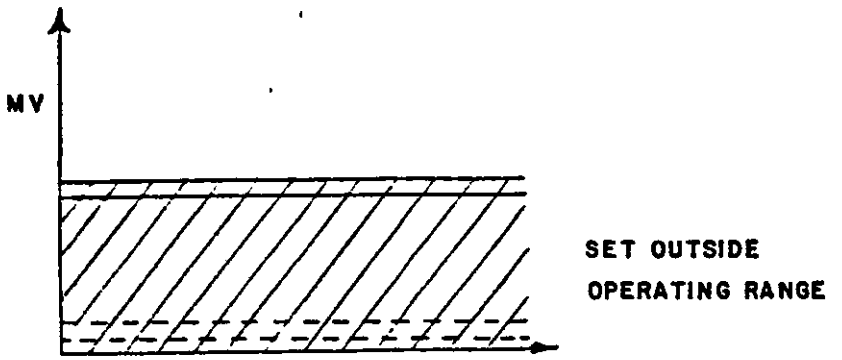
Figure B.1. Analogue Range Parameters

Note that Figure B.1a illustrates the event range for XLO, LO, HI, XHI, and TREND events. Figure B.1b shows the event range for DEVI (deviation) and TDEVI (trend deviation) events. XLO, LO, HI, and XHI event labels are used for the user's convienence. There is no difference in the way that the event range limits are used to detect events. The parameter values are entered in the same fashion for all event types. All parameters must be entered and in the value order 1<=2<3<=4. To implement event types so that only one limit is used refer to the Figure B.2.

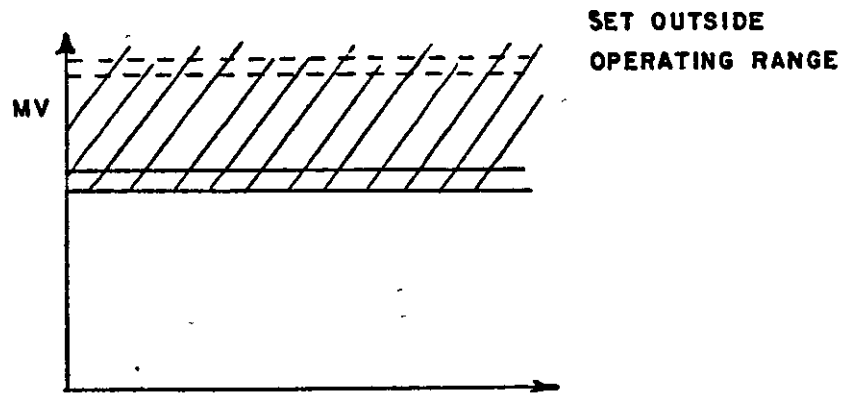
The deviation events inverse the event range so that all values outside the band are considered to represent TRUE events.

Hysteresis causes a lag in response when variables are close to limit parameters. This is implemented by shifting the event range limit from one parameter value to the other depending upon the direction of approach of the measured variable. The limit parameters 1,2 and 3,4 are used to set the upper and lower hysteresis values for each band limit.

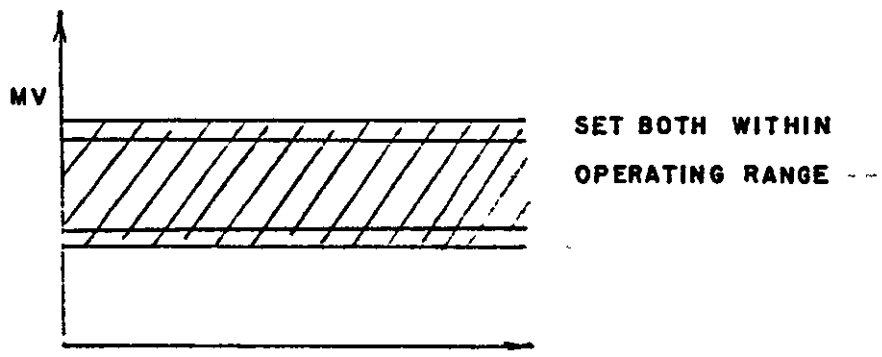
Referring to Figure B.3, the lower limit 1 is the lowest value of the value band in which the event status condition is true. If the measured plant value makes a positive going excursion across the lower limit, the event is not true until the value crosses the lower limit 2. The reverse is true when the measured value makes a negative going excursion across the band limits. The same hysteresis shift occurs on the high limit. If no hysteresis is required then set the low limit 1 equal to the low limit 2 and the high limit 3 equal to the high limit 4.



LO EVENT WITH SINGLE LIMIT



HI EVENT WITH SINGLE LIMIT



LO OR HI EVENT WITH DUAL LIMITS

Eg.; Measured Plant Value Change = MV

Low Limit 1 = 10
Low Limit 2 = 12

Hi Limit 3 = 20
Hi Limit 4 = 22

MV = 9 to 11; Event = False
MV = 9 to 12; Event = True
MV = 23 to 21; Event = False
MV = 21 to 19; Event = True

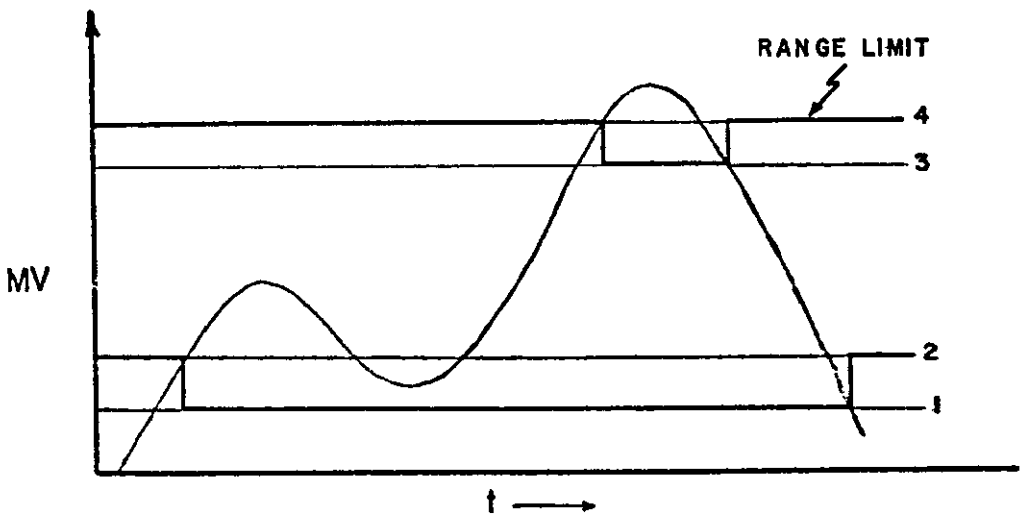


Fig B.3. Diagram of Limit Hysteresis Response

The program next prints

END OF EVENT PROCESSOR INPUT

and returns to the COMMAND prompt, ready for further instructions.

Alarm Generation - A. With this response to the ITEM: prompt the program responds:

INPUT ALARM CONDITION INFORMATION

This indicates that the program is entering the routine for inputing information relating to alarm conditions and alarm output assignments. This is the third working file. The alarm processor in the on-line computer inspects the event status image generated by the event processor. The alarm processor compares the event image with the conditions for alarm ON and alarm OFF. If these conditions are met, the alarm processor informs the alarm display unit of the change in the alarm status image. Each question must be answered to ensure proper compilation of the alarm handling system data base. However, if no response is required or the question does not apply then press (RETURN) . This results in no change to the data in the working file and is useful when using the CHange command.

The program asks:

ALARM NAME []:

This is the English description of the alarm. Any alphanumeric response may be entered. It is recommended that the alarm name be as brief as possible. The maximum length should be not more than 20 characters.

Next the program asks:

OUTPUT CODE []:

This code is the alarm reference code which is sent to the display unit when a change in alarm condition occurs. This code must be recognizable by the display unit otherwise the on-line alarm handling system will not function correctly. Any alphanumeric response may be entered.

The program continues:

CONDITION ON []:

A Boolean expression containing the event names which must be true is entered. The allowable Boolean operators are:

NOT

OR

AND

XOR

(Pseudo operator

) Pseudo operator

SEQ TIL

VOT

The format of the expression must be as follows:

1) Space between operators and operands.

2) No spaces can exist in the event name. If there are spaces in the event name substitute the space with ^.

3) Parentheses must be entered with a space as

_(except if preceeded by a (

)_ except if succeeded by a)

Examples:

LEGAL INPUTS

1) FLOW^TANK AND LEVEL1

2) NOT (LEVEL2 OR NOT (LEVEL1))

ILLEGAL INPUTS

1) LEVEL1AND FLOW TANK

2) NOT(LEVEL2ORNOT(LEVEL1))

Most errors are detected during compilation, however, great care should be taken to ensure correct functioning of the on-line alarm handling system.

Simple sequences can also be used as condition statements. The operator SEQ indicates that the time of occurance of the succeeding events must be in chronological order. The operator TIL is used to set a time limit on the detection. This time limit in seconds represents the time from the occurance of the first event in the sequence. All events must have occurred with this time limit for the sequence to be valid.

Example:

SEQ LEVEL1 LEVEL2 LEVEL3 TIL 30

The condition statement requires that event LEVEL1 must occur before LEVEL2. Also LEVEL2 must occur before LEVEL3. The time difference between LEVEL3 and LEVEL1 must be less than 30 seconds.

Note: No logical operators are allowed in a SEQ TIL statement.

A similar operation is the VOT operator. This operator examines the succeeding events and takes a majority vote. If the majority of events in the statement have occurred than the condition statement is satisfied.

Example:

VOT LEVEL1 HIFLOW TEMP

The program now asks:

CONDITION OFF []:

Again, a Boolean expression containing the event names which must be true is entered. In the on-line system the alarm output code is sent to the display unit with a data packet indicating that either the CONDITION ON expression is true or that the CONDITION OFF expression is true.

The program continues:

PERSISTENCY []:

The persistency value indicates the display characteristics of the alarm with respect to acceptance and reset. See the display unit document for further details. Acceptable values are 1 to 5.

The program responds

```
END OF ALARM CONDITION INPUT
```

and returns to the COMMAND prompt, ready for further instructions.

B.2.4.4 Editing Files

Change or Edit a Working File - CH. This command allows elements in the working files to be changed or deleted. Any of the three working files may be edited.

The program responds:

```
ITEM:
```

The working file is now entered in the same manner as when the ENter command is used.

Type:

D for DA

E for EP

A for AG

The program now asks which group of file entries are to

be changed or deleted.

For the DA unit working file the program responds:

ENTER PLANT CODE TO BE CHANGED:

For the EP working file the program responds:

ENTER EVENT NAME TO BE CHANGED:

For the AG working file the program responds:

ENTER ALARM NAME TO BE CHANGED:

In each case the exact Plant Code, Event Name, or Alarm Name respectively must be entered. If the entered request does not match any Plant Code, Event Name, or Alarm Name in the respective working file, the program responds with

CAN NOT FIND

and returns to the command prompt. If the response is found then the edit mode is entered allowing changes to be made to any of the questions in the group. The current contents of the file elements are shown in the brackets such as

PLANT CODE [F201]:

Enter the new data or press (RETURN) for no change. To delete a group of data in a working file, enter a space and (RETURN) in response to

PLANT CODE [____]:

EVENT NAME [____]:

ALARM NAME [____]:

Although the data group will still exist in the working file, when the working file is STored, the data group will be deleted.

List a Working File - LI. This command lists the contents of a loaded working file with a description header. The program responds:

WHICH ONE?

Enter D, E, or A for the DA, EP, or AG file respectively. The program continues:

HARD COPY?

If a printed copy of the working file contents is required, enter "Y". The output will then be transferred to the T43 Printer. Otherwise with an "N" response the listing is sent to the Chromatics VDU display.

Note: The T43 Printer must be connected to port SIO-0 with parity off, full duplex, and 300 BAUD.

When complete the program returns to the COMMAND prompt, ready for further instructions.

B.2.5 Compile Alarm Data Base

B.2.5.1 Introduction to COMP

The second program module in the off-line software is the COMP or compilation module. This program retrieves DA,

EP and AG files stored on the currently loaded floppy disk, processes the data, and finally builds and stores an activated alarm data base (ADB) file on the loaded disk. The compiler performs the following functions:

1) Inspects the DA, EP and AG files for obvious syntax errors.

2) Sorts the DA, EP and AG file information into a systematic order required for the alarm data base.

3) Cross correlates the DA, EP and AG files, inspecting for missing, duplicate, or mismatched data.

4) Provides a listing of the data immediately prior to data conversion.

5) Builds and stores an alarm data base ready for transfer to the on-line system.

Any errors in the data text are noted by an "*" before the incorrect element. Compilation errors are also noted with an error message for each error occurrance. Compilation will not succeed if there are any compilation errors. If errors do occur, the user must edit the appropriate file using the OFLAD module which is automatically loaded and run upon completion of the COMP module.

B.2.5.2 Use of COMP

Compile an Alarm Data Base - CO. This command allows the user to enter the COMP program module and compile and build an alarm data base. The alarm data base is stored on the disk unit.

The program responds:

DA FILE NO.?

Enter the DA file to be used in the compilation.

Next the program asks:

EP FILE NO.?

Enter the EP file to be used in the compilation.

The program continues:

AG FILE NO.?

Enter the AG file to be used in the compilation. The files entered to the above questions must exist on the currently loaded disk directory. The data in the files must be consistent with one another to ensure successful compilation. The files on the disk are not destroyed during compilation. The file contents are copied in the program module.

The program now asks:

ADB NO.?

Enter the file identifier number of the Alarm Data Base to be generated by the compilation. If the ADB number already exists on the currently loaded disk, the current ADB file will be overwritten by the ADB generated during compilation. The program will respond:

*** WARNING - ADB FILE __ ALREADY EXISTS ***

ARE YOU SURE?

Any response other than "Y" results in another 'ADB NO.?' request.

The program continues:

HARD COPY?

If a printed copy of the compilation listings are required, enter "Y". The output will then be transferred to the T43 Printer. Otherwise with an "N" response the listings are sent to the Chromatics VDU display.

Note: The T43 Printer must be connected to port SIO-0 with parity off, full duplex, and 300 BAUD.

The program will now load the requested DA file. The file is sorted and an alphanumeric Plant Code listing is produced. Any errors will be noted with a "*". A response

DA TYPE ERROR

indicates that a non-existent data type is present. The total DA errors are noted in the trailing error statement.

*** 0 ERRORS ***

A second listing will be produced of the DA file which is in Scan Group and Scan Priority order.

The program now loads the requested EP file. The file is sorted and an alphanumeric Event Name listing is produced. Any errors will be noted with a "*". A response

EP TYPE ERROR

indicates that a non-existent event type is present. The total EP errors are noted in the trailing error statement.

*** 0 ERRORS ***

The program now loads the AG file. The file is sorted and an alphanumeric Alarm Name listing is produced. Any errors will be noted with a "*". Carefully examine the Boolean expressions for errors. These expressions have been converted into Reverse Polish notation. The alarm listing format is:

ALARM NAME: alarm name (output code)

CONDITION ON: reverse polish expression

CONDITION OFF: reverse polish expression

PERSISTENCY: number

The total AG errors are noted in the trailing error statement.

*** 0 ERRORS ***

If no errors have occurred during compilation, the program responds:

*** TOTAL ERRORS = 0 ***

PASS 1 OK

*** COMPILATION OK ***

The program now builds the alarm data base, stores it on disk, and returns to the OFLAD module. The COMMAND: prompt indicates that compilation has completed and ready for further instructions.

If errors occurred during compilation the program responds:

*** TOTAL ERRORS = __ ***

*** COMPILATION FAILED ***

The program aborts and returns to the OFLAD module. The COMMAND: prompt indicates that the system is ready for further instructions.

The compilation time is proportional to the size of the files used. Compilation time grows rapidly with the total number of file elements.

B.2.6 Transfering the Alarm Data Base (TRANSFER)

B.2.6.1 Introduction to TRANSFER

The third program module in the off-line software is the TRANSFER module. This program retrieves ADB files stored on the currently loaded floppy disk and installs the compiled alarm data base in the on-line alarm handling computer. The transfer routine performs the following functions:

1) Enquires the user for the alarm data base (ADB) file to be transferred.

2) Establishes communication with the on-line

computer.

3) Transfers the data base to the on-line computer performing some error checking for link and transfer errors.

Any errors generated during the transfer task are most likely due to difficulties with the communication link to the on-line computer. If errors do occur the routine will abort, returning to the OFLAD module. The user should check all link lines and repeat the transfer command.

B.2.6.2 Use of TRANSFER

Transfer an Alarm Data Base - TR. This command allows the user to enter the TRANSFER program module and transfer an alarm data base file to the on-line alarm handling computer. The program starts:

```
ALARM DATA BASE TRANSFER ROUTINE

THE CHROMATICS MUST BE CONNECTED TO THE PDP11/03.

THE ALARM HANDLING SYSTEM MUST BE INSTALLED AND RUNNING
BEFORE PROCEEDING.
```

A link error will occur if the setup is not correct, thus aborting the transfer.

The program responds with a directory list of the alarm data base files available on the currently loaded floppy disk. The format is the same as for the List Directory command DI in OFLAD.

```
AVAILABLE ALARM DATA BASES
```

ADB
100 - 250, etc.

ADB No.?

Enter the appropriate alarm data base number to be transferred. The program continues:

ARE YOU SURE?

A "N" reponse will abort the transfer routine and the system returns to the OFLAD program module. The prompt:

COMMAND:

indicates that the system is ready for further instructions.

A "Y" reponse to the 'ARE YOU SURE?' prompt will initiate the transfer of the alarm data base down line to the PDP11/03. When completed, the program will return to the OFLAD program module ready for further instructions.

There are three forms of error messages which may be encountered during the transfer of the data base.

1) Type Error in Element

2) Link Error

3) Xfer Error

Type errors occur when the transfer routine has found an invalid element in the alarm data base. The element number in error is noted in the error message. This indicates that the alarm data base may be corrupted. Transfer will continue however the alarm data base in the

on-line computer will not be fully functional. If this error is encountered, execute the transfer again. If still persistent, re-compile the alarm data base.

Link errors occur when the transfer routine can not establish communication with the on-line alarm handling system. The transfer routine is aborted and program control is returned to OFLAD. Ensure that the on-line system is functional and that all communication lines are secured in the correct place. Try the transfer again. No data will have been transferred to the on-line system.

Xfer errors occur when communication checks on the link line show that there is a possibility that a corrupted data transfer has occured. The alarm data base in the on-line system will have been corrupted as well. Restart the transfer procedure after ensuring that the on-line system has also been reset. Program control will have been returned to the OFLAD program module.

B.3.0 INITIALISING DISKS

New floppy disks must be processed by the Chromatics CG 1999 File Control System program 'FORMAT'. This program formats the disk. The otherwise blank disk should be at hand before continuing. Obtain a copy of the off-line Alarm Handling System master disk which contains the system programs

OFLAD.BAS
LOADOFLAD.SRC
COMP.BAS
LOADCOMP.SRC
DISKINI.BAS

The procedure essentially consists of transfering all of the Chromatics System Files and the off-line Alarm Handling System programs to the new disk. The procedure is identical to that outlined in the Chromatics Disk Operating Manual[4].

1) Press the (DISK OS)

2) Insert the Master disk

3) Type: DUPE *.* (RETURN)

4) Follow the instructions on the Chromatics display

5) When finished store Master disk in a safe place.

B.4.0 REFERENCES

1) Hoenig, G., 1980, "A Survey of Alarms and Alarm Systems in the Process Industries", Warren Spring Laboratory, Department of Industry, Stevenage, England.

2) Singleton, W.T., 1974, "Man-Machine Systems", Penguin, London.

3) Fitter, M.J., 1979, "Dialogues for Users", Proc. Infotech State of the Art Conference on User-Friendly Systems, pp 5.1-5.21, 28-30 March 1979.

4) Chromatics Incorporated, 1978, "Disk Software Reference Manual", CG Series, Atlanta, pp 2.9-2.10.

5) Chromatics Incorporated, 1978, "Operators Manual', CG Series, Altanta.

TABLE B-1

ANALOGUE CONVERSION ALGORITHMS

User defined. See on-line software documentation.

APPENDIX C

USER'S GUIDE FOR THE ON-LINE COMPONENT OF THE ALARM HANDLING SYSTEM

APPENDIX C

TABLE OF CONTENTS

C.1.0 SYSTEM SPECIFICATION

C.1.1 Introduction

The on-line portion of the alarm handling system is capable of interpreting and implementing alarm data base structures as developed in the off-line component. Since the on-line system is a target machine for the data base, the operation of the on-line system has been greatly simplified. In normal operation the user must insure that the alarm data base has been installed correctly, otherwise the operation is fully automatic. The alarm handling system presented here is in prototype form, so obviously the user may find peculiarities with operation of the system contrary this document. However the greatest care has been taken to foresee such difficulties.

C.1.2 Equipment Available

As discussed in previous appendices it is usually the best policy when designing a computer based system to concentrate on the functional specification before considering the implementation of the system. In this way the design approach is not limited by the physical entities which are available, but instead is directed towards satisfying the overall system objectives. However, where resources are limited some account must be taken of the hardware that can be provided by the available resources.

Prototype design and development often requires changes to be made to the overall system objectives. The possible changes in system requirements necessitates the choice of hardware which not only meets preliminary requirements but which is also capable of servicing future needs. The most suitable equipment available for implementing the on-line component of the alarm handling system considered here is

discussed in the hardware documentation.

C.1.3 System Requirements

The on-line alarm handling system is an independent stand-alone computer system which is used to collect process data, process this data, generate alarm and display information as defined by the installed alarm data base. The user having developed such a data base off-line can install the alarm data base which defines the alarm functions to be performed by the on-line system in a specific application. Once installed the user need only start the on-line system and the operation is fully automatic. Alarms and alarm information will be generated as instructed in the alarm data base.

C.1.4 Organisation of the Appendix

The remainder of this appendix report presents a user's guide to the operation of the on-line alarm handling system. Other documentation is available describing details and configuration of the software and hardware systems.

C.2.0 On-Line USER'S GUIDE

C.2.1 Introduction

The on-line portion of the Alarm Handling System (AHS) is straight forward and easy to operate. It is recommended that a system manager be assigned to perform the startup and shutdown procedures in order to avoid difficulties with equipment and software. Once the system is secured no further attention should be required.

Described here are the basic procedures for startup and shutdown. If further details are required, a detailed descripion, listings and flowcharts can be found in the on-line software documentation.

C.2.2 Startup

The following steps must be executed to startup the alarm handling system:

1) Ensure all connections to the PDP 11/03 alarm handling computer are correct and secure. These include:

a) TT0: Console Terminal
b) TT1: Printer
c) TT2: Chromatics Link Line
d) TT3: Host Computer Link Line (if present)
e) DD0:/DD1: TU58 Tape Drive

Refer to the hardware document for more details.

2) Power up the Chromatics and PDP 11/03 computers. Also power up all peripherals.

3) Place the switches on the front panel of the PDP

11/03 in the following positions:

DC ON	on
ENABLE	on
LTC	off

The Media Active power bin must be switched OFF.

4) Insert Alarm Handling System cassette tape into tape drive DD1:

5) Make certain that the console terminal is in the On-line mode. An '@' symbol should be present indicating that the computer is in ODT (On-line Debugging Tool). Enter 173000G, the boot strap address. The alarm system will now boot off the tape drive DD1:

6) Once the boot is complete the program will respond:

```
Restart 00:00
DAY =
```

Before continuing switch the LTC switch located on the front panel of the PDP 11/03 to the ON position. Also switch all Media Plant interface equipment ON. This is essential since the alarm system accesses the Media interface at startup. If the Media interface is OFF the system will crash. In this situation the user must return to step 3 and try again. (If there is no Media Interface in your system consult the system manager to modify the POW powerup task.)

Once all equipment is switched ON, enter the day of the month. The program will continue:

```
MONTH =
YEAR  =
```

HOURS =
MINS =

Enter the day 1-31, month 1-12, year e.g. 82, hours 0-23, and minutes 0-59.

7) Now enter the following into the console terminal:

<cntrl>C
$LOG <return>

This protects the system from tampering. If the system manager requires to enter the system there are two user names suitable for this purpose. These names and passwords should be issued only with the system manager's approval.

User	Password	Protection
GOD	SSD	Top priority- should be used for software development, access is given to all job slots.
TOP	WSL	General access to AHS jobs required to operate system. Protection is provided to vital software.

It is recommended that for all operations in this document that the TOP user be entered to protect software task in the case of a miss-entry.

8) Next startup the Chromatics alarm display system. First insert the appropriate alarm display disk in the Chromatics floppy disk drive.

9) On the Chromatics keyboard enter:

(RESET)
(BOOT)
(BASIC)
Memory Size = &HB000 (RETURN)
DOS "LOAD DISPLAY"
RUN

10) Remove the AHS tape from the TU58 tape drive for security.

11) The alarm handling and display system is now installed and running. Refer to the COMAH task commands discussed later in this document for further details of how to activate the alarm handling functions.

C.2.3 Shutdown

1) Remove all tapes and floppy disks from the system drives.

2) Turn all power points off.

C.2.4 Use of COMMAH

The COMAH task allows the system manager to evoke engineering funcions with the system. A summary of the operations available are presented here. For more details refer to the on-line software documentation.

Via the console terminal TT0: the system manager must use the SWEPSPEED utility $ACT14 to activate the COMAH job slot. Once started the prompt '##' will indicate that the task is ready for input. All inputs consist of up to 2 character strings followed by a carriage return.

ED Enter EDIT mode. Overlay storage tape must be in DD0:

TI Print time and date to console.

RE Restart alarm handling system from scratch.

ST Stop the alarm handling system except for the watchdog task WD.

RU Run or 'warm start' the alarm handling system. Useful during fault finding after using the ST command.

X Exit COMAH task.

An error message may be encountered when entering the EDIT mode if the overlay tape is not inserted in drive DD0:.

C.2.5 Activating the Alarm Handling System

With the alarm handling system software installed and running in both the alarm handling computer and the display computer, the user can activate the alarm handling system functions as follows:

1) Install appropriate Alarm Data Base for application at hand from the off-line alarm handling system into the on-line alarm handling computer. Refer to the Off-Line User's Guide for details describing the procedure.

2) With the Alarm Data Base installed the system is ready to be activated. Ensure that the on-line alarm display package is installed and running in the display computer.

3) Log into the high security user name TOP with the password WSL.

4) Activate job slot 14, the COMAH task.

5) Wait for the '##' prompt.

6) Insert 'RU'. The alarm handling system functions are now running. Data Acquisition has now started. The user must wait 15 minutes to ensure that all data acquisition units are initialized after which time the alarm handling system is primed. Alarms can now be generated in accordance with the event and alarm definitions in the alarm data base.

7) Insert 'X' to exit the COMAH task.

8) Most Important: Log out to maintain security of the system. Enter:

```
<cntrl> C
$LOG  <return>
```

Note that the alarm display package provided with the system contains a display personality module for a simple paging display format. Sample alarm message texts are provided for demonstration purposes. If the user requires other forms of display, display personality modules must be written. Software hooks are provided for this purpose. See the On-Line Software Documentation for further details.

C.2.6.0 Use of EDIT

The EDIT command allows the system manager to make simple modifications on-line to the alarm data base. Use this program with care. A good understanding of the structure of the alarm data base is required. Incorrect entries will cause malfunction of the alarm handling system. It is recommended that the user read the on-line document concerning the EDIT task and the Alarm Data Base before proceeding.

1) Make certain that the overlay tape AHS OVERLAY is inserted in tape drive DD0: and stop any display or off-line tasks in the attached Chromatics.

2) Activate alarm handling command task COMAH.

3) Enter 'ED' after the ## prompt.

4) Wait until the overlay file is retrieved from the tape drive and installed in the system.

5) The editor prompt > indicates that the system is ready for edit mode sub-commands described below.

6) When editing is complete, enter the sub-command 'X'. Wait until the overlay is complete. The system will be returned to the COMAH task.

7) The system must be restarted after a modification to the alarm data base has been made. The array sizes in the SETUP task may need adjusting.

C.2.6.1 Edit Sub-Commands

P Print alarm data base to the T43 Printer on device TT1:.

L List alarm data base to console.

I Insert a new element into the data base. This function is followed by:

ELEMENT NUMBER:

The user must enter and element number at which the insertion is to be made. The element number is the alarm data base array subscipt. The present contents of the element entered will be shifted to the next higher element number. The same occurs for any elements above the insertion point.

The program will display the present contents of the element. If a new entry is to be made the user enters an '='. The program will respond with an '=' prompt awaiting input. Any numeric values are acceptable. A carrage return completes the entry. To exit the insert mode enter a RETURN. The insert sub-command readjusts the data base size in element one.

Example:

```
>I
ELEMENT NUMBER = 25
25> [old contents]  =
                    =  [new entry]
26> [old contents]
```

R Replace a data base element. This function is

followed by:

```
>R
ELEMENT NUMBER = [number]
```

The user must enter an element number at which a modification may be made. The contents of the element entered will be displayed as follows:

```
25>  321.0      =
                = [new entry]
25> [new entry]
```

By entering a '+' the program steps to the next data base element, a '-' decrements the data base pointer and a (RETURN) terminates the function.

```
25>  321.0      +
26>  432.0      -
25>  321.0      (RETURN)
>
```

D Delete a data base element. This function removes an element from the alarm data base. The remaining elements are shifted downwards to take up the space in the data base. The data base size in element one is also decremented. The program responds:

```
>D
ELEMENT NUMBER = 25
25>  321.0      Y
25>  432.0      (RETURN)
>
```

Any response other than 'Y' aborts the function.

X Exit the alarm data base editor, remove the overlay, restart necessary program tasks and return to the alarm handling command task COMAH.

C.2.6.2 Errors

Error messages may be encountered when entering the edit mode. The error message will be generated by the SWEPSPEED system indicating an overlay error. The principal causes of this error are:

1) Alarm Handling Overlay files not inserted correctly in tape drive DD0:

2) Tape drive not properly connected to the system or not powered up.

3) The Display package in the Chromatics computer was transferring data to the alarm handling system at the time of calling the Edit command.

Correct difficulty, enter the COMAH task again and repeat procedure.

If errors occur when exiting the EDIT mode consult the system manager or re-boot the alarm handling software. Sorry about that!

During editing the only error message generated by the program is the following:

RE-SIZE ADB

This message is generated when an attempt is made to insert an element into an alarm data base which has filled all available space in the %A() alarm data base array. No

further entries may be made. This size may be increased by modifying the SETUP task in job slot 16. Consult the system manager.

APPENDIX D

SOFTWARE DESCRIPTION
FOR THE OFF-LINE COMPONENT
OF THE ALARM HANDLING SYSTEM

APPENDIX D

TABLE OF CONTENTS

OFF-LINE SOFTWARE DESCRIPTION

D.1.0 INTRODUCTION

As described in the user's guide for the off-line component of the alarm handling system, there are three BASIC programs which comprise the software. The alarm data base generator OFLAD functions as the command program calling up all functions in the off-line system. It also performs as the editor for the raw data files which later are compiled by the COMP program into a coded alarm data base. The third program is the XFER or transfer program which loads the compiled data base into the on-line alarm handling computer.

Presented here is an explanation of the structure of these three programs and how they interact with each other. The programs are written entirely in Microsoft BASIC and are intended for use in the Chromatics CG 1999 intelligent colour graphics terminal. Pecularities in program statements will be due to Chromatic's specific instructions. The reader should refer to the Chromatic's user's manuals provided by the manufacture for more details of these statements.

D.2.0 OFLAD

The Off-Line Alarm Data base generator (OFLAD) is the core program in the off-line system. The program provides a system command structure and raw data file management and editing facilities. The program is best explained by walking through the flowchart. Because of the highly interactive nature of this program the off-line user's guide is also a software description. As a result, only

additional information is presented here which will clarify the program listings.

D.2.1 Setup 100-199

The setup section is executed each time the program is run. The screen colours and windows are set. The baud rate is set match the T-43 printer on port SIO-0. As much string space as possible is cleared to make maximum room for data files in the form of string arrays. These file string arrays are also dimensioned here. The variable D is used to set the maximum data file or array size. The arrays are as follows:

R$(D,8)	Data Aquisition data
S$(D,8)	Event Definition data
A$(D,5)	Alarm Definition data
CO$(11)	Command ref file
DI(100)	Disk directory

Finally a directory listing of the contents of the currently loaded floppy disk is generated on the display.

D.2.2 Command Monitor 200-259

The command monitor prompts the user for an instruction entry. The entry is checked and the program control is temporarily transfered to the appropriate subroutine. A listing of the available commands can be found in the off-line user's guide.

D.2.3 Load Sub 260-359

There are three 'working files' in the program represented by the arrays R$, S$, and A$. In order for the user to use these work areas, the file must be loaded, i.e.,

identified with a raw data file number. The load command makes this assignment. If the file number already exists on the floppy disk directory listing, the file is loaded into the appropriate working file area. Thus the arrays R$, S$, or A$ are filled with raw data information. Once the files are loaded, subsequent additions or editing may be performed as required. Examination of the program listing will clarify this explanation. It should be noted that the variables FR, FS, and FA are used as flags to indicate that the working files are loaded (1= loaded, 0= empty). The variables NR, NS, and NA are used to store the total number of entries made in the working files.

D.2.4 Q$ Sub 360-369

This internal service routine is used to convert a string response given by the user in Q$ into a number value returned in Q.

D.2.5 Files Loaded Sub 370-379

The command routine lists the numbers of the working files which are currently loaded in the system.

D.2.6 Compile Sub 380-385

Turns off the screen window and executes a SUBMIT file to load in the COMP program. Refer to Chromatics manuals for SUBMIT command.

D.2.7 Help Sub 400-460

Lists available OFLAD commands.

D.2.8 Change Sub 450-460

The edit flag FC is set to 1 and the program control is temporarily transfered to the Item Sub. The edit flag FC is set to zero and program control returned to the Monitor section.

D.2.9 List Sub 470-499

Depending upon the user's reponse to "WHICH ONE?", the routine branches to the appropriate subroutine. These routines print headers describing working file contents and then prints out the file contents.

D.2.10 Item Sub 500-620

The command allows a particular working file entry if the change flag is set or places the working file pointer to the next available entry point in the file for data insertion. When the change flag is set the routine asks for the name or first section of a particular file entry. The program scans the working file to find the entry. If no entry can be found, the program reports this. With the working file pointer set the program control is temporarily transfered to the appropriate data entry routine.

D.2.11 Help Item Sub 630-660

Lists available subcommands in the Item Sub.

D.2.12 Store Sub 700-790

Via several subcommands, this routine provides options for storing the contents of working files onto the floppy disk store. The working files can be deleted, stored, stored with number change or the working file can be cleared. The routine branches to the appropriate subroutine which provide the appropriate file name and number to the

disk operating system and then the directory is updated.

D.2.13 Delete Sub 791-799

This command allows existing files on a currently loaded floppy disk to be deleted. The file number is converted into a file name for the disk operating system. The directory is updated and the floppy disk is compressed.

D.2.14 Dir Sub 800-810

This routine evokes the directory listing routine located in the Setup section.

D.2.15 Transfer Sub 820-830

This command turns off the display window and SUBMITs the file LOADXFER which subsequently loads the XFER program into the system.

D.2.16 Data Entry

The remainder of the program is commited to data entry routines for the various working files. The working files as mentioned are in the form of string arrays. Each array is two dimensional. The first dimension is the entry number and the second is the section within the entry. The program prints the description of the section, then the current contents and finally asks for input. An examination of the program listing clearly shows the content of each of the entry section elements in the working file arrays. The data entry routines line numbers are as follows:

Data Acquisition	1000 - 1230
Event Definitions	2000 - 2110
Alarm Definitions	4000 - 4090

D.3.0 COMP

D.3.1 Introduction

The COMP or compiler program is the most complex of the three off-line programs. The program is loaded into the computer via a submit file as evoked by the OFLAD program module. The purpose of the COMP program is to convert the data stored in the Data Acquisition, Event, and Alarm files into a coded Alarm Data Base suitable for loading into the on-line alarm handling system. File entries are checked and cross checked to insure that all data syntax was valid and correct. As with the OFLAD program description it is best to describe the program details by walking through the flowchart and listing. Many aspects of the OFLAD and COMP are similar since the data files generated in the OFLAD program are used by the COMP program. Variables used for specific functions are similar if not the same as in OFLAD.

D.3.2 Setup 50-140

This section sets the baud rate for port SIO-0 to 300 for the T-43 printer. The screen window is set, arrays dimensioned and a file directory is printed to the screen of the contents of the currently loaded floppy disk.

D.3.3 Input 150-175

This program section issues prompts to the user for the data files to be used to build an alarm data base. The following file numbers must be specified: DA, EP, AG and the destination ADB file. The file numbers are checked for validity and and their existence on the currently loaded floppy disk. The user is also asked if a hard copy print

out is required. If so the logic output device A is set to the display screen as well as I/O port SIO-0.

D.3.4 DA Processing 180-590

This program section represents the Data Acquisition file processing. The section is comprised of a variety of routines which convert the DA file data into alarm data base coded information. The program records the number of compilation errors that occur in variable ER. First the program reconstructs the raw data files from the files on disk in string array R$(,). This file is printed verbatim to the screen. N is the number of entries in the DA file.

The program next sorts the file according to the plant code alphanumeric order [R$(N,1)]. Next, a check is made to see if there is any duplication of plant codes. Any error is marked by a '*' next to the duplicate plant code. An intermediate listing is made of the file. An examination of the scan rates [R$(N,7)] is made and a sort is made to place the list of entries in scan order. Next the list is sorted according to priorities within each scan rate group. Another listing is made of the file. DA related elements in the Alarm Data Base header are now calculated and placed in the temporary header array. Refer to the alarm data base documentation for more details regarding the alarm data base contents.

Errors which do not appear on the listings as '*' are recorded in EO and an error message is issued. A good example for this case is in the next section of the compiler where data type and range evaluations are performed. Here the descriptions of data type are converted to a numeric code representing the data type as follows:

0 = Binary

1	=	Inverted Binary
2	=	Analogue Conversion 0
3	=	Analogue Conversion 1
4	=	and so on

This section combines information in array elements R$(N,4) and R$(N,5) into the above code placed in R$(N,4). Range information if present is located in R$(N,6) as a combined low/high text string. The string is separated into its numeric values. The low value is then placed in R$(N,5) and the high value in R$(N,6).

Next the input device number in R$(N,3) is examined to see if it is valid. The value is decremented by 1 and replaced in R$(N,3).

The significant change value remains in R$(N,9). After compilation R$(N,2) contains the DA data storage location and is not presently used. No further processing is done on the DA file until the alarm data base is built later.

D.3.5 Event Processing 600-865

The program next compiles the event data. The event data file is retrieved form the floppy disk. NS contains the total number of entries and is used to dimension the event array S$(,). As with the DA files, the raw event data file is printed to the screen. Errors are summed for this section in variable ES and errors are marked on the listing with the '*' as well.

The file entries are sorted according to the alphanumeric order of event names. The file entries are next sorted according to event types. Duplicate event names generate an error and are marked on the listing. Next the plant codes are checked against plant codes in the DA file

R$(N,1). If a plant code does not exist, the plant code is error marked. A listing is now made to the display screen.

Finally the event type in S$(I,3) is checked for validity and converted to codes as follows:

1	=	ON
2	=	OFF
3	=	XLO
4	=	LO
5	=	HI
6	=	XHI
7	=	TREND
8	=	DEVIATION
9	=	TREND DEVIATION

An error message is made if an illegal event type is detected. No more processing is performed until the alarm data base is built.

D.3.6 Process Alarms 870-1240

Alarm definitions are next compiled. The alarm data file is retrieved from the floppy disk. NA contains the total number of entries and is used to dimension the alarm array A$(,). The raw data contained in this file is printed to the screen. Errors are summed for this program section in variable EA. Errors are marked on the listing with a '*'.

The file entries are sorted according to the alphanumeric order of alarm names. Duplicate alarm names generate an error and are marked on the listing.

Next the Boolean expressions representing the ON condition and OFF condition statements are translated into

Reverse Polish Notation (RPN). During the translation aspects of the validity of the Boolean expression are also checked.

The Boolean processor performs the conversion to RPN. There are two passes made of the Boolean expression through the processor. The first is used to check the validity of the Boolean expressions. Both the construction of the expression and the existence to the events used in the expression are checked. Later, when the alarm data base is built, the second pass is made. During this pass the Boolean operators and events are coded.

The Boolean processor actually converts algebraic Boolean notation into RPN. The expressions for the ON or OFF alarm condition expressions are examined character by character. A temporary stack B$(M) is used to store operators temporarily. The resulting expression is placed in stack AA$(J). The stack pointers M and J indicate the next available stack location. Referring to figure below the process operates as follows:

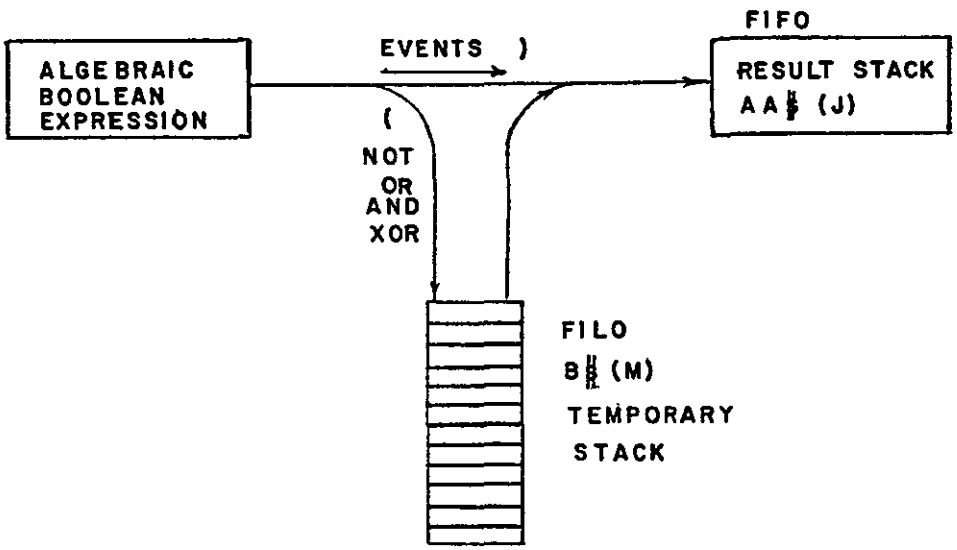


Figure: Using a Stack to convert expression to RPN

Individual characters are taken from the condition expression string and are stored in X$. If the contents of X$ does not appear to be an operator or an event name (identified by a leading and following space character) then X$ is added to T$ until the contents of T$ is either a recognizable operator and if not it is an event. Operators are stored consecutively in the temporary stack $B(M). Event are placed directly in the result stack AA$(J). When all elements in the condition statement are processed the operators are transferred from the temporary stack to the result stack in FILO (first in last out) fashion. The result in the stack AA$(J) is now in RPN with the first entry as the left hand compenent of the expression.

Unfortunately nesting by using brackets causes additional complications. Expressions within brackets are intermediate results so when a left hand bracket is encountered it is passed directly to stack B$(M). When a right hand bracket is encountered it is passed directly to the result stack AA$(J). Operators are then retrieved off the B$(M) stack until a left hand bracket is encountered after which the next component in the original expression is evaluated.

The operator NOT is also not really a true Boolean operator. As a result any time an event or nested expression is passed on to the result stack, the B$(M) stack must be examined to see if there is a NOT on the top of the stack.

The RPN result of the Boolean expressions are printed to the screen along with the alarm definition information.

D.3.7 Build Data Base Header 1300-1330

At this point the alarm data base header is built. The appropriate values are inserted in the data base array. Refer to the Alarm Data Base documentation for further details.

D.3.8 Comp Fail Check 1340-1355

This section generates a status report to the user. Up to this point in the compilation is referred to as the first pass. If errors have occured, the second pass is aborted and the program restarts the OFLAD program module.

D.3.9 Build Data Base 1360-1500

Elements of the alarm data base are now ordered and sent to an opened floppy disk file. Elements are printed to the open file in the order in which the elements appear in the final data base. Refer to the Alarm Data Base documentation for further details. The data is sent as follows:

1) Data Base Header from DB%()

2) Data Acquisition Definitions from R$(,)

3) Event Definitions from S$(,)

4) Alarm Definitions from A$(,) after condition expressions are reprocessed with operators coded and event names substituted with event locations in the event status image.

If all goes well the alarm data base file ADB_.DAT is closed on the floppy disk. A message is printed stating

that the compilation is OK. The directory is updated, the hardcopy output is turned off, and the OFLAD program module is re-installed.

D.4.0 TRANSFER

D.4.1 Introduction

TRANSFER, the third program module in the off-line system transfers an alarm data base, as compiled by the COMP program, to the on-line system in the PDP 11/03 alarm handling computer. The program complements the alarm handling on-line task LOAD. Refer to the documentation for this task for further details. In order for the program to function correctly, the Chromatics computer must be properly installed, that is, port SIO-0 must be connected to port TT2: on the PDP 11/03 and the alarm handling software must be up and running. The TRANSFER program is entered via an OFLAD command and a LOADXFER submit file.

D.4.2 Setup 50-165

This program section informs the user of the conditions of use of the program. the program also questions the user about which alarm data base stored on the currently loaded floppy disk is to be transfered to the on-line system. A listing of the available alarm data base files is also given.

D.4.3 Ask for Overlay 200-250

First the program requests the LOAD overlay task to be installed in the on-line system. This is done by issuing an 'L' to the on-line system. Remember that in normal running mode the alarm handling system computer uses device TT2: as

an output line for display commands to the Chromatics in the on-line mode. This means that when issuing an 'L' that the data packet is received by the CHROM link task and passed on to the DISPlay task where it is recognized as a request for the LOAD task. The CHROM link task is shut down and the LOAD task is installed over the DISPlay task.

A timeout error is set in the event that the on-line system does not respond. TRANSFER will try up to 5 times to establish contact. If stil no response indicating that the LOAD task is installed is received, a link error occurs and the program is aborted.

D.4.4 Start Transfer 300-410

With communication established, recognized by a '*' response from the LOAD task, the timeout time for a response is decreased and 'READY' is sent down line. Next several null data packets are sent to clear the line. the first data sent is the alarm data base size, the first element in the alarm data base. This allows the on-line system to see if there is enough room for the incoming alarm data base. The size is also used to set the number of data transfers to be made in the TRANSFER program.

Any data packet which is not equal to "0" received from the on-line system is interpretted as a transfer error and the program aborts. Also if a data packet is sent and no response is received, a transfer error is initiated.

D.4.5 Close Down 450-470

Once all data has been transfered, the program returns to the OFLAD program module. Several additional null data packets are sent to clear the link line.

D.4.6 Prod 500-550

This routine is called each time data is to be received fom the PDP 11/03. the program turns off the output section of the port SIO-0 to prevent any echo down line from an input statement. Once input is received the output port is turned back on.

D.4.7 Float Check 600-750

This routine is called each time an alarm data base entry is sent to the on-line system. The alarm data base is stored in a real array in the on-line system. Also the LOAD task is only capable of dealing with real values. The float check routine examines the numeric data in the alarm data base files and converts it if necessary to a floating point format acceptable to the on-line system. When data is stored in .DAT files on the floppy disk, often extra space characters are present in the data file entries. The CLR SPC subroutine removes these spaces before the float check is performed.

D.4.8 ERR Link 1000-1020

If a link error occurs this error routine will print an error message, ring the bell, and send program control to the Close Down routine. This occurs after 5 attempts are made to establish the link with the on-line system.

D.4.9 ERR Comm 2000-2020

If a communication or transfer error occurs this error routine will print an error message. ring the bell, and send program control to the Close Down routine. This occurs after 5 attempts are made to re-establish the transfer communication protocol.

D.5.0 LISTINGS

```
10 ' ---- OFF-LINE ALARM HANDLING DATA BASE GENERATOR
20 ' ---- VER. 3.1
30 ' ---- By G. Hoenig, LUT, FEB. 1982
40 '
100 PRINTCHR$(12);"~C2"
102 CLEAR 2000:PRINTCHR$(27);"OA0"
103 PRINT CHR$(27);"R04"    '---SET BAUD 300
104 CLEAR (FRE(X)-4000)
105 D=50
110 DIMR$(D,9),S$(D,8),A$(D,5),CO$(11),DI(100)
115 DOS"ARYLOAD DI,DI"
120 PRINT"---- FILE DIRECTORY ----":PRINT"FILE NUMBER - NO. OF ELEMENT
S"
125 PRINT:PRINT"DA":FORI=1TO25:GOSUB140:NEXTI
127 PRINT:PRINT"EP":FORI=26TO50:GOSUB140:NEXTI
129 PRINT:PRINT"AG":FORI=51TO75:GOSUB140:NEXTI
130 PRINT:PRINT"ADB":FORI=76TO100:GOSUB140:NEXTI:PRINT:IFX=1THENRETURN
ELSE200
140 IFDI(I)<>0THENPRINTI;"-";DI(I),:RETURNELSERETURN
200 '
210 ' ---- COMMAND MONITOR
220 '
225 PRINT:CO$(1)="EN":CO$(2)="CH":CO$(3)="DI":CO$(4)="LI":CO$(5)="HE":
CO$(6)="LO":CO$(7)="ST":CO$(8)="FI":CO$(9)="CO":CO$(10)="DE":CO$(11)="
TR"
230 LINEINPUT"~C6COMMAND:~C3";C$:PRINT"~C2":C$=LEFT$(C$,2):I=0
235 IFC$=CO$(9)THEN380
240 I=I+1
250 IFI<12THENIFC$=CO$(I)THENONIGOSUB500,450,800,470,400,260,700,370,3
80,791,820:GOTO230ELSE240
252 IFC$<>""THENPRINT"~C4Syntax error~C2";CHR$(7)
255 PRINT:GOTO230
260 ' ---- LOAD SUB
270 IFFR=1THENPRINT"DA FILE";DR;"LOADED.":GOTO300
275 Q$="":LINEINPUT"DA FILE NUMBER: ";Q$:GOSUB360:IFQ$=""ORQ$="+"THEN3
00
285 IFQ<1ORQ>25THEN275
290 FR=1:DR=Q:IFDI(Q)=0THENPRINT"NEW FILE":GOTO300
295 DOS"OPEN 5 R DA"+Q$+".DAT":NR=DI(Q)
296 FORI=1TONR:FORJ=1TO9:LINEINPUT#5;R$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
300 IFFS=1THENPRINT"EP FILE";DS;"LOADED.":GOTO330
305 Q$="":LINEINPUT"EP FILE NUMBER: ";Q$:GOSUB360:IFQ$=""ORQ$="+"THEN3
30
315 IFQ<26ORQ>50THEN305
```

```
320 FS=1:DS=Q:IFDI(Q)=0THENPRINT"NEW FILE":GOTO330
325 DOS"OPEN 5 R EP"+Q$+".DAT":NS=DI(Q)
326 FORI=1TONS:FORJ=1TO7:LINEINPUT#5;S$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
330 IFFA=1THENPRINT"AG FILE":DA;"LOADED.":GOTO358
335 Q$="":LINEINPUT"AG FILE NUMBER: ";Q$:GOSUB360:IFQ$=""ORQ$="*"THEN3
58
340 IFQ<51ORQ>75THEN335
345 FA=1:DA=Q:IFDI(Q)=0THENPRINT"NEW FILE":GOTO358
350 DOS"OPEN 5 R AG"+Q$+".DAT":NA=DI(Q)
355 FORI=1TONA:FORJ=1TO5:LINEINPUT#5;A$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
358 PRINT:RETURN
360 '--Q$ SUB
362 IFQ$=""THENQ=0:RETURN
364 M=0:FORI=1TO100:IFQ$=STR$(I)THENM=1:I=100
365 NEXTI:IFM<>0THENQ$="*":RETURN
366 Q=VAL(Q$):RETURN
370 ' ---- FILES LOADED SUB
372 IFFR=1THENPRINT"DA FILE";DR
373 IFFS=1THENPRINT"EVENT PROCESSOR FILE";DS
374 IFFA=1THENPRINT"ALARM FILE";DA
375 IFFR+FS+FA=0THENPRINT"NO FILES LOADED."
376 PRINT:RETURN
380 ' ---- COMPILE SUB
385 PRINTCHR$(27);"0AF":DOS"SUBMIT LOADCOMP":END
400 ' ---- HELP COMMAND SUB
410 PRINT"~C3EN~C2TER":PRINT"~C3CH~C2ANGE":PRINT"~C3DI~C2RECTORY"
420 PRINT"~C3LI~C2ST":PRINT"~C3LO~C2AD":PRINT"~C3ST~C2ORE":PRINT"~C3HE
~C2LP"
425 PRINT"~C3FI~C2LES LOADED":PRINT"~C3CO~C2MPILE":PRINT"~C3DE~C2LETE"
:PRINT"~C3TR~C2ANSFER"
430 PRINT:RETURN
450 ' ---- CHANGE SUB
460 FC=1:GOSUB500:FC=0:RETURN
470 ' ---- LIST SUB
471 PRINT:LINEINPUT"~C6WHICH ONE?~C2";Q$:Q$=LEFT$(Q$,1):IFQ$="D"THENGO
SUB480
472 IFQ$="E"THENGOSUB490
473 IFQ$="A"THENGOSUB494
474 RETURN
480 IFFR=0THEN499ELSEPRINT:PRINT"PLANT CODE","NAME",,"I/P DEV","TYPE",
"ALG NO.","RANGE","SCAN","PRIORITY"
481 PRINT:FORI=1TOD:IFR$(I,1)=""THENRETURNELSEFORJ=1TO8:PRINT R$(I,J),
:IFJ=2ANDLEN(R$(I,J))<14THENPRINT"",
482 NEXTJ:PRINT:PRINTTAB(85);"(";R$(I,9);")":NEXTI:RETURN
490 IFFS=0THEN499ELSEPRINT:PRINT"PLANT CODE","EVENT NAME",,"TYPE","L.
LIMIT","L. HYS.","U. LIMIT","U. HYS."
491 PRINT:FORI=1TOD:IFS$(I,1)=""THENRETURNELSEFORJ=1TO7:PRINTS$(I,J),:
IFJ=2ANDLEN(S$(I,J))<14THENPRINT"",
492 NEXTJ:PRINT:NEXTI:RETURN
494 IFFA=0THEN499
495 FORI=1TOD:IFA$(I,1)=""THENRETURNELSEPRINT"ALARM NAME:    ";A$(I,1)
```

```
320 FS=1:DS=Q:IFDI(Q)=0THENPRINT"NEW FILE":GOTO330
325 DOS"OPEN 5 R EP"+Q$+".DAT":NS=DI(Q)
326 FORI=1TONS:FORJ=1TO7:LINEINPUT#5;S$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
330 IFFA=1THENPRINT"AG FILE";DA;"LOADED.":GOTO358
335 Q$="":LINEINPUT"AG FILE NUMBER: ";Q$:GOSUB360:IFQ$=""ORQ$="*"THEN3
58
340 IFQ<51ORQ>75THEN335
345 FA=1:DA=Q:IFDI(Q)=0THENPRINT"NEW FILE":GOTO358
350 DOS"OPEN 5 R AG"+Q$+".DAT":NA=DI(Q)
355 FORI=1TONA:FORJ=1TO5:LINEINPUT#5;A$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
358 PRINT:RETURN
360 '--Q$ SUB
362 IFQ$=""THENQ=0:RETURN
364 M=0:FORI=1TO100:IFQ$=STR$(I)THENM=1:I=100
365 NEXTI:IFM<>0THENQ$="*":RETURN
366 Q=VAL(Q$):RETURN
370 ' ---- FILES LOADED SUB
372 IFFR=1THENPRINT"DA FILE";DR
373 IFFS=1THENPRINT"EVENT PROCESSOR FILE";DS
374 IFFA=1THENPRINT"ALARM FILE";DA
375 IFFR+FS+FA=0THENPRINT"NO FILES LOADED."
376 PRINT:RETURN
380 ' ---- COMPILE SUB
385 PRINTCHR$(27);"0AF":DOS"SUBMIT LOADCOMP":END
400 ' ---- HELP COMMAND SUB
410 PRINT"~C3EN~C2TER":PRINT"~C3CH~C2ANGE":PRINT"~C3DI~C2RECTORY"
420 PRINT"~C3LI~C2ST":PRINT"~C3LO~C2AD":PRINT"~C3ST~C2ORE":PRINT"~C3HE
~C2LP"
425 PRINT"~C3FI~C2LES LOADED":PRINT"~C3CO~C2MPILE":PRINT"~C3DE~C2LETE"
:PRINT"~C3TR~C2ANSFER"
430 PRINT:RETURN
450 ' ---- CHANGE SUB
460 FC=1:GOSUB500:FC=0:RETURN
470 ' ---- LIST SUB
471 PRINT:LINEINPUT"~C6WHICH ONE?~C2";Q$:Q$=LEFT$(Q$,1):IFQ$="D"THENGO
SUB480
472 IFQ$="E"THENGOSUB490
473 IFQ$="A"THENGOSUB494
474 RETURN
480 IFFR=0THEN499ELSEPRINT:PRINT"PLANT CODE","NAME",,"I/P DEV","TYPE",
"ALG NO.","RANGE","SCAN","PRIORITY"
481 PRINT:FORI=1TOD:IFR$(I,1)=""THENRETURNELSEFORJ=1TO8:PRINT R$(I,J),
:IFJ=2ANDLEN(R$(I,J))<14THENPRINT"",
482 NEXTJ:PRINT:PRINTTAB(85);"(";R$(I,9);")":NEXTI:RETURN
490 IFFS=0THEN499ELSEPRINT:PRINT"PLANT CODE","EVENT NAME",,"TYPE","L.
LIMIT","L. HYS.","U. LIMIT","U. HYS."
491 PRINT:FORI=1TOD:IFS$(I,1)=""THENRETURNELSEFORJ=1TO7:PRINTS$(I,J),:
IFJ=2ANDLEN(S$(I,J))<14THENPRINT"",
492 NEXTJ:PRINT:NEXTI:RETURN
494 IFFA=0THEN499
495 FORI=1TOD:IFA$(I,1)=""THENRETURNELSEPRINT"ALARM NAME:    ";A$(I,1)
```

```
;" (";A$(I,2);")"
496 PRINT"CONDITION ON:  ";A$(I,3):PRINT"CONDITION OFF: ";A$(I,4):PRIN
T"PERSISTENCY:   ";A$(I,5):PRINT:NEXTI:PRINT:RETURN
499 PRINT:PRINT"NO FILE LOADED.":PRINT:RETURN
500 ' ---- ITEM SUB
510 LINEINPUT"~C6ITEM:~C3";C$:C$=LEFT$(C$,1)
515 IFFC=1THENGOTO530
520 IFC$="ALL"THENGOSUB1000:GOSUB2000:GOSUB4000:GOTO230
530 IFC$="D"THENN=0ELSE560
535 IFFC=1THENPRINT:LINEINPUT"~C6PLANT CODE TO BE CHANGED:~C3";CC$
540 N=N+1:IFN=D+1ANDFC=1THENPRINT"~C4Can't Find":RETURNELSEIFN=D+1THEN
PRINT"~C4Array Full":GOTO620
545 IFFC=1THENIFCC$=R$(N,1)THENGOSUB1000:RETURN:ELSE540
550 IFR$(N,1)=""ORLEFT$(R$(N,1),1)=" "THENGOSUB1000:GOTO200:ELSE540
560 IFC$="E"THENN=0ELSE580
562 IFFC=1THENPRINT:LINEINPUT"~C6EVENT NAME TO BE CHANGED:~C3";CC$
564 N=N+1:IFN=D+1ANDFC=1THENPRINT"~C4Can't Find":RETURNELSEIFN=D+1THEN
PRINT"~C4Array Full":GOTO620
566 IFFC=1THENIFCC$=S$(N,2)THENGOSUB2000:RETURN:ELSE564
568 IFS$(N,1)=""ORLEFT$(S$(N,1),1)=" "THENGOSUB2000:GOTO200ELSE564
580 IFC$="A"THENN=0ELSE600
582 IFFC=1THENPRINT:LINEINPUT"~C6ALARM NAME TO BE CHANGED:~C3";CC$
584 N=N+1:IFN=D+1ANDFC=1THENPRINT"~C4Can't Find":RETURNELSEIFN=D+1THEN
PRINT"~C4Array Full":GOTO620
586 IFFC=1THENIFCC$=A$(N,1)THENGOSUB4000:RETURNELSE584
588 IFA$(N,1)=""ORLEFT$(A$(N,1),1)=" "THENGOSUB4000:GOTO200ELSE584
600 IFC$="H"THENGOSUB630:GOTO500
610 IFC$<>""THENPRINT:PRINT"~C4Syntax error~C2";CHR$(7)
620 PRINT:FC=0:GOTO200
630 ' ---- HELP ITEM SUB
640 PRINT:IFFC=0THENPRINT"~C3ALL"
650 PRINT"~C3D~C2ATA ACQ":PRINT"~C3E~C2VENT PROCESSOR"
660 PRINT"~C3A~C2LARM":PRINT"~C3H~C2ELP":PRINT:RETURN
700 ' ---- STORE SUB
705 PRINT:PRINT"RETURN = STORE NO CHANGE IN FILE NUMBER."
710 PRINT"'N'    = DO NOT STORE."
715 PRINT"'NF'   = STORE WITH NEW FILE NUMBER."
720 PRINT"'K'    = KILL WORKING FILE.":PRINT
725 IFFR=1THENPRINT"STORE DA";DR;:LINEINPUT"?";SS$:ELSE733
727 IFSS$=""THENM=0:DOS"OPEN 5 W *.DOS"ELSE731
728 FORI=1TOD:FORJ=1TO9:IFLEFT$(R$(I,1),1)<>" "ANDR$(I,1)<>""THENM=M+1
:PRINT#5;R$(I,J);CHR$(13);
729 NEXTJ:NEXTI:IFM<>0THENDOS"CLOSE 5 DA"+MID$(STR$(DR),2,2)+".DAT"ELS
EDOS"CLOSE 5"
730 DI(DR)=INT(M/9):DOS"ARYSAVE DI,DI":DOS"COMPRESS":GOTO733
731 IFSS$="NF"THENINPUT"ENTER NEW NUMBER";Q:IFQ<26ANDQ>0THENDR=INT(Q):
SS$="":GOTO727ELSE733
732 IFSS$="K"THENERASER$:DIMR$(D,9):FR=0
733 IFFS=1THENPRINT"STORE EP";DS;:LINEINPUT"?";SS$:ELSE750
735 IFSS$=""THENM=0:DOS"OPEN 5 W *.DOS"ELSE743
737 FORI=1TOD:FORJ=1TO7:IFLEFT$(S$(I,1),1)<>" "ANDS$(I,1)<>""THENM=M+1
```

```
:PRINT#5:S$(I,J);CHR$(13);
739 NEXTJ:NEXTI:IFM<>0THENDOS"CLOSE 5 EP"+MID$(STR$(DS),2,2)+".DAT"ELS
EDOS"CLOSE 5"
741 DI(DS)=INT(M/7):DOS"ARYSAVE DI,DI":DOS"COMPRESS":GOTO750
743 IFSS$="NF"THENINPUT"ENTER NEW NUMBER";Q:IFQ<51ANDQ>25THENDS=INT(Q)
:SS$="":GOTO735ELSE750
745 IFSS$="K"THENERASES$:DIMS$(D,8):FS=0
750 IFFA=1THENPRINT"STORE AG";DA;:LINEINPUT"?";SS$:ELSE790
752 IFSS$=""THENM=0:DOS"OPEN 5 W *.DOS"ELSE760
754 FORI=1TOD:FORJ=1TO5:IFLEFT$(A$(I,1),1)<>" "ANDA$(I,1)<>""THENM=M+1
:PRINT#5:A$(I,J);CHR$(13);
756 NEXTJ:NEXTI:IFM<>0THENDOS"CLOSE 5 AG"+MID$(STR$(DA),2,2)+".DAT"ELS
EDOS"CLOSE 5"
758 DI(DA)=INT(M/5):DOS"ARYSAVE DI,DI":DOS"COMPRESS":GOTO790
760 IFSS$="NF"THENINPUT"ENTER NEW NUMBER";Q:IFQ<76ANDQ>50THENDA=INT(Q)
:SS$="":GOTO752ELSE790
765 IFSS$="K"THENERASEA$:DIMA$(D,5):FA=0
790 RETURN
791 ' ---- DELETE SUB
792 X=1:GOSUB120:X=0:Q=0:INPUT"DELETE FILE NUMBER (1-100)";Q:Q=INT(Q):
IFQ<1ORQ>100THENRETURN
793 IFDI(Q)=0THENPRINT"NO FILE":PRINT:RETURN
794 GOSUB795:DOS"KILL "+F$+MID$(STR$(Q),2,2)+".DAT":GOSUB799:RETURN
795 IFQ<26THENF$="DA":RETURN
796 IFQ<51THENF$="EP":RETURN
797 IFQ<76THENF$="AG":RETURN
798 F$="ADB":RETURN
799 DI(Q)=0:DOS"ARYSAVE DI,DI":DOS"COMPRESS":RETURN
800 ' ---- DIR SUB
810 X=1:GOSUB120:X=0:PRINT:RETURN
820 ' ---- TRANSFER
830 PRINTCHR$(27);"0AF":DOS"SUBMIT LOADXFER":END
1000 '
1010 ' ---- DA UNIT - ROUTINE FOR ENTERING DATA AQUISITION INFORMATION
1020 '
1025 IFFR=0THEN499
1030 PRINT:PRINT:PRINT"~C7INPUT DATA ACQUISTION INFORMATION~C2"
1040 PRINT:PRINT"~C6PLANT CODE [";R$(N,1);:LINEINPUT"]:~C2";R$:IFR$<>"
"THENR$(N,1)=R$
1050 PRINT:PRINT"~C6NAME [";R$(N,2);:LINEINPUT"]:~C2";R$:IFR$<>""THENR
$(N,2)=R$
1060 PRINT:PRINT"~C6INPUT DEVICE [";R$(N,3);:LINEINPUT"]:~C2";R$:IFR$=
""THEN1090
1070 IFR$="M"ORR$="0"ORR$="1"ORR$="2"ORR$="3"THENR$(N,3)=R$ELSE1060
1080 IFR$="M"THENPRINT:LINEINPUT"~C6ADDRESS:~C2";R$:R$(N,3)=R$(N,3)+R$
1090 PRINT:PRINT"~C6DATA TYPE [";R$(N,4);:LINEINPUT"]:~C2";R$
1100 IFR$=""ANDR$(N,4)="A"THENPRINT:GOTO1130ELSEIFR$=""THEN1190
1110 R$=LEFT$(R$,1):IFR$<>"A"ANDR$<>"B"THEN1090ELSEPRINT:IFR$="B"THEN1
170
1120 R$(N,4)=R$
1130 PRINT"~C6CONVERSION ALGORITHM # [";R$(N,5);:LINEINPUT"]:~C2";R$
```

```
1140 IFR$<>""THENR$(N,5)=R$
1150 PRINT:PRINT"~C6RANGE [";R$(N,6);:LINEINPUT"]:~C2";R$:IFR$<>""THEN
R$(N,6)=R$
1155 PRINT:PRINT"~C6SIGNIFICANT ABSOLUTE CHANGE [";R$(N,9);:LINEINPUT"
]:~C2";R$:IFR$<>""THENR$(N,9)=R$
1160 GOTO1190
1170 LINEINPUT"~C6DATA INVERSION (Y/N):~C2";R$
1180 R$=LEFT$(R$,1):IFR$<>"Y"ANDR$<>"N"THEN1170ELSER$(N,4)=R$
1190 PRINT:PRINT"~C6SCAN RATE [";R$(N,7);:LINEINPUT"]:~C2";R$:IFR$<>""
THENR$(N,7)=R$
1200 PRINT:PRINT"~C6SCAN PRIORITY [";R$(N,8);:LINEINPUT"]:~C2";R$
1210 IFR$<>""THENR$(N,8)=R$
1220 PRINT:PRINT"~C7END OF DA UNIT INPUT":PRINT
1230 PRINT:RETURN
2000 '
2010 ' ---- STATUS - ROUTINE FOR ENTERING EVENT PROCESSOR INFORMATION
2020 '
2025 IFFS=0THEN499
2030 PRINT:PRINT:PRINT"~C7INPUT ANALOG EVENT PROCESSOR INFORMATION~C2"
2040 PRINT:PRINT"~C6PLANT CODE [";S$(N,1);:LINEINPUT"]:~C2";S$:IFS$<>"
"THENS$(N,1)=S$
2045 PRINT:PRINT"~C6EVENT NAME [";S$(N,2);:LINEINPUT"]:~C2";S$:IFS$<>"
"THENS$(N,2)=S$
2050 PRINT:PRINT"~C6EVENT TYPE [";S$(N,3);:LINEINPUT"]:~C2";S$:IFS$<>"
"THENS$(N,3)=S$
2060 PRINT:PRINT"~C6LOWER LIMIT [";S$(N,4);:LINEINPUT"]:~C2";S$:IFS$<>
""THENS$(N,4)=S$
2070 PRINT:PRINT"~C6LOWER HYSTERESIS [";S$(N,5);:LINEINPUT"]%:~C2";S$:
IFS$<>""THENS$(N,5)=S$
2080 PRINT:PRINT"~C6UPPER LIMIT [";S$(N,6);:LINEINPUT"]:~C2";S$:IFS$<>
""THENS$(N,6)=S$
2090 PRINT:PRINT"~C6UPPER HYSTERESIS [";S$(N,7);:LINEINPUT"]%:~C2";S$:
IFS$<>""THENS$(N,7)=S$
2100 PRINT:PRINT"~C7END OF ANALOG EVENT PROCESSOR INPUT":PRINT
2110 PRINT:RETURN
4000 '
4010 ' ---- ALARM - ROUTINE FOR ENTERING ALARM GENERATION INFORMATION
4020 '
4025 IFFA=0THEN499
4030 PRINT:PRINT:PRINT"~C7INPUT ALARM CONDITION INFORMATION~C2"
4040 PRINT:PRINT"~C6ALARM NAME [";A$(N,1);:LINEINPUT"]:~C2";A$:IFA$<>"
"THENA$(N,1)=A$
4045 PRINT:PRINT"~C6OUTPUT CODE [";A$(N,2);:LINEINPUT"]:~C2";A$:IFA$<>
""THENA$(N,2)=A$
4050 PRINT:PRINT"~C6CONDITION ON [";A$(N,3);:LINEINPUT"]:~C2";A$:IFA$<
>""THENA$(N,3)=A$
4060 PRINT:PRINT"~C6CONDITION OFF [";A$(N,4);:LINEINPUT"]:~C2";A$:IFA$
<>""THENA$(N,4)=A$
4070 PRINT:PRINT"~C6PERSISTENCY [";A$(N,5);:LINEINPUT"]:~C2";A$:IFA$<>
""THENA$(N,5)=A$
4080 PRINT:PRINT"~C7END OF ALARM CONDITION INPUT":PRINT
4090 PRINT:RETURN
=C2R0k
```

```
10 '---- OFF-LINE ALARM DATA BASE COMPILER --------
20 '---- VER 2.1
30 '---- G. HOENIG, LUT, FEB 1982
40 '
50 PRINTCHR$(27);"R04"      '--SET PORT BAUD 300
60 CLEAR2000:PRINTCHR$(27);"OA0";CHR$(12)
70 DIM DI(100),V$(9),H(3)
80 DOS"ARYLOAD DI,DI"
90 PRINT"---- FILE DIRECTORY ----":PRINT"FILE NUMBER - NO. OF ELEMENTS
"
100 PRINT:PRINT"DA":FORI=1TO25:GOSUB140:NEXTI
110 PRINT:PRINT"EP":FORI=26TO50:GOSUB140:NEXTI
120 PRINT:PRINT"AG":FORI=51TO75:GOSUB140:NEXTI
130 PRINT:PRINT"ADB":FORI=76TO100:GOSUB140:NEXTI:PRINT:GOTO150
140 IFDI(I)<>0THENPRINTI;"-";DI(I),:RETURNELSERETURN
150 IFX=1THENRETURNELSEPRINT:R=0:INPUT"DA FILE NO.";R:R=INT(R):IFR=0TH
EN1500ELSEIFR>25ORR<1THEN150
151 M=R:GOSUB160:N=DI(R)
152 S=0:PRINT:INPUT"EP FILE NO.";S:S=INT(S):IFS=0THEN1500ELSEIFS>50ORS
<26THEN152
153 M=S:GOSUB160:NS=DI(S)
154 A=0:PRINT:INPUT"AG FILE NO.";A:A=INT(A):IFA=0THEN1500ELSEIFA>75ORA
<51THEN154ELSE156
155 M=A:GOSUB160:NA=DI(A)
156 DB=0:PRINT:INPUT"ADB NO.";DB:DB=INT(DB):IFDB=0THEN1500ELSEIFDB>100
ORDB<76THEN156
157 IFDI(DB)<>0THENPRINT"** WARNING **  ALARM DATA BASE  ADB";DB;" ALR
EADY EXISTS!"
158 PRINT"ARE YOU SURE? ";:LINEINPUT Q$:IFLEFT$(Q$,1)<>"Y"THEN150ELSE1
70
160 IFDI(M)=0THENPRINT"FILE NOT FOUND":ENDELSERETURN
170 DIM R$(N,9)
175 Q$="":INPUT"HARD COPY (Y/N)";Q$:IFQ$="Y"THENPRINTCHR$(27);"O&4"
180 DOS"OPEN 5 R DA"+MID$(STR$(R),2,2)+".DAT"
190 FORI=1TON:FORJ=1TO9:LINEINPUT#5;R$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
200 FORI=1TON:PRINT:FORJ=1TO9:PRINTR$(I,J);"  ";:NEXTJ:NEXTI:PRINT
210 ER=0:FORK=2TON:J=K
220 IFR$(J,1)<R$(J-1,1)THEN230ELSE240
230 FORM=1TO9:V$(M)=R$(J-1,M):R$(J-1,M)=R$(J,M):R$(J,M)=V$(M):NEXTM:J=
J-1:GOTO220
240 NEXTK
250 FORI=2TON:IFR$(I,1)=R$(I-1,1)THENER=ER+1:R$(I-1,1)="*"+R$(I-1,1):N
EXTIELSENEXTI
260 PRINT
```

```
270 PRINT"PLANT CODE","NAME",,"I/P DEV","TYPE","ALG NO.","RANGE","SCAN
","PRIORITY"
280 PRINT:FORI=1TON:FORJ=1TO8:PRINTR$(I,J),:IFJ=2ANDLEN(R$(I,J))<14THE
NPRINT"",
285 NEXTJ:PRINT:PRINTTAB(85);"(";R$(I,9);")"
290 PRINT:NEXTI
300 PRINT:IFER=1THENPRINT"****  1 ERROR  ****"ELSEPRINT"**** ";ER;" ER
RORS  ****"
310 PRINT:IFX=1THENX=0:RETURN
330 '--SORT SCAN RATES
340 FORK=2TON:J=K
350 IFR$(J,7)<R$(J-1,7)THEN360ELSE370
360 FORM=1TO9:V$(M)=R$(J-1,M):R$(J-1,M)=R$(J,M):R$(J,M)=V$(M):NEXTM:J=
J-1:GOTO350
370 NEXTK
380 'X=1:GOSUB260
390 '--SORT PRIOR
400 FORK=2TON:J=K
410 IFVAL(R$(J,8))<VAL(R$(J-1,8))ANDR$(J,7)=R$(J-1,7)THEN420ELSE430
420 FORM=1TO9:V$(M)=R$(J-1,M):R$(J-1,M)=R$(J,M):R$(J,M)=V$(M):NEXTM:J=
J-1:GOTO410
430 NEXTK
431 '--CK SIG CHG
440 X=1:GOSUB260
450 '--DA HEADER
460 DIMRH(8):FORI=1TON:FORJ=1TO4:IFVAL(R$(I,7))=JTHENRH(J+1)=RH(J+1)+1
470 NEXTJ:NEXTI:RH(1)=N
480 '--TYPE AND RANGE
490 FORI=1TON:X$=R$(I,4):IFX$="A"THENR$(I,4)=STR$(1+VAL(R$(I,5)))
500 IFX$="Y"THENR$(I,4)=STR$(1)ELSEIFX$="N"THENR$(I,4)=STR$(0)
510 IFX$<>"N"ANDX$<>"Y"ANDX$<>"A"THENPRINT"DA TYPE ERROR  ";R$(I,4):EO
=EO+1
520 P=INSTR(2,R$(I,6)," "):IF(P=0ORP=LEN(R$(I,6)))ANDX$="A"THENPRINT"R
ANGE ERROR  ";R$(I,6):EO=EO+1:GOTO540
530 R$(I,5)=LEFT$(R$(I,6),P):R$(I,6)=RIGHT$(R$(I,6),LEN(R$(I,6))-P)
540 NEXTI
550 '--PROCESS I/P
560 FORI=1TON:J=VAL(R$(I,3)):IFJ<=0ORJ>4THEN570ELSER$(I,3)=STR$(J-1):N
EXTI:GOTO580
570 EO=EO+1:PRINT"I/P ERROR  ";R$(I,3):NEXTI
580 '--SET DA ADDRESS OFFSET
590 FORI=1TON:R$(I,2)=STR$(I):NEXTI
600 '--PROCESS EP
610 'PRINT:X=1:GOSUB90:X=0:PRINT:INPUT"EP FILE NO.";S:S=INT(S):IFS>500
RS<26THEN510
620 NS=DI(S)
630 DIM S$(NS,8)
640 DOS"OPEN 5 R EP"+MID$(STR$(S),2,2)+".DAT"
650 FORI=1TONS:FORJ=1TO7:LINEINPUT#5;S$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
660 FORI=1TONS:PRINT:FORJ=1TO7:PRINTS$(I,J);"  ";:NEXTJ:NEXTI:PRINT
670 ES=0:FORK=2TONS:J=K
```

```
680 IFS$(J,2)<S$(J-1,2)THEN690ELSE700
690 FORM=1TO7:V$(M)=S$(J-1,M):S$(J-1,M)=S$(J,M):S$(J,M)=V$(M):NEXTM:J=
J-1:GOTO680
700 NEXTK
710 FORK=2TONS:J=K
720 IFS$(J,3)<S$(J-1,3)ANDS$(J,2)=S$(J-1,2)THEN730ELSE740
730 FORM=1TO7:V$(M)=S$(J-1,M):S$(J-1,M)=S$(J,M):S$(J,M)=V$(M):NEXTM:J=
J-1:GOTO720
740 NEXTK
750 FORI=2TONS:IFS$(I,2)=S$(I-1,2)THENES=ES+1:S$(I-1,2)="*"+S$(I-1,2):
NEXTIELSENEXTI
760 FORI=1TONS:M=0:FORJ=1TON:IFS$(I,1)=R$(J,1)THENM=M+1
770 NEXTJ:IFM=0THENS$(I,1)="*"+S$(I,1):ES=ES+1:NEXTIELSENEXTI
780 PRINT
790 PRINT"EVENT CODE","EVENT NAME",,"PLANT CODE","TYPE","L. LIMIT","L.
 HYS.","U. LIMIT","U. HYS."
800 FORI=1TONS
810 PRINT USING "E**##";I.:PRINT"",S$(I,2),:IFLEN(S$(I,2))<14THENPRINT
"",
820 PRINTS$(I,1),:FORJ=3TO7:PRINTS$(I,J),:NEXTJ:PRINT:NEXTI
830 PRINT:IFES=1THENPRINT"****  1 ERROR  ****"ELSEPRINT"**** ";ES;" ER
RORS  ****"
840 PRINT
850 V$(1)="ON":V$(2)="OFF":V$(3)="LO LO":V$(4)="LO":V$(5)="HI":V$(6)="
HI HI":V$(7)="TREND"
860 FORI=1TONS:M=0:FORJ=1TO7:IFS$(I,3)=V$(J)THENS$(I,3)=STR$(J):J=7:M=
1
865 NEXTJ:IFM=0THENEO=EO+1:PRINT"EP TYPE ERROR  ";S$(I,3):NEXTIELSENEX
TI
866 FORI=1TONS:FORJ=4TO7:H(J-4)=VAL(S$(I,J)):NEXTJ:M=0
867 IFH(0)>H(1)THEN868ELSEFORJ=0TO2:IFH(J+1)>H(J)THENNEXTJ:GOTO869ELSE
M=1:NEXTJ:GOTO869
868 FORJ=0TO2:IFH(J+1)<H(J)THENNEXTJELSEM=1:NEXTJ
869 IFM<>0THENEO=EO+1:PRINT"LIMIT ERROR  ";S$(I,1):NEXTIELSENEXTI
870 '--PROCESS AL
880 'PRINT:X=1:GOSUB90:X=0:PRINT:INPUT"AG FILE NO.";A:A=INT(A):IFA>750
RA<51THEN810
890 NA=DI(A)
900 DIM A$(NA+1,7):TN=0:AN=0
910 DOS"OPEN 5 R AG"+MID$(STR$(A),2,2)+".DAT"
920 FORI=1TONA:FORJ=1TO5:LINEINPUT#5;A$(I,J):NEXTJ:NEXTI:DOS"CLOSE 5"
930 FORI=1TONA:PRINT:FORJ=1TO5:PRINTA$(I,J);"  ";:NEXTJ:NEXTI:PRINT
940 X=0:EA=0:FORK=2TONA:J=K
950 IFA$(J,1)<A$(J-1,1)THEN960ELSE970
960 FORM=1TO5:V$(M)=A$(J-1,M):A$(J-1,M)=A$(J,M):A$(J,M)=V$(M):NEXTM:J=
J-1:GOTO950
970 NEXTK
980 FORI=1TONA:IFA$(I,1)=A$(I-1,1)THENEA=EA+1:A$(I-1,1)="*"+A$(I-1,1):
NEXTIELSENEXTI
990 '--BOOLEAN PROCESS
1000 A1$="NOT":A2$="OR":A3$="AND":A4$="XOR"
```

```
1010 FORI=1TONA:PRINT:PRINT"ALARM NAME:     ";A$(I,1);" (";A$(I,2);")"
:FORL=3TO4
1020 DIMAA$(50),B$(50)
1030 J=0:M=0:T$="":X$="":FORK=1TOLEN(A$(I,L))+1:X$=MID$(A$(I,L),K,1)
1040 IFX$="("THENM=M+1:B$(M)="(":GOTO1130
1050 IFX$=")"THENIFT$<>""THENJ=J+1:AA$(J)=T$:T$="":GOSUB1510:GOTO1150E
LSE1150
1060 IFX$<>" "ANDX$<>""THENT$=T$+X$:GOTO1130
1070 IFT$="NOT"THENM=M+1:B$(M)=A1$:T$="":GOTO1130
1080 IFT$="OR"THENM=M+1:B$(M)=A2$:T$="":GOTO1130
1090 IFT$="AND"THENM=M+1:B$(M)=A3$:T$="":GOTO1130
1100 IFT$="XOR"THENM=M+1:B$(M)=A4$:T$="":GOTO1130
1110 IFX$=""THENIFT$<>""THENJ=J+1:AA$(J)=T$:GOSUB1510:GOTO1180ELSE1180
1120 IFX$=" "THENIFT$=""THEN1140ELSEJ=J+1:AA$(J)=T$:T$="":GOSUB1510:GO
TO1140
1130 NEXTK
1140 IFB$(M)=A1$THENJ=J+1:AA$(J)=A1$:M=M-1:GOTO1140ELSE1130
1150 IFB$(M)<>"("ANDB$(M)<>""THENJ=J+1:AA$(J)=B$(M):M=M-1:GOTO1150
1160 IFB$(M)="("THENM=M-1:X$=" ":GOTO1120
1170 IFB$=""THENX$=" ":GOTO1120
1180 IFM>0THENJ=J+1:AA$(J)=B$(M):M=M-1:GOTO1180
1185 IFX=1THENRETURN
1190 IFL=3THENPRINT"CONDITION ON:   ";:A$(I,6)=STR$(J)ELSEPRINT"CONDIT
ION OFF:  ";:A$(I,7)=STR$(J)
1200 TA=TA+J:FORK=1TOJ:IFAA$(K)="("ORAA$(K)=")"THENAA$(K)="#"+AA$(K):E
A=EA+1
1205 PRINTAA$(K);", ";:NEXTK:PRINT
1210 ERASE AA$,B$
1220 NEXTL:PRINT"PERSISTANCY:    ";A$(I,5):NEXTI
1230 PRINT:IFEA=1THENPRINT"****  1 ERROR  ****"ELSEPRINT"**** ";EA;" E
RRORS  ****"
1240 PRINT
1250 '--ASSIGN AG AD OFFSET
1260 FORI=1TONA:A$(I,1)=STR$(N+NS+I):NEXTI
1270 '--ASSIGN DA & EP AD OFFSET TO EP
1280 FORI=1TONS:FORJ=1TON:IFS$(I,1)=R$(J,1)THENS$(I,1)=R$(J,2):J=N
1290 NEXTJ:S$(I,8)=S$(I,2):S$(I,2)=STR$(I-1):NEXTI
1295 TS=0:FORI=1TONS:VS=VAL(S$(I,3)):IFVS>1ANDVS<8THENVS=7ELSEVS=3
1296 S$(I,1)=STR$(VS):TS=VS+TS:NEXTI
1300 '--DATA BASE HEADER
1310 DIMDBZ(11)
1320 DBZ(1)=TA+NA*4+TS+N*6+11:DBZ(2)=N*6:DBZ(3)=RH(2):DBZ(4)=RH(3):DBZ
(5)=RH(4)
1330 DBZ(6)=RH(5):DBZ(7)=11+N*6+1:DBZ(8)=11+N*6+TS+1:DBZ(9)=N:DBZ(10)=
NS:DBZ(11)=NA
1340 '--COMP FAIL
1350 IFER+ES+EA+EO<>0THENPRINT"****  COMPILATION FAILED  ****":PRINT:P
RINT"****  TOTAL ERRORS =";ER+ES+EA+EO;" ****":GOTO1500
1355 PRINT"****  PASS 1 OK  ****"
1360 '--DB BUILDER
1400 DOS"OPEN 5 W *.DOS"
```

```
1410 FORI=1TO11:PRINT#5;DBZ(I);CHR$(13);:NEXTI
1420 FORI=1TON:PRINT#5;R$(I,1);CHR$(13);:FORJ=3TO6:PRINT#5;VAL(R$(I,J)
);CHR$(13);:NEXTJ:PRINT#5;R$(I,9);CHR$(13);:NEXTI
1430 FORI=1TONS:VS=VAL(S$(I,1)):PRINT#5;VAL(S$(I,1));CHR$(13);:PRINT#5
;VAL(S$(I,VS+1))+1;CHR$(13);:FORJ=3TOVS:PRINT#5;VAL(S$(I,J));CHR$(13);
:NEXTJ:NEXTI
1440 A1$="-1":A2$="-2":A3$="-3":A4$="-4"
1450 X=1:FORI=1TONA:PRINT#5;VAL(A$(I,2));CHR$(13);VAL(A$(I,5));CHR$(13
);
1460 FORL=3TO4:GOSUB1020:IFL=3THENPRINT#5;VAL(A$(I,6));CHR$(13);ELSEPR
INT#5;VAL(A$(I,7));CHR$(13);
1470 FORK=1TOJ:FORM=1TONS:IFAA$(K)=S$(M,8)THENAA$(K)=S$(M,2):M=NS
1480 NEXTM:PRINT#5;VAL(AA$(K));CHR$(13);:NEXTK:ERASEAA$,B$:NEXTL:NEXTI
:X=0
1490 DOS"CLOSE 5 ADB"+MID$(STR$(DB),2,3)+".DAT"
1495 DI(DB)=DBZ(1):DOS"ARYSAVE DI,DI":DOS"COMPRESS"
1496 PRINT"****  COMPILATION OK  ****"
1500 PRINTCHR$(27);"0aF";CHR$(27);"0AF":DOS"SUBMIT LOADOFLAD":END
1510 '--EVENT C/K SUB
1530 F=0:BB$="":FORG=1TOLEN(AA$(J)):IFMID$(AA$(J),G,1)="^"THENBB$=BB$+
" "ELSEBB$=BB$+MID$(AA$(J),G,1)
1540 NEXTG:AA$(J)=BB$:IFX=1THENRETURN
1545 FORH=0TONS:IFBB$=S$(H,2)THENF=F+1:H=NS
1550 NEXTH:IFF<1THENAA$(J)="*"+AA$(J):EA=EA+1:F=0
1560 RETURN
=C2R0k
```

```
10 '---- TRANSFER LINK TO 11/03 -------
20 '---- VER 1.0
30 '---- G. HOENIG, LUT FEB 1982
40 '
50 PRINTCHR$(27);"ROC"    '--SET PORT BAUD 300
60 CLEAR2000:PRINTCHR$(27);"DA0";CHR$(12)
70 DIM DI(100),V$(8)
80 DOS"ARYLOAD DI,DI"
85 PRINT"ALARM DATA BASE TRANSFER ROUTINE":PRINT:PRINT:PRINT"THE CHROM
ATICS MUST BE CONNECTED TO THE PDP 11/03."
86 PRINT:PRINT"THE ALARM HANDLING SYSTEM MUST BE INSTALLED AND RUNNING
 BEFORE PROCEEDING."
87 PRINT:PRINT"A LINK ERROR WILL OCCUR IF THE SET UP IS NOT CORRECT, T
HUS ABORTING THE TRANSFER."
90 PRINT:PRINT:PRINT"---- AVAILABLE ALARM DATA BASES ---"
100 PRINT:PRINT"ADB":FORI=76TO100:GOSUB140:NEXTI:PRINT:GOTO150
140 IFDI(I)<>0THENPRINTI;"-";DI(I),:RETURNELSERETURN
150 DB=0:PRINT:INPUT"ADB NO.";DB:DB=INT(DB):IFDB=0THEN465ELSEIFDB>100D
RDB<76THEN150
155 IFDI(DB)=0THENPRINT"ADB DOES NOT EXIST!":PRINT"ENTER 0 TO EXIT":GO
TO150
160 PRINT:LINEINPUT"ARE YOU SURE? ";Q$:IFLEFT$(Q$,1)<>"Y"THEN465
165 DOS"OPEN 5 R ADB"+MID$(STR$(DB),2,2)+".DAT"
200 ' -- ASK FOR OVERLAY
210 ONERRORGOTO1000
220 TIMEOUT375:T=0
230 PRINT#1
240 PRINT#1;"L"
250 GOSUB500:IFN$=""THENPRINT#1;"L":GOTO250
300 ' -- START XFER
310 ONERRORGOTO2000
320 TIMEOUT200
330 IFN$<>"*"THENPRINT#1;"L":GOSUB500:GOTO330
340 PRINT#1;"READY"
350 GOSUB500
360 IFN$<>"0"THENFORI=1TO3:GOSUB500:NEXTI:GOTO330
365 INPUT#5;S:S$=STR$(S):GOSUB600
370 FORI=1TOS
380 GOSUB600:PRINT#1;S$
390 GOSUB500
400 IFN$<>"0"THENPRINT"XFER ERROR":GOTO450
405 LINEINPUT#5;S$
410 NEXTI
450 ' -- CLOSE DOWN
```

```
455 FORI=1TO10:PRINT#1:NEXTI
460 DOS"CLOSE 5"
465 ONERRORGOTO0
470 PRINTCHR$(27);"0AF":DOS"SUBMIT LOADOFLA":END
500 ' -- PROD
510 PRINTCHR$(27);"0BF"
520 T1=0:N$=""
530 LINEINPUT#1;N$
540 PRINTCHR$(27);"0B4"
550 RETURN
600 REM"CK FLOAT"
605 GOSUB700
610 L=LEN(S$)-1
620 P=INSTR(S$,"."):P1=INSTR(S$,"E")
630 GOSUB650:GOSUB700:GOSUB730:RETURN
650 IFP=0ANDP1=0THENS$=S$+".0":RETURN
660 IFP=0THENS$=LEFT$(S$,P1-1)+".0E"+RIGHT$(S$,L-P1):RETURN
670 IFP=1THENS$="0"+S$:RETURN
680 IFP=2THENS$="-0"+LEFT$(S$,L)
690 RETURN
700 REM"CLR SPC"
705 L=LEN(S$)
710 P=INSTR(S$," ")
720 IFP>0THENS$=LEFT$(S$,P-1)+RIGHT$(S$,L-P):GOTO705
725 RETURN
730 X=ASC(S$)
740 IFX<48ORX>57THENIFX<>45THENPRINT"TYPE ERROR IN ELEMENT";I:S$="9999
.9999"
750 RETURN
1000 ' -- ERR LINK
1010 IFERR=25THENT=T+1:IFT>5THENPRINT"LINK ERROR - ABORT";CHR$(7):RESU
ME450
1020 IFERR=25THENRESUME540ELSEONERRORGOTO0
2000 ' -- ERR COMM
2010 IFERR=25THENT1=T1+1:IFT1>5THENPRINT"XFER ERROR - ABORT";CHR$(7):R
ESUME450
2020 IFERR=25THENRESUME530ELSEONERRORGOTO0
=C2R0k
```

```
10 '----- INITIALIZE A DIRECTORY ARRAY -------
20 '
30 '----- ENTER A "0" WHEN FINISHED ----------
40 '
50 DIM DI(100)
60 FORI=1TO100:DI(I)=0:NEXT
70 INPUT"FILE NO.";F:IFF=0THEN100
80 INPUT"NUMBER OF ELEMENTS";N
90 DI(F)=N:GOTO70
100 DOS"ARYSAVE DI,DI"
110 DOS"COMPRESS"
120 END
=C2R0k
```

```
 LOADXFER.SRC FILE LISTING
VER 1.0
G. HOENIG, FEB. 1982, LUT



    NEW
    DOS"LOAD XFER"
    RUN
```

```
LOADOFLA.SRC FILE LISTING
VER 1.0
G. HOENIG, FEB. 1982, LUT



    NEW
    DOS"LOAD OFLAD"
    RUN
```

LOADCONP.SRC FILE LISTING
VER 1.0
G. HOENIG, FEB. 1982, LUT

```
NEW
DOS"LOAD CONP"
RUN
```

APPENDIX E

SOFTWARE DESCRIPTION FOR THE ON-LINE COMPONENT OF THE ALARM HANDLING SYSTEM

APPENDIX E

TABLE OF CONTENTS

E.1.0 ALARM HANDLING SYSTEM OVERVIEW

E.1.1 INTRODUCTION

The purpose of this section is to give an overview of the alarm handling system operator, documentation, software and hardware. The alarm handling system is a stand-alone device intended for process plant applications where there may be a need to improve process alarm data generation and presentation. The device is passive in nature, that is, the system collects and processes plant data, manipulating the data, generating alarm information, and finally displaying the information without performing process control functions. The system acquired process data, manipulates the data, generates alarm and other status information and displays this information to the plant operator.

A combination of microprocessor based equipment is implemented in the alarm handling system. A PDP 11/03 computer forms the basis of the system. The accompanying display package runs on a Chromatics CG-1999 intelligent colour graphics terminal.

E.1.2 THE ON-LINE SYSTEM

Software for the alarm system is comprised of two major sections; 1.) the alarm handling software and 2.) the alarm display software. The alarm handling software written entirely in SWEPSPEED II runs in the PDP 11/03 computer. Alarm display software written principally in Microsoft BASIC runs in the Chromatics graphics computer. An additional software section will be resident in the host computer if present. Described here is a summary of the software organisation of the alarm handling systems functional structure and task inter-

relationships. Details of the individual program tasks are described in subsequent sections. The reader should be familiar wth SWEPSPEED II and Microsoft BASIC before proceeding. An introducton to SWEPSPEED II can be found elsewhere in the documentation.

Since the alarm handling system software is comprised of many tasks running independently in a real-time environment, coordination of tasks requires an overall or global program structure capable of performing housekeeping functions such as inter-task communication, system startups and other program task supervisions. The software communication structure for the entire on-line alarm handling system (AHS) is shown in Fig E.1.1. The alarm system is comprised of 20 SWEPSPEED program tasks resident in the PDP11/03, or Microsoft BASIC display task in the Chromatics and interface tasks in the host computer if present. All program sections must be installed and running for the alarm handling system to function correctly. Communication tasks between the computers are driven by software drivers which detect the absence of a link line. Software resident in the PDP11/03 computer constitutes the core of the alarm handling system. Program tasks coordinate all the functions of the system. The SWEPSPEED tasks can be classified according to their functions as follows:

1. Supervision tasks
2. Link drivers
3. Device drivers
4. Alarm handling tasks
5. Auxillary tasks

BASIC programs located in the Chromatics are used to implement a variety of colour VDU based alrm displays. Note

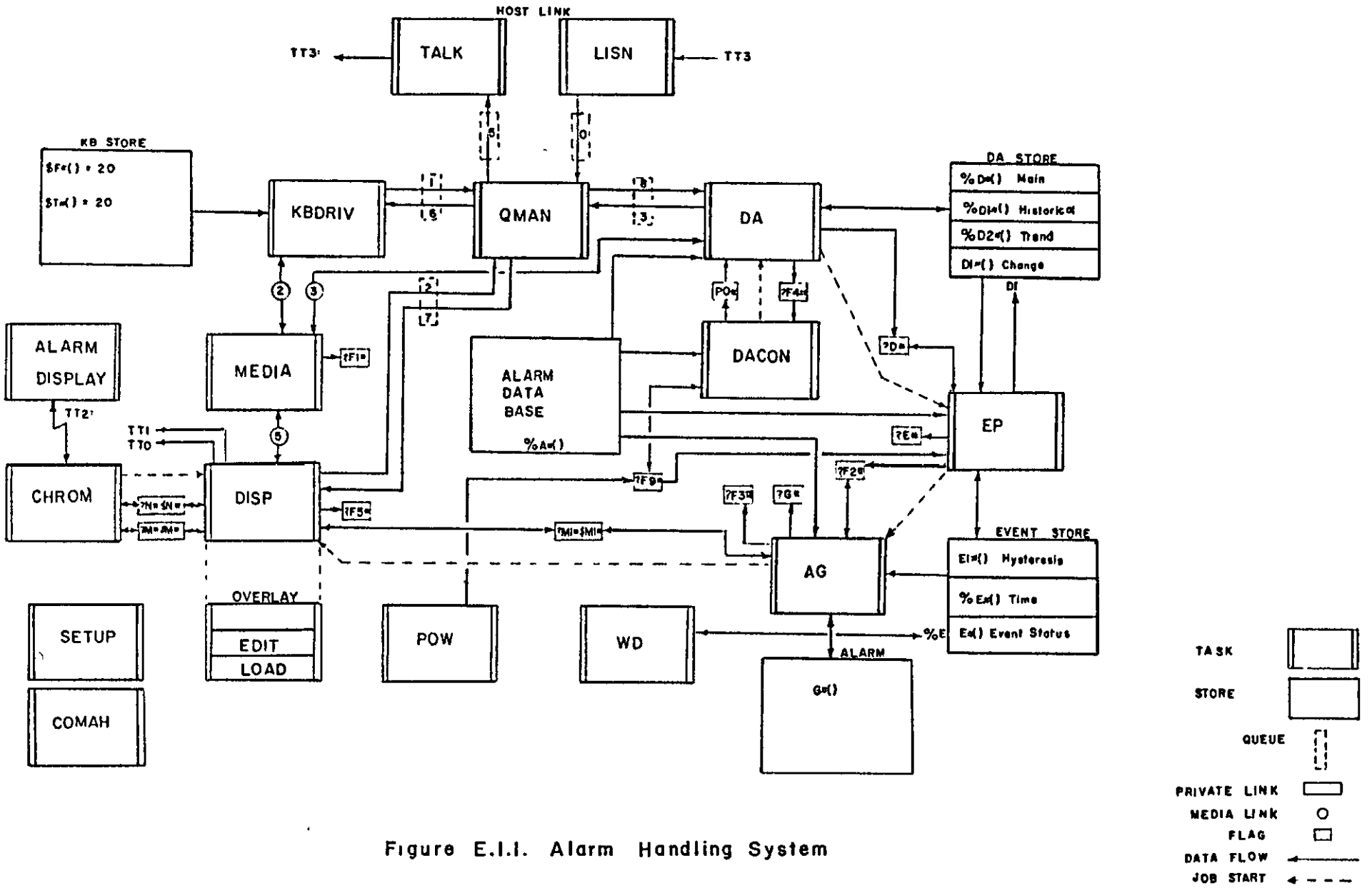


Figure E.I.I. Alarm Handling System

that this document refers to the on-line duties of the Chromatics. Off-line alarm data base building and transfer routines are executed only when the Chromatics is in the off-line mode. These off-line programs are discussed elsewhere.

When the alarm handling system is used with a host computer, program tasks resident in the host are used for:

1. Link with alarm handling computer.

2. Data acquisition routines for returning process data from the host data base.

3. Other application specific functions.

The language used for the host task is dependent on the application.

The system overhead tasks in the alarm handling computer are as follows:

Task Name	Section	Function
Q-MAN	E.2.0	Supervise queue communication system.
POW	E.3.0	Coordinate alarm system startups.
SETUP	E.4.0	Contains alarm handling system array dimensions. Must be adjusted to meet specifications of alarm data base.

COMAH	E.5.0	Engineering command task, allows system manager to evoke data base editor, restart or stop the system, etc.
WD	E.6.0	Controls the system hardware monitor watchdog and other time related overhead functions.

The inter-computer link drivers are as follow:

Task Name	Section	Function
LISTEN	E.7.2	Retrieve data packets from the host computer link and place them in the system queue.
TALK	E.7.3	Send data packets as obtained from the system queue down the host computer link.
CHROM	E.7.4	Manager data packet swap routine with the Chromatics computer. Used as a interface between the alarm handling DISPLAY tasks and the Chromatics display package.

The Device Driver tasks in the alarm handling computer are as follows:

Task Name	Section	Function
KBDRIV	E.8.0	Software task which supervises all special operator keyboard functions.
MEDIA	E.9.0	Normalises and supervises all I/O through the Media Plant Interface hardware.
DISPLAY	E.14.0	Although not specifically a device driver this task coordinates data flow out to peripheral display devices.

The software tasks which perform the alarm handling functions are as follows: These tasks all use the alarm data base %A#() as reference.

Task Name	Section	Function
DA	E.10.0	Data Acquisition supervision. Initiates data acquisition, converts process data into engineering values.
DACON	E.11.0	Controls the data acquisition sampling rates.
EP	E.12.0	Event Processor. Examines the data

		retrieved and processed by the DA tasks, generating a binary event status image based upon event definitions in the alarm data base.
AG	E.13.0	Alarm Generator. Examines the event status image and based upon Boolean expressions coded in the alarm data base, generate alarm output codes.
DISPLAY	E.14.0	Alarm output codes are received from the AG task and passed on to the appropriate display output device. This task also has access to the system queue.

Due to memory space restrictions several program tasks are overlayed into job slot 2 where the DISPLAY tasks normally resides. These tasks are service routines required for on-line data base editing and loading. Alarm handling system is automatically stopped whenever tasks are overlayed over the DISPLAY tasks. Overlayed tasks are stored on the magnetic tape cassette which should be located in drive DDO:. Any time an overlay task is executed the system message must restart the system. The overlay tasks are as follows:

Task Name	Section	Function
EDIT	E.15.2	Evoked through COMAH. Allows the system manager to make simple changes to the alarm data base currently residing in the alarm handling system.
LOAD	E.15.3	Evoked by the off-line Chromatics program XFER via the SWEPSPEED task DISPLAY. This task supervises the transfer of an alarm data base from the Chromatics computer when in the off-line mode.
DISPLAY	E.14.0	See previous description. This task is also an overlay since both of the above tasks overlay into the DISPLAY task job slot. When the above overlays are complete, the DISPLAY task overlays back into its original job slot.

Alarm handling computer resident tasks are described in brief here in order to give the reader an idea of the organization of the sysem and the inter-relationship between program tasks.

E.1.3 THE DATA BASE

The alarm handling system is a generalised device. The basic alarm system is not capable of performing any functions without first being programmed. The alarm handling system may be thought of as an operating system ready to be programmed for a specific user application. The alarm data base is the 'program' which defines how and what duties, the alarm system will perform. In the case of the prototype system. this data base is constructed by the user in an off-line development computer. Application specific information concerning data acquisition alarm generation and display is condensed by the off-line computer into a compact coding. Compression of the application data into this data base minimizes the amount of memory space required by the on-line computer to store the alarm system definition. Additionally, the data base is organised in such a manner as to maximize the speed of execution of the data base program.

When the on-line alarm handling system is running, the data base is constantly referenced by the tasks that comprise the system. The data base remains unchanged by the on-line system since any modifications the data base would result in an alteration of the alarm system operator.

Following is a brief summary of the system definitions coded in the alarm data base:

Overhead Information: Sizes required for data base, arrays, lists, etc.

Data Acquisition: Plant addresses, range, conversion, data type and scan rates.

Event Definitions: Type of event and parameters.

Alarm Definitions: Coded expressions describing combinations of events required for an alarm.

Display Data: Alarm output codes, etc.

More details of the data base structure are described in Section E.17.0. The off-line documentation gives a detailed description of the information stored in the alarm data base and how the data base is built.

E.1.4 THE SOFTWARE LANGUAGE

SWEPSPEED II was found to be a convenient language for the Alarm Handling software. The real-time multi-tasking capablities of SWEPSPEED are suitable to build the system from a collection of well defined and structured program tasks. Although SWEPSPEED itself is not particularly a structured language, care has been taken to insure that all program tasks are uniformally organized and formatted. This approach for example results in consistent allocations of program line numbers. For example, all program tasks contains a 'Setup' module located at line 20, program queue communications are at lines 900 and 950, and so forth. The user will find that a clear understanding of the program organization of any one task is directly applicable to any other program task. Details of program organization convention are shown in Section E.1.5. In addition care has been taken to ensure that in general variables in one task will have the same or similar function in other tasks.

E.1.5 PROGRAM TASK SOFTWARE ORGANISATION

Program Line Number	Typical Function
10	Task title with version number.
20	Setup - Variables assigned here are dependent upon the location of supporting tasks and upon the location of the task within the system's communicaton structure. Busy flags and other housekeeping duties relating to globally interacting software functions are also found in this section.
100	Run - This represents the starting point of the main body of the program. Generally, this section is used as a program control module which supervises function within the task via GOSUB commands. Branching to subroutines adds structure to the program making it easier to follow and fault find a program. The Run section also includes all the task shutdown housekeeping functions.
200	Function Selection - This section generally is used for decoding task input commands and selects program routines as dictated by the task command messages.
300+	Flexible and dependent on task.

800 Errors - This program line is always reserved for error trapping routines. Initial error vectoring is performed in the Setup section.

900 INQ - The INQ subroutines are exclusively assigned to lines 900-950. Queue assignments are made in the Setup section.

950 OUTQ - The OUTQ subroutines are exclusively assigned to lines 950-970. Any program task requiring access to the alarm handling queue communication system must have either or both the INQ and OUTQ routines. These routines are identical in all tasks.

Local variable assignments are usually consistent as illustrated by the first example for inter-task communication shown in Figure E.1.2a. In the case of the inter-task queue structure all tasks requiring access to the queue use identical service subroutine software. Other variables which do not have such globally related functions also follow a similar convention. For example, the list of variables shown in Figure E.1.2b generally perform the same functions throughout the alarm system. Global variable assignments are dependent upon the task to which they pertain. Figure E.1.2c illustrates some of these assignments.

Figure E.1.2a

$N = Incoming data packets.

$M = Outgoing data packets.

Q1 = In queue number.

Q2 = Out queue number.

Q = Queue Manager job slot location.

Figure E.1.2b

I = Index or array pointer.

D, D1, D2, etc. = Alarm data base pointers.

F = Function number.

V = Measured process variable value.

L = List location or pointer.

Figure E.1.2c

%A#() = Alarm data base.

?F1#, ?F2#, etc. = Busy flag for an individual task.

%D#(), D#(), etc. = Data acquisition data store.

%E#(), E#(), etc. = Event processor data store.

G#() = Alarm generation data store.

Examples of a typical variable assignment conventions.

Figure E.1.2

E.2.0 THE QUEUE MANAGER

E.2.1 INTRODUCTION

Real time multi-tasking software systems have an inherent difficulty with intertask communication. Since tasks are being executed at differing priority levels and require varing execution times, synchronous communication between tasks can significantly decrease the response time of the entire software system. When a task requires intertask communication, both tasks must wait for each other to complete the necessary handshaking protocol. The 'waiting' process can consume large amounts of processor time and hold up the execution of other tasks. Ideally tasks in a multitasking environment should be able to communicate with other tasks at any time as required.

A common method for implementing such a communication structure is a system queue. As tasks require intertask communication, output messages from tasks are stored up in a queue or stack until the receiving task has time to deal with the incoming message. The sending task is not held up waiting for the receiving task to accept the message. The receiving task can retrieve the message packet at a convenient time. With all tasks communicating via a queue structure the system is not held up by intertask communication.

Each task requiring communication will have an output queue and an input queue. Messages are transferred from output queues to input queues. The queue system is supervised by a task called the Queue Manager. This task examines all queues which contain output messages. If any messages are in the output queues of any task, the Q Manager examines a data header contained within the message to determine the destination queue and makes the transfer.

The header is stripped off and the message is placed in the appropriate input queue corresponding to the receiving task.

An added feature of this system is that any task has access to any other task which contains input/output (I/O) queues. By placing the appropriate header code on the message, a message can be transferred by the Q Manager to any task with queueing facilities.

A queue (Q) is simply a means of emulating a cyclic file which allows data packets to be entered in sequential order and removed in a first in, first out (FIFO) order. An input pointer I# is used to indicate the next available location in the Q. Similarily an output pointer o# is used to indicate the last message location in the Q. The Q is empty when O# and I# are equal. An array is used for the Q. The pointers are incremented to the maximum number of available array elements and then reset to the beginning of the array to start over again, i.e. a cyclic file. As messages are entered into the Q the I# pointer is incremented accordingly. If the next available element (I#+1) is full, that is the O# pointer is pointing to the same location, the Q is full. No further entries may be made until the Q is serviced by the Q Manager.

The Q Manager removes messages from the Q by incrementing the output pointer O# and removing the message packet, until the pointers O# and I# are equal. The Q is now full.

The pointers are 'rotated' around the Q as shown in Fig. E.2.1 making the Q appear continuous. The size of the Q or rather array defines how many message packets can be backlogged before the Q is full. Tasks which intermittently produce large amounts of data for slower tasks are ideal candidates for such a communication structure.

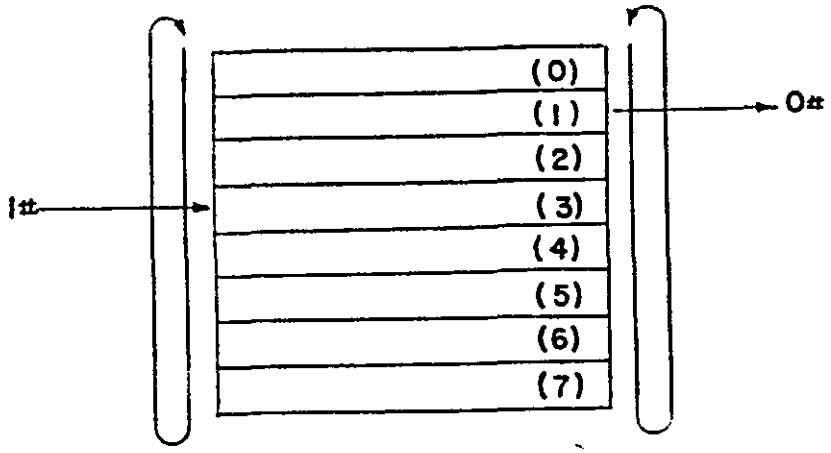


Figure E.2.1 Queue array structure and pointers

E.2.2 OPERATION SUMMARY

This section describes in brief the operation of the Queue Manager. More deatils are given in the subsequent sections. The Queue Manager is always accessed via other tasks in the system so its operation is transparent to the operation of the alarm handling system and the operator.

1) The Q Manager software must be loaded into a SWEPSPEED job slot. This job slot must be assigned a higher priority than any job requiring queue servicing.

2) There are 10 queues available, 5 input types and 5 output types, comprised of reserved global array variables S1#, I#(10), O#(10), and $Q#(70)=20.

3) Each job requiring queue servicing must contain the appropriate IN queue and/or OUT queue software routines.

4) Message packets which are to be sent by a task to another task must contain a data header containing the code

of the destination queue. (Refer to Section E6.0).

5) The calling job places its outgoing message packets in its OUT Q.

6) The calling job must start the queue manager using interactive statements (included in Q software routines).

7) The receiving job is started by the Q Manager if necessary. The receiving job then removes the data packet which has been stripped of the header from its IN Q.

8) The reserved global variables I#() and O#() should be cleared at system startup. The queue manager task should also be the first job started at power up.

E.2.3 COMMUNICATION STRUCTURE AND PROTOCOL

Generally the operation of the Q Manager is transparent to the operation of the system, thus the importance of understanding the Q Manager's operation is non-essential. However, when the user wishes to add or modify jobs which require intertask communication, the system designer should be aware of the functions and operation of the Q Manager. This is necessary to avoid possible conflict with other jobs using the Q system.

The Q Manager transfers message packets from one job's OUT Q to another job's IN Q.

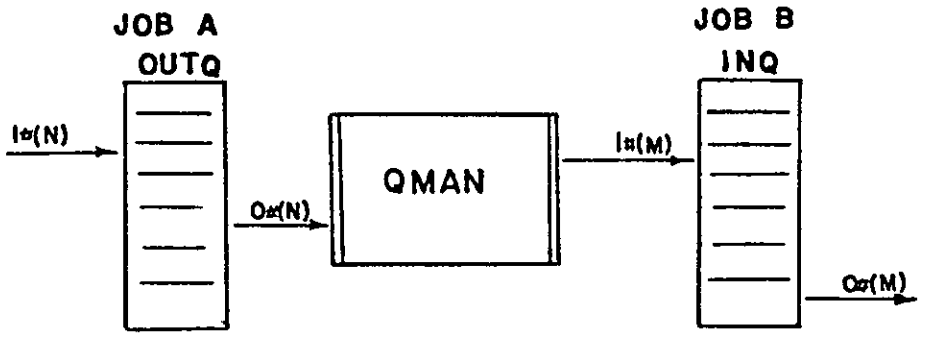


Figure E.2.2 Q Manager Transfer Task

The above figure illustrates the typical information flow through the queue system. Job A is sending message packets to job B via job A's OUT Q through the Q Manager and on to job B's IN Q. The header placed on the message packet by job A indentifies the destination Q in which the message is to be inserted.

E.2.4 JOB PRIORITY

The only restriction on job priority assignment made to the Q Manager is that any task requiring queue servicing must have a job priority assignment lower than that of the Q Manager. If a calling job has a higher priority than the Q Manager, it may be possible that Q pointers are corrupted or confused.

E.2.5 ERRORS

Error messages are generated when any sub-queue within

the system becomes full. The error message

Q-n WAITING n= Q number

is generated by the Q Manager whenever the Q Manager is waiting for space in a task IN Q. In other words queue n is full.

OUT Q routines located in the sending tasks can generate a similar error message. For example;

KB Q WAITING

DISP Q WAITING

These mesaages are generated whenever the corresponding OUT Q is full and waiting service from the Q Manager.

IN Q routines do not generate error messages. If WAITING error messages are persistent, the system manager should consider reassignment of job slot priorities and/or increase the sub-queue size (Sl#).

In the event that the Q Manager detects an invalid header in a message packet, the message packet is dumped and no further action is taken on the packet. No error message is generated.

E.2.6 SOFTWARE FUNCTIONAL DESCRIPTION

As previously described the Q Manager transfers message packets for OUT Q's to IN Q's to and from various tasks. In order to accomplish this function without intertask conflict, the tasks subscribing to the Q system also must contain certain software routines to service their individual Q's. A functional sketch of the system is

illustrated in Figure E.2.3.

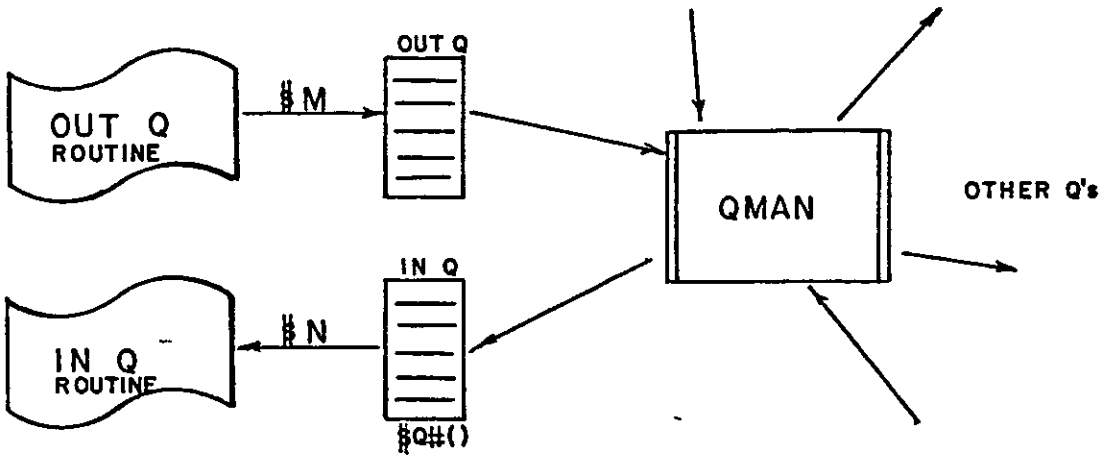


Figure E.2.3 Q Service Routine Information Flow

The OUT Q and IN Q routines are the same for each individual task and are described in the next section. Variables which identify Q numbers assigned to the task are specified in the 'Setup' section at the beginning of the task requiring queue services. These variables are used by the OUT Q and IN Q routines.

Q1 = output Q number [5-9]
Q2 = input Q number [0-4]
Q = Q Manager job slot [7]

The Q is comprised of a string array so therefore all message contents must be in string form not exceeding 20 characters. By convention

$N = Input data
$M = Output data

When outputing a message to another task in the Q system, the message text must contain a 'header' to identify the IN Q to which the message packet is directed. The header by convention consists of the first two characters of the message text. The characters are the string

representation of the IN Q number to which the message is directed. Queue number assignments are preselected in the Q Manager software and are summarized below:

Q Number	Q Type	Task	Job Slot
00	OUT Q	LISN	12
01	OUT Q	KBDRIV	4
02	OUT Q	DISP	2
03	OUT Q	DA	8
04	OUT Q	PCPDAT	17
05	IN Q	TALK	3
06	IN Q	KBDRIV	4
07	IN Q	DISP	2
08	IN Q	DA	8
09	IN Q	PCPDAT	17

By example, if the Keyboard Driver KBDRIV is required to send a message to the DISPlay task, the message outputed through OUT Q 01 would read:

```
$M = 07MESSAGE
```

Once the keyboard driver has defined $M as above, the OUT Q routine within the keyboard driver is called. The routine places the message in the next available queue location and requests the Q Manager to start.

The Q Manager detects the presence of a message in the OUT Q 01 by checking the I#(1) and O#(1) queue pointers. The message is removed form the queue. The Q Manager examines the first two characters of the string and decodes which IN Q the message, stripped of the header, should be placed. The Q Manager having sent the message to IN Q 07, starts the task, if necessary, which contains the specified IN Q. In this case job 4 is started.

Finally, the receiving task must check its own IN Q occasionally to see if there are any entries. This is accomplished by calling the standard IN Q routine which is the same for all jobs nevertheless unique due to the Setup variables. The message text is returned to the job through the variable $N which should now read

$N = MESSAGE

Summarizing, the sending job places outgoing messages in $M with a header identifying the destination Q number. The OUT Q routine is called. The message is placed in the job's OUT Q. The Q Manager transfers the message to the correct IN Q by examining and stripping off the header. Finally, the receiving task, after checking its IN Q by calling the standard IN Q routine removes messages from the Q and are available in the local variable $N.

In the prototype alarm handling system, the IN Q assignment numbers contained in the message packet header is complicated by the fact that the alarm handling system is linked to the PDP 11/34 host computer. As described elsewhere, the PDP 11/34 contains the PCP software package which is also comprised of a large number of separate tasks in the same manner as the alarm handling system. Figuratively, these tasks also contain IN and OUT Q's. The structure and operation of this system is described elsewhere in the PDP 11/34 Link documentation. Nevertheless, IN Q number assignments in the PDP 11/34 start at 10 as follows:

Q Number	Q Type	Task
10	IN Q	GETDAT
11	IN Q	OCP
12	IN Q	PCPMC

When the Q Manager encounters any of the above headers, the header is not stripped off. The entire message packet is placed in the Link task TALK's IN Q. The TALK task send the complete message packet to the 11/34 for further processing. Data returning from the 11/34 does not have any labeling difficulties since all destinations are within the Alarm Handling System.

E.2.6.1 The Queue

The system queue and pointers are comprised of the following reserved global variables:

$Q(70)=20	Main Q
I#(10)	Input pointers on sub-queues
O#(10)	Output pointers on sub-queues
S1#	Sub-queue size defined in Q Manager Setup section

The main Q is a string array with a maximum of 20 characters per element. The main Q is divided into 10 sub-queues as follows:

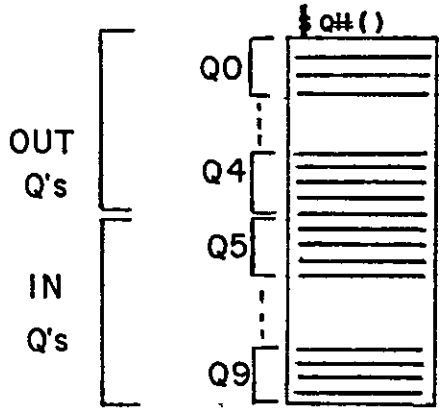


Figure E.2.4 Main Q with sub-queues

The Q input/output pointer are located in the reserved global array variables I#(n) and O#(n) where the subsript indicates the Q number 0 - 9. The location is stored in a relative form. i.e., the first location in the sub-queue is zero. The absolute location in the main queue is calculated a follows:

$$Li = I\#(Q1)+S1\#*Q1 \qquad (1)$$
$$Lo = O\#(Q2)+S1\#*Q2 \qquad (2)$$

In the above equations

L = absolute location in the main queue.
O#(Q2) or I#(Q1) = relative sub-queue location.
S1# = the sub-queue size (currently set at 7).
Q1 = out sub-queue number.
Q2 = in sub-queue number.

The Q-Manager has preassigned OUT queues as Q0 - Q4. Similarly the IN queues are Q5 - Q9. This means that the Q Manager will transfer from the Out Queue to the In Queues where OUT queues are interpreted as 'out' from sending tasks and IN queues as 'in' to the receiving tasks. Any task requiring two way communication via the queue system must contain both a IN and an OUT queue.

The input/output pointers increment around the queue array by using the MOD function. This function is used to evaluate the modulus of two integer expressions. The modulus is defined as the remainder after dividing one number by another. In this way the pointers are always incremented yet in reality are 'rotated' around a queue array. For a queue size of three the pointers are rotated as follows:
0,1,2,0,1,2,0... and so forth, where the modulus is 2.

The input and output pointers I#(q) and O#(q) store these relative values.

Refering to the flowcharts in the Appendix, these two procedures involved in queue servicing, data insertion and data removal. Eack task inserting data into a queue must use an insertion routine called OUTQ. While each task extracting data from a queue must use the complementing removal routine called INQ. The queue manager contains both these routines with some additional software sorting functions.

E.2.6.2 QMAN

The Q Manager performs both INQ and OUTQ functions using almost identical software. For information relating to the insertion and removal of data from the Q refer to the INQ and OUTQ sections.

The Q Manager examines all Output Q's (0-4) pointers I#(q) and O#(q) to determine whether there are any entries in an output queue which require servicing. If an entry is found, message packets are removed from the queue. The header is examined. If the header value is valid the message packet is placed in the corresponding input queue. Next the Q Manager locates the job slot in which the input queue is located. These assignments are located in the array J() and are allocated in the Setup section of the Q Manager. A START command is issued for the corresponding job slot.

In the event that a destination input queue is full the Q Manager generates an error message, waits 2 ticks and tries again as many times as necessary. If a message packet header does not make sense then the packet is dumped. Any header with a value of 10 or more retains its header and is

placed in the PDP11/34 link task queue.

When a scan is complete, if a message packet had been serviced on the scan, a further scan is initiated until all output queues are emptied.

E.2.6.3 INQ

The INQ routine must be present in a task which has an input Q. Refer to Appendix for listing. The routine performs the following funcitons:

1) Examine I#() and O#() pointers for the queue in question.

2) If I#(Q2)=O#(Q2) then the queue is empty, no action is taken and program control is returned to the main task.

3) If I#(Q2)>O#(Q2) then queue entires are present. The data packet located in $Q#(Lo) is placed in $N. O#(Q2) is incremented to the next queue location, and program control is returned to the main task.

By convention Q2 is used to assign the input queue number and must be specified in the task Setup section along with the job slot location of the Q Manager; Q.

E.2.6.4 OUTQ

The OUTQ routine must be present in a task which has an output Q. Refer to section E.20. for listing. The routine performs the following functions:

1) Examine I#(Q1) and O#(Q1) pointers for the queue in question.

2) If I#(Q1)+1=O#(Q1) then the Q is full. An error message is generated, the routine waits 2 ticks, and tries again.

3) The contents of $M is loaded into $Q#(Li) and I#(Q1) is incremented. The Q Manager is started if necessary. Program control is returned to the main task.

By convention Q1 is used to assign the output queue number and must be specified in the task Setup section along with the job slot location of the Q Manager; Q. $M must contain the header identification in the first two characters of the text. The header must be present before loading into the queue.

E.3.0 POWERUP

E.3.1 INTRODUCTION

The POWerup task POW is an alarm handling system housekeeping task. Initialisation of the alarm handling system is performed by this task. When the software system is first booted into the computer various aspects of the software must be initialised for correct operation. The POW task also executes functions which will restart the alarm handling system. In this mode the task is started by the alarm handling command task COMAH.

E.3.2 OPERATION SUMMARY

As described above in normal operation the POW task is automatically started when the SWEPSPEED alarm handling system software is first booted or restarted. In the event that the user requires to re-initialise or 'clear' the alarm handling sytem, the POW task can be started via the command task COMAH.

The only interaction that the user has with the task is that of entering the date and time as prompted by the program on the console terminal.

No error messages are generated by the task, however if the user miss-keys an entry, the program will detect the syntax error when trying to set the time or date. If this should occur, the task is restarted after generating a SWEPSPEED error message.

E.3.3 SOFTWARE DESCRIPTION

The POWerup task is a small and concise program requiring little explanation. Refer to the flow chart and listing for clarifications.

The task performs the following functions:

1) Stops all job slots which are not idle.

2) Permits user to insert real time and date into system.

3) Clears all temporary storage arrays which may cause confusion during a system restart.

4) Initialises the time of occurrence event processor storage array %E#(). All elements are set to the minimum or 'datum' time value -10E6.

5) Connects the watchdog task WD to the system clock.

6) Appropriate job slots are connected to the system clock or started to get the system going.

E.4.0 SETUP

E.4.1 INTRODUCTION

The SETUP task is an initialisation module for the alarm handling system. The task performs no operational functions. Principal data array sizes are dimensioned here. SWEPSPEED is a compiled language therefore itmes such as array sizes can not be dynamically adjusted. The user of the alarm system may need to redimension key arrays to suit the requirements of a particular Alarm Data Base. All arrays which may require redimensioning are included in the SETUP task. All these arrays are global types, therefore redimensioning requires a knowledge of the SWEPSPEED utilities as follows:

$GLO — Used to condense the global table after reducing array sizes. (SQUEEZE)

$INS — Used to re-install overlay files since a new global table will be in use. The EDIT, LOAD, and DISPlay overlays must be installed into job slot 2 after a redimensioning.

$MON — Used to inspect the contents of the Alarm Data Base header.

E.4.2 SOFTWARE DESCRIPTION

Presented here is a list of the global arrays found in the SETUP task and how to calculate the array dimensions.

%A#() — The Alarm Data Base. Dimension to at least the size of the Alarm Data Base as given in %A#(1).

%D#() Data Acquisition current measured value array store. Dimension to the number of data acquisition units as given in %A#(2).

%D1#() Data Acquisition historical measured value array store. Dimension to the number of data acquisition units as given in %A#(2).

%D2#() Data Acquisition trend data array store. Dimension to the number of data acquisition units as given in %A#(2).

D1#() Data Acquisition change data stored in bit form (see BIT function). Dimension to the number of data acquisition units as given in %A#(2) divided by 16, the integer bit size.

%E#() Event Processor 'time of event' store. Dimension to the number of event definitions as given in %A#(10).

E#() Event Processor event status store. Data is stored bit wise (see BIT function). The dimension is calculated by dividing %A#(10), the number of event definitions, by 16, the integer bit size.

E1#() Event Processor measured data hysteresis store. Dimension to the number of event definitions as given in %A#(10).

G#() Alarm Generator output alarm status store. Data is stored in bit wise format (see BIT function). The dimension is the number of alarm definitions as in %A#(11) divided by 16, the integer bit size.

NOTE: Arrays used for bit wise formats are dimensioned by

calculating the total number of bits required. There are 16 bits in each integer represented by the array starting with array subscript 0. The calculation is as follows:

Dimension = INT((number of bits required)/16) MIN 1

Note that since location 0 is used the result of the equation is truncated to integers only.

E.5.0 COMMAND TASK - COMAH

E.5.1 INTRODUCTION

The COMAH task is intended for use by the alarm handling system manager to evoke engineering functions within the system. The principal uses of the COMAH commands are to enter the alarm data base EDIT mode and system startup and shutdown. The task is not intended for use by the operator.

E.5.2 OPERATION SUMMARY

To start the task the system manager must use the SWEPSPEED utility $ACT14 on the console terminal TT0: thus activating job slot 14 where the COMAH task is located. Once in COMAH the prompt '##' will indicate that the task is ready for input. All inputs consist of up to 2 character strings followed by a carriage return. The avaiable functions are as follows:

ED Enter EDIT mode. Overlay storage tape must be in DD0:

TI Print time and date to console.

RE Restart alarm handling system from scratch.

ST Stop the alarm handling system except for the watchdog task WD.

RU Run or 'warm start' the alarm handling system. Useful during fault finding after using the ST command.

X Exit COMAH task.

An error message may be encountered when entering the EDIT mode if the overlay tape is not inserted in drive DD0:.

E.5.3 SOFTWARE DESCRIPTION

The COMAH task is simple and concise. It provides a means of command input and subsequent branching to the appropriate subroutine. For details of program structure refer to flow charts and listings.

E.6.0 WATCH DOG

E.6.1 INTRODUCTION

The Watch Dog task WD drives a system monitor card in the Media Plant Interface hardware. The task also performs some alarm handling system housekeeping functions, the most important of these being the updating of the time of event records in the event processor array %E#().

E.6.2 OPERATION SUMMARY

In normal operation the functioning of the Watch Dog task is transparent to the operator. The task is set to run once each second with a top priority job slot assignment. The POWerup task contains a CONNECT statement for the Watch Dog task so the Watch Dog begins running only after the POWerup task is execute. The Media Interface hardware requires this task to run at least once each second so that the system monitor card does not think that the computer has failed.

E.6.3 SOFTWARE DESCRIPTION

The WD task is a short and concise task requiring little explanation. Refer to the flow chart and listing for further details.

When the task runs a specific bit pattern is sent to the Media system monitor to inform the plant interface that the computer is operational. The current real time in the alarm handling system is then examined to see if the time is 00:00:00, and if so, subtract 18280 seconds (1 day) from each event time in the event processor time of occurance

array %E#(). The minimum time or 'datum' time value is -10E6 (5 days). After completion the WD task remains idle until the next second at which time the execution is repeated.

E.7.0 THE COMMUNICATION LINK TASKS

E.7.1 INTRODUCTION

The communication link tasks supervise software communications with peripheral computer systems. These link tasks are TALK, LISN, and CHROM. The link tasks must have there own job slot allocations since in SWEPSPEED once an input statement is executed the program stops until the input request is satisfied. If inter-computer communications were incorporated into other tasks, the system would not be able to function properly.

Job slot priority assignments must be arranged such that any incoming inter-computer link tasks, LISN and CHROM, are as high or higher than other tasks. This is to insure that incoming data packets are not lost. Links with heavy traffic such as the host computer link should have very high job slot priority.

Described in this section are details of the three link tasks in the alarm handling system.

E.7.2 TALK

The TALK task is a unidirectional communication link with the host process control computer. The TALK task sends ASCII data packets down the serial link line TT3:. To avoid conflict with the LISN task which also uses this line, TT3: has been set to no-echo via the SWEPSPEED system generation. The TALK task prints down line data packets comprised of text strings as retrieved from the alarm handling system queue system. The data packets are sent verbatim without any additional termination characters added to the end or

the string contents.

E.7.2.1 Operation Summary

The TALK task operation is transparent to the operator. Since its only means of retrieving data packets is via the system queue, the task is automatically started by the queue manager when necessay. No program error messages are generated. When the alarm handling system is restarted the local print buffer @T3: may still be operned for output. In this case the program notes the error, rectifies the situation and restarts.

E.7.2.2 Software Description

Little explanation of the flowchart and listing is required. The task uses the standard INQ subroutine for data packet inputs.

E.7.3 LISN

The LISN task is a unidirectional software communication link with the host process control computer. This task is intended to complement the TALK task above. The LISN task retrieves ASCII data packets from the serial link line TT3:. The task uses the same serial line as the TALK task so several precautions must be noted as explaned above. In general the LISN task retrieves data packets from the serial line and passes them verbatim to the alarm handling system queue system. The LISN task utilises the standard input statements so in fact a terminating carriage return is required to input data packets. This means that all incoming data must not contain a carriage return within a data packet. The contents of the data packet must conform with the format as required by the queue manager and the

destination task.

The task is assigned a high priority so that incoming data is not lost.

E.7.3.1 Operation Summary

The LISN task is automatically started during system powerup. If a host computer is not present consult the system manager to remove this automatic startup which is performed in the POW task. This is important to avoid difficulties with subsequent alarm system restarts. If the LISN task has been activated, the program with remain waiting for an input. In this condition the POW task can not stop the job. If this should occur the system manager must use the SWEPSPEED utility $RES to clear and restart the alarm handling system from scratch.

In some situations when the alarm system is restarted the input buffer @L3: may still be opened for input. In this case the program detects this and rectifies the difficulty.

E.7.3.2 Software Description

Little explanation of the flowchart and listing is required. The task uses the standard OUTQ subroutine for inserting data packets into the alarm handling system queue.

E.7.4 CHROM

The CHROM task is a bidirectional software communication task which links the Alarm Display Package resident in the Chromatics display computer with the alarm handling system DISPlay task. The inter-computer link is

comprised of a constant data packet swapping procedure down the serial link line TT2: with the Chromatics. Data packets consist of coded ASCII text strings. The Chromatics Alarm Display Package sends data packets to the DISPlay task headed with a funciton code initiating further functions in the DISPlay task. The DISPlay task sends display commands to the Chromatics as appropriate. If no data packet is sent during a data packet swap an empty data packet is sent. This constant swapping is used to detect the health of the Alarm Display Package. To avoid echoing data packets the TT2: serial line is set to no echo in the SWEPSPEED system generation.

E.7.4.1 Operation Summary

The operation of the CHROM task is automatic once started by the DISPlay task. The CHROM task will not function correctly if the complementing link routine in the Alarm Display Package in the Chromatics display computer is not functioning. It is important that the Alarm Display Package be fully installed and running for the alarm handling system to function correctly. If the display package is not running the alarm handling system will perform overhead and data aquisition without performing any alarm handling funcitons.

During alarm data base transfer from the off-line alarm handling system in the Chromatics and data base editting, the CHROM task is stopped to avoid difficulties with the serial line between the Chromatics and The PDP 11/03.

No error messages are generated by the task. However, if a SWEPSPEED error results form an open I/O buffer during alarm system restart, the task detects the error, rectifies it, and restarts the task.

E.7.4.2 Software Description

The CHROM task operates very closely with the DISPlay task since its main purpose is to maintain the private software link between the two tasks. The bulk of the task deals with this supervision. The inter-computer link portion functions the same as in the LISN and TALK tasks.

The CHROM task performs the following functions:

1) Listen to the TT2: serial line for an input via an input command.

2) If the response is not an empty data packet, the busy flag of the link to the DISPlay task (?N#) is checked. If the flag is set the program waits until it is not set. The data packet is inserted in N# and the busy flag is set.

3) The program next checks the link form the DISPlay task. If the busy/request flag ?M# is not set then an empty data packet is sent to the TT2: serial line. Program control is then returned to step 1. Otherwise, if the busy/request flag ?M# is set then a data packet is retrieved from $M#, the ?M# flag cleared, and the data packet is sent to the TT2: serial line.

Further program details can be obtained from the flowchart and listing.

E.8.0 KEYBOARD DRIVER

E.8.1 INTRODUCTION

The Keyboard Driver software provides software support for the special purpose operator keyboard described elsewhere. Although the functions included in this routine are specific to the prototype alarm handling system, the module may readily be modified to meet the requirements of other types of keyboards. The keyboard driver is a general purpose software driver for indiviually addressed keys accessible through digitial input/output type interfaces. The driver will not support multiplexed keyboards.

E.8.2 OPERATION SUMMARY

This section describes in brief the use of the Operator Keyboard Driver Module. The Operator Keyboard Driver Module is used as follows:

1) The Media Driver software module must be loaded into a SWEPSPEED job slot having a priority higher than the Keyboard Driver.

2) The Queue Manager software must be loaded into a SWEPSPEED job slot having a priority higher than the Keyboard Driver.

3) Ensure that the Keyboard Driver has been assigned the correct job locations of the Media Driver and the Q-Manager. Also ensure that the Queue number and Media link numbers are unique.

4) Load the appropriate Key Function and Key Text

arrays into the global area. This need not be necessary if new Function or Text arrays are to be generated.

5) Activate the Keyboard Driver using the $ACT command or other job interaction instructions.

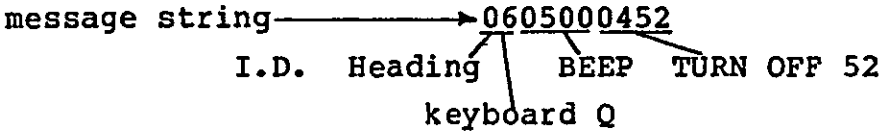
6) The Keyboard Driver will remain active until stopped by the user.

E.8.3 COMMUNICATION STRUCTURE AND PROTOCOL

The key board driver communicates with other software modules via the alarm handling system queue. (See Queue Manager Documentation). The driver utilises both an IN queue and an OUT queue to allow two way communication with other jobs. Key press functions can be assigned to indicate the transfer of text from the keyboard driver into the system queue. Also the driver will read messages received from its input queue. Messages received in this way are interpreted as key press function codes and are executed in the same manner as if a key press had been detected.

The message protocol from sending a command to the keyboard is as follows:

The key board driver input queue number is b. Key code functions, described in section E.8.7.8, are strung together and sent as a single string text packet



The message packet is placed into the queue system and subsequently sent on to the keyboard driver. The output message texts are defined by the user in the $T#(K) array. The text is sent verbatim into the system queue when the appropriate key press is detected. See section E.8.7.8 for further details.

E.8.4 SUPPORT TASK PRIORITY ASSIGNMENT

The Keyboard Driver utilizes two supporting tasks, the Media Driver and the Queue Manager. Job Priority assignments follow the rules outlined elsewhere for the supporting tasks. In general, the keyboard driver must be assigned a sufficiently high job priority to maintain a reasonable response time from the operator keyboard. With this in mind, it has been found that a priority assignment of 28 works satisfactorily in the prototype alarm handling system. The Media Driver and Queue Manager necessarily have job priority assignments higher than the keyboard driver.

E.8.5 ERRORS

The keyboard driver does not generate error messages with the exception of the Q routine. Any invalid key function code will not be executed. If an invalid function code is imbedded in a string of functions codes, all valid codes preceeding the invalid code will be executed. When the invalid function code is detected, the remainder of the function codes in the string are aborted.

The OUT Q routine within the driver will generate the following error message on the console VDU whenever the keyboard driver's output Q is full and waiting to be

serviced:

KB - Q WAITING

E.8.6 SOFTWARE FUNCTIONAL DESCRIPTION

The keyboard driver software module resides in the Swepspeed system as a job. The job, once activated, runs continuously until stopped by another Swepspeed job or command. The Media driver module, described elsewhere, must be installed in the system when activating the Keyboard driver module since the operator keyboard can only be accessed via the Media plant interface.

Functionally the driver performs the following operations:

1) Poll the operator keyboard inputs to determine whether or not a key has been pressed.

2) If no key has been pressed, wait a short while and try again.

3) If a key has been pressed determine which specific key was pressed.

4) Check key routine to determine whether or not the key is enabled, if not abort any further key functions and resume polling scan.

5) Locate the key function codes stored in $F#(K) where K is the Key number, see appendix.

6) Execute key functions.

7) Resume key board polling scan.

E.8.7 SOFTWARE DETAILED DESCRIPTION

This software description explains in detail the keyboard driver module flowcharts presented in the following section. Please refer to the flow charts to clarify the description presented here.

E.8.7.1 Set Up

This section sets the initial values of certain variables used in the driver module which are dependent upon the location of the keyboard module and other supporting modules in the Swepspeed system. The following variables are set as indicated below:

M = 10 = Media Driver Module Job number
M1 = 2 = Media Driver Communication link assignment (must be unique)
Q1 = 7 = QUEUE Manager Job number
Q = 1 = Keyboard Driver Output Queue Number (must be unique)

As the set up implies, the keyboard driver module requires both the Queue Manager and the Media Driver Module to be fully installed in the Swepspeed system to ensure proper operation of the keyboard driver. Any of the above variables may be set to other values without effecting the operation of the keyboard driver, however care must be taken to ensure that new assignments correspond with priority allocation requirements of the supporting jobs and that communication assignments do not conflict with other jobs. Rerer to the desciption of the other modules concerned for

more details.

E.8.7.2 Scan Keys

This section polls the digital input channels which have been assigned to the operator keyboard. The operator keyboard is physically connected to the computer via the Media plant interface package. In the prototype system, the keyboard keys are connected to the digital inputs on a one to one basis. Any key press will result in a single unique digital input being activated. Due to physical constraints the keyboard digital input assignments have been split, some are maintained contact type input and the remainder are fleeting contact type inputs. The behaviour of the inputs differs in the manner by which the inputs are read by the computer.

The result of this anomally is that the Scan Key section performs two types of key scan. Keyboard inputs assigned to Media channels 145-154 are the maintained input type. There are 16 such inputs on the Media card. The Media driver module allows the user to read the status of the entire card as an integer representation of the input status of all 16 channels (See Media Driver Documentation). The keys on Media Channel 145-154 represent 11 digital inputs on the card, so by scanning one channel in the group, a status can be obtained for the entire group. If the result of the status is non-zero a key press can be detected. The Media Channel number can then be calculated by adding the location of the TRUE bit in the input card status to 145 which represents the first media channel on the card.

A similar method is used for detecting the inputs on the Fleeting cards. Fleeting cards are organised in groups of 8 inputs. The corresponding Media Channels are 161 to

185. Similarly the inputs are scanned in such a manner that only one card status is obtained from the Media Plant interface. This is important when addressing the Fleeting inputs since the entire group of 8 inputs on a card is automatically reset whenever any channel on the card is read. By identifying non-zero status words indicating a key press, the bit location of the TRUE bit is added to the Media channel number of the first input on the card to establish the Media input channel number activated. There is a major difference in the way that the two types of Media inputs respond, this effecting the key operations. Key connected to Digital fleeting (DIF) Media Channels give only one output for each key process. Keys connected to Digital Maintained (DIM) Media Channels remain "on" as long as the key is pressed. This difference makes some of the key function codes available unsuitable for DIF keys.

All the above input information is obtained via the Media Driver Module. Details of the protocol for communicating with the Media Driver Module are described in the Media Driver Documentation.

E.8.7.3 Check In Queue

The Check In Queue Section is executed after each scan of the operator keyboard keys. A check is made to see if an addition has been made to the keyboard driver modules input queue. If so the input queue is serviced once. Only one incoming message is retrieved from the input queue and sent to the Key Function Section. Any further queue entries are processed after the next keyboard scan. The input queue messages are used to generate keyboard functions such as turning key backlight lamps on and off, etc. Message texts are interpreted as if they are key codes and operators. Key functions codes obtained via the module's input queue are not maskable. Execution of the text if valid is immediate.

The content of the input messages must be in accordance with the key function codes and operators as discussed in the Function Section. The procedures for servicing software module input queues is discussed in the Queue Manager documentation.

E.8.7.4 Poll Lapse

The Poll Lapse section in effect sets the keyboard scan rate. After each key scan procedure the driver suspends operations for 100 mS before starting again. This brief pause also allows lower priority jobs to be allocated processor time. The suspension time does not reflect the exact scan rate due to the unknown amount of time allocated to higher priority jobs.

E.8.7.5 Which Key

Having established that a key has been pressed, this section translates the Media Channel number of the key input into a Key number used by the driver to identify the location of key function codes, etc. The Key numbers (K) are 1 to 40 and are assigned as described in section 8.0. The translation of Media Channel numbers to Key numbers improves the flexibility of the system for use in other hardware configurations and also optimises the amount of array area required for look up tables.

E.8.7.6 Key Control

The Key Control section supervises the action taken when a key press has been detected. Firstly, the program calls the CK Mask Subrutine to determine whether or not the key mask shows that the key is enabled. The result of the subroutine is in ?M which is True of the key is disabled.

The key control will sound a brief tone burst from the operator keyboard's audible output device to indicate that the key pressed is no masked. No further key functions are performed and the program control is returned to the Scan Keys section if the key is masked. Finding the key enabled the Key Control sends program control to the Function subroutine described later. After completion of the Function subroutine Key Control returns program control to the Scan Keys section.

E.8.7.7 Check Mask

This subroutine returns the variable ?M which is True if the key number K is disabled as indicated by the mask array ?M( ). The mask is set or reset with Key function codes as described later.

E.8.7.8 Function

Key functions are executed by Function Section. Key Function codes are located by the routine and used to select predefined key function operations. A global string array $F#(40) = 20 contains user defined key codes and operations to build up key operation sequences. The array elements are identified by the Key number K. Each individual key can thus be defined to initiate the execution of any number of key function operations. The key code is used in conjunction with an operator. The key code identifies the type of operation and the operator provides additional information required. The key codes and operators are strung together to from a sequential execution of operations. When all operations are completed the Function Section returns program control to Key Control.

A list ot the key codes and operations currently available are presented below.

Code & Operator		Function	Description
00	-	NOP	No operation is performed.
01	nn	MASK	Disable the key represented by the key number nn.
02	nn	UNMASK	Enable the key represented by the key number nn.
03	nn	ON	Turn on the digital out represented by the Media Channel Number nn.
04	nn	OFF	Turn off the digital out represented by the Media Channel Number nn.
05	00	BEEP	Sound the keyboard audible output device for a short burst.
06	ff	CLEAR	Clear function with the following parameters:

ff=01 Clear key input mask (enable all keys)
ff=02 Clear all keyboard digital outputs
ff=03 Clear all (mask and digital outputs)

07	qq	SEND TEXT	Send the text contained in Key Text Array $T#(K) to the output queue. qq defines the termination character.

qq=01 CR
qq=02 ESC
qq=03 None

qq=04 Buffer key text
qq=05 Send and Clear Buffer with CR
qq=06 Send and Clear Buffer with ESC
qq=07 Send and Clear Buffer, no termination
qq=08 Delete last char in buffer

08	nn	HOLD ON	This operation suitable only for DIM type inputs, suspends further key function operations while the key is pressed. Hold on also turns the keyboard Media digital output number nn on until the key is released.
09	nn	TOGGLE	If the keyboard Media digital output number nn is already ON, the digital output is turned off. If the digital output is OFF then the output is turned on.
10	ff	SP FUNC	This operation selects via ff a variety of special functions specific to the prototype system. These functions enable the keyboard to respond correctly when interfacing with the PCP process control language OCP task.

The key function codes and operators are stored in $F#(K) from left to right representing the order in which the

functions are to be performed. Both the Text and Function array currently have a maximum of 20 characters available. Key message text is stored in $T#(K). The following examples illustrate the operations.

Example 1 : Key number K = 14 is pressed
$F#(14) = "035501140704"
$T#(14) = "Hello"

The key operations are executed from left to right.

0355 Turn on a Media Channel number 55
0114 Mask (disable) key 14
0704 Send "Hello" to the software device with input queue number 04

Example 2 : Key number K = 10 is pressed
$F#(10) = "035008550450"
$T#(10) = " "

0350 Turn On Media Output Channel number 50.
0855 Turn on Media Output Channel 55 and hold until key is released. Key number must represent a DIM type input. When key is released turn off 55.

0450 Turn off Media Output Channel number 50.

Key message texts are stored in $T#(40) = 20 by key number K. The contents of each element forms the message text plus a header. The header is used to identify the software device input queue for which the message is intended. A typical message text would be "10THREE". For more details on the queue communication system see the queue manager documentation.

K = the corresponding key number
$T#(K) = "01THIS IS A TEST"

If the key function code 0701 has been specified in $F#(K) the text THIS IS A TEST will be placed in a input queue of the software module or device which has been allocated input queue No 01. The operator in the following function code indicated the type of terminating character to be added to the end of the text.

01 = CR ASCII 13
02 = ESC ASCII 27
03 = No TERMINATING character

The above function codes are for single text messages. Additional codes are available to string or buffer text strings together before the message is sent into the queue.

04 = Buffer key text.
05 = Send to out queue with CR at end and clear.
06 = Send to out queue with ESC at end and clear.
07 = Send to out queue with no termination char, clear.
08 = Delete one character from the end of the buffer.

The leading characters in the key text describing the destination queue remains the same as previously discussed. When a buffer is built the header is taken from the first key text to be loaded into the buffer. Subsequent entries to the buffer are stripped of their headers before insertion into the buffer. Once the buffer is complete a key such as ENTER can be programmed with the fucntion 05, 06, or 07 to send buffer with the termination character as defined by the subfunction number. The buffer is also cleared. A key such as ERASE may be programmed with function code 08 to delete the last character in the buffer.

A string of key texts may be built and sent as follows:

Key	Key Text	Key Function
1	"011"	0704
2	"012"	0704
3	"013"	0704
ENTER	-	0705

The buffer text will read "01123CR".

These termination characters are required by some driver module output queue ready for receiving by the queue manager. Details of the generation of the output queue and other queue communication functions is discussed in the Queue Manager Documentation.

E.8.7.9 Queue

The Keyboard Driver incorporates both an IN and OUT Queue. The Queue assignments are OUT 1 and IN 6. The queues software is identical to calling job queue software as described in the Queue Manager Documentation.

E.8.8 SPECIAL OPERATOR KEYBOARD KEY ASSIGNMENTS

Key No.	Media Channel	Key Text
1	145	ACCEPT ALARM
2	146	PROCESS ALARM
3	147	COMPUTER FAIL
4	148	TEST LAMPS
5	149	START/YES
6	150	FUT HOLD
7	151	OPEN/THRU
8	152	CLOSE/DIVERT
9	153	HOLD
10	154	STOR
	155	
	156	
	157	
	158	
	159	
	160	
11	161	TIME
12	162	DISPLAY
13	163	LOOP
14	164	SEQUENCES
15	165	PLAD
16	166	REJECT
17	167	EXECUTE
18	168	CHANGE
19	169	SPEC CHANGE
20	170	7
21	171	8
22	172	9
23	173	4
24	174	5
25	175	6
26	176	M

27	177	S
28	178	1
29	179	2
30	180	3
31	181	C
32	182	I
33	183	0
34	184	.
35	185	-
36	186	R
37	187	-SPARE-
38	188	ERASE
39	189	ENTER

Status no.		Digital outputs
-	46	HIGHLAND ACCEPT
-	47	HIGHLAND TEST LAMPS
-	48	
1	49	TIME INDICATOR
2	50	DISPLAY INDICATOR
3	51	LOOP INDICATOR
4	52	SEQUENCES INDICATOR
5	53	PLAD INDICATOR
6	54	REJECT INDICATOR
7	55	ACCEPT ALARM INDICATOR
8	56	PROCESS ALARM INDICATOR
9	57	TEST LAMPS INDICATOR
10	58	EXECUTE INDICATOR
11	59	AUDIO OUTPUT RATE
12	60	AUDIO OUTPUT PITCH

E.9.0 MEDIA DRIVER MODULE

E.9.1 INTRODUCTION

The SWEPSPEED II Media Driver Module is used to send or retrieve data from the Alarm Handling Media Plant Interface hardware. The driver, written in SWEPSPEED II, allows the user to confidently communicate with the Media system through the use of normalised data transfer protocol. Since the Media is accessed through the PDP 11/03 memory addresses, use of the module prevents system failures caused by improper accessing. Media I/O Devices have differing forms of computer inputs and/or outputs which require substantial bitwise data manipulation. The driver module performs all necessary calculations yielding uniform normalised values. Error detection intercepts most user protocol and overflow errors and generates console error messages. The reader should be familiar with SWEPSPEED II before proceeding. A description of the SWEPSPEED II system can be found in the SWEPSPEED II User's Guide. The purpose of this document is to describe operation and use of the SWEPSPEED II Media Driver Module.

E.9.2 OPERATION SUMMARY

This section describes in brief the use of the Media Driver Module. More details are given in the subsequent sections. The Media Driver Module is used as follows:

1. The Media Driver software must be loaded into a SWEPSPEED job slot. This job slot must be assigned a higher priority than any job the driver services.

2. There are 5 bidirectional communication links with

the driver comprised of reserved global array variables. The subscripts identify the communication link number (n = 1 to 5).

3. When writing to the driver a job must load data, if any, into the appropriate global variable %V#(n), V#(n), and ?V#(n). The Media I/O Device channel number (refer to Appendix) is loaded into C#(n) last.

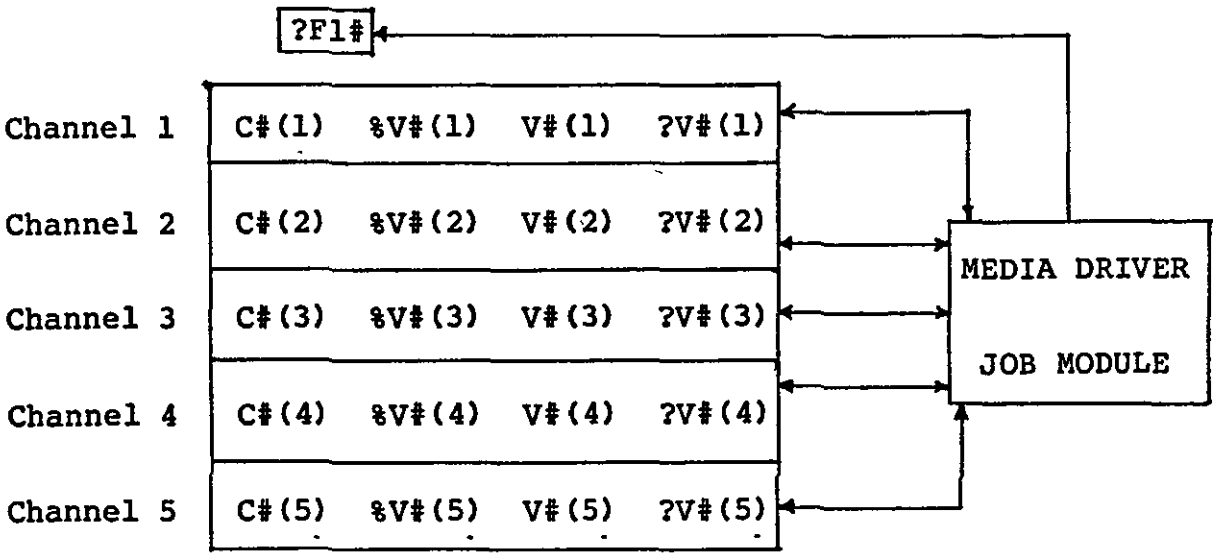
4. The calling job must start the driver using job interaction statements and then wait until C#(n) is 0 or less.

5. When using the driver to read data the calling job can read data, if any from the appropriate global transfer variable %V#(n), V#(n), and ?V#(n). If C#(n) = 0 the transfer is OK. If C#(n) is less than 0 an error has occurred.

6. ?Fl# is TRUE when the driver is running.

E.9.3 COMMUNICATION STRUCTURE AND PROTOCOL

Data transfer to and from the module is accomplished through reserved global transfer variables as follows:



Reserved Global Variables Job

There are five communication data links available. Each link contains space for a Media I/O Channel Number, a real variable, an integer variable, and a logical variable. The variables used for this data transfer are reserved for Media Driver use, that is, when the Media Driver module is installed in the system care must be taken not to use the same global variables for other purposes.

The reserved global variables are:

?Fl# - Media Busy Flag [True or False]

C#(n) - Media I/O Channel Number array

%V#(n) - Real Variable array normalised [0.0 to 1.0]

V#(n) - Integer Variable array [-32767 to +32767]

?V#(n) - Logical Variable array [True or False]

The individual communication links are necessary in a real-time multi-tasking environment to avoid collision or corruption of data as several tasks compete for the same global variables.

Each SWEPSPEED job that requires Media I/O data is assigned its own communication link 1 through 5. It is important that not more than one job uses a communication link. Communication through the link is bidirectional so that a single link is used for both output and input data. The generalised communication procedure is as follows:

1) The job requiring communication places any output data (if there is any) in the three array variables %V#(n), V#(n), and ?V#(n) where 'n' is the job's communication link number 1 to 5. Which variables, if any, need to be assigned depends upon the Media I/O device as shown in Section E.8.0.

2) The job now places the Media I/O channel number, also see Section E.8.0, in the array variable C#(n), where 'n' is the job's communication link number 1 to 5.

3) The job must now start the Media Driver with the interjob instruction START.

4) The job must now test the value of C#(n). If C#(n) is 0, the communication is complete and the appropriate data is located in the three array variables %V#(n), V#(n), and ?V#(n) as specified in Section E.8.0. If C#(n) < 0 then an error has occured and the data transfer is not valid. The error codes are as follows:

C#(n) = -1 -- invalid Media Channel Number

C#(n) = -2 -- analogue overflow

E.9.4 JOB PRIORITY

The Media Driver must be assigned a job priority higher than any job that uses the module for effective operation. This is necessary to insure that the driver is not interrupted by a job which may change a global transfer variable at a critical moment. Since a variable is comprised of more than one byte of information, job interruptions between byte transfers could lead to erroneous data if several jobs were trying to access a global transfer variable.

E.9.5 DRIVER - JOB HANDSHAKING

The Media Driver can not detect when a data transfer is required so therefore the job module requiring communication must start the driver with a START instruction. In the same manner, the calling job cannot detect when the Media Driver is finished, so the calling job must test the value of C#(n) as previously described. WARNING: Any attempt to use the Media Driver Module with no power on the Media I/O hardware will result in a fatal software error. i.e., SWEPSPEED will crash.

E.9.6 COMMUNICATION LINK PRIORITY

The Media Driver scans the C#(n) array to determine which communication links require servicing. When a Media Channel Number is detected the driver assumes that service is required. This array is scanned in the order 1 to 5, so lower numbered communcation links have higher priority. It

is important that when the communication link is loaded that the Media Channel number C#(n) is loaded last to prevent data collision.

E.9.7 ERRORS

As previously mentioned the drive can detect most errors which can cause system failure. An error code is placed in C#(n) in response to a data transfer request which causes an error. All error codes are less than 0. When an error does occur a message is also displayed on the console which gives an indication of the type of error generated. The module can not detect the absence of power on the Media I/O hardware. If the module is used in this case SWEPSPEED will crash.

Error Code	Console Message	Description
-1	NON-EXISTENT MEDIA I/O ERROR	Improper channel no.
-2	MEDIA ERROR AO = xxx.xxx	Overflow on output calls. %V#(n) must be 0.0 to 1.0.

E.9.8 MEDIA I/O DEVICE DATA

In this section the input and output information associated with specific Media I/O Devices is discussed. Note that all real variable data must be normalised 0.0 to 1.0. See appendix for nomenclature explanations.

E.9.8.1 AOV's and AOI's - Analogue Outputs

%V#(n) Write Only - Load normalised output data 0.0

to 1.0 where 0.0 = 0% and 1.0 = 100% output range.

V#(n) Not Used

?V#(n) Not Used

E.9.8.2 AI - Analogue Inputs

%V#(n) Read Only - Contains normalised input from analogue input where 0.0 = 0% and 1.0 = 100% of input range.

V#(n) Not Used

?V#(n) Not Used

E.9.8.3 DO - Digital Outputs

%V#(n) Not Used

V#(n) Read Only - Contains integer representation of the current status of the entire digital output group in which the channel number addressed resides. The least significant (LS) bit is the LS output of the group of 16 where a 1 bit = TRUE.

?V#(n) Write Only - Load TRUE for output 'ON'

E.9.8.4 DIM - Digital Inputs Maintained

%V#(n) Not Used

V#(n) Read Only - Contains integer representation of the current status of the entire digital

input group in which the channel number resides. The LS bit is the LS input of the group of 16 where a 1 bit = TRUE.

?V#(n) Read Only - Contains TRUE for input 'ON'

E.9.8.5 DIF - Digital Inputs Fleeting

%V#(n) Not used

V#(n) Read Only - Contains an integer representation of the current status of the entire digital input group in which the channel number resides. The LS bit is the LS input of the group of 8 where a 1 bit = TRUE. WARNING: Whenever a fleeting input card is addressed, the entire group of 8 is reset to FALSE.

?V#(n) Read Only - Contains TRUE for input 'ON'. WARNING: Whenever a fleeting input card is accessed the entire group of 8 is reset to FALSE.

E.9.8.6 WD - System Monitor Watch Dog

%V#(n) Not Used

V#(n) Read Only - Contains an integer representation of the watch dog status word. See Watch Dog documentation.

?V#(n) Read Only - Contains TRUE for system OK.

E.9.9 SAMPLE PROGRAMS

Job 2 = User's job assigned to communication link 3.

Job 10 = Media Driver Module

E.9.9.1 Analogue Output Example

User's Job 2

```
.
.
50 %V#(3)=0.5              %V# contains the normalised output
60 C#(3)=12                set Media channel 12
70 START10                 start driver
80 IFC#(3)>0GOTO70    |    if error detection required
90 IFC#(3)=-1GOTOxx   |
100 IFC#(3)=-2GOTOxx  |
.
.
```

E.9.9.2 Digital Output Example

User's Job 2

```
.
.
40 ?V#(3)=TRUE             ?V# contains the output status
50 C#(3)=26                set Media channel 26
60 START10                 start driver
70 IFC#(3)>0GOTO60    |    if error detection required
80 IFC#(3)=-1GOTOxx   |
90 IFC#(3)=-2GOTOxx   |
100 T=V#(3)                optional read of group status
.
.
```

E.9.9.3 Analogue Input Example

```
User's Job 2
•
•
70 C#(3)=109              set Media channel 109
80 START10                start driver
90 IFC#(3)>0GOTO80
100 %T=%V#(3)             transfer data
110 IFC#(3)=-1GOTOxx   |  optional error detection
120 IFC#(3)=-2GOTOxx   |
•
•
```

E.9.9.4 Digital Input Example

```
User's Job 2
•
•
80 C#(3)=167              set Media channel 167
90 START10                start driver
100 IFC#(3)>0GOTO90
110 ?T=?V#(3)             transfer status
120 T=V#(3)               optional group value reading
130 IFC#(3)=-1GOTOxx   |  optional error detection
140 IFC#(3)=-2GOTOxx   |
•
•
```

E.9.10.0 SOFTWARE DESCRIPTION

The Media Driver Module is comprised of software routines specifically developed for dealing with the Media Plant Interface hardware. The plant interface consists of a variety of different types of interface cards. Each type of interface card requires a software routine to code or decode information to or from the cards. The Media Driver Module recognizes which type of interface card is being addressed and subsequently selects the appropriate software routine. Routine selection is based on the channel number requested by the calling job.

E.9.10.1 Setup

The Setup section of the Media Driver Module contains all variable initialisation.

E.9.10.2 Communications

The module communicated with other program tasks via software links as described previously. When the Media Driver is started by a calling job, the program scans all communication links to see if a service request has been entered. This is accomplished by scanning the channel number array C#(N), where N is the Media link number. If the value of C#(N) is greater than zero, a request is noted. The values of the global transfer variables are inserted into local variables. Program control is temporarily transferred to the Channel Select section for further processing of the I/O request. When the I/O request has been serviced program control returns to the communication section. Local program variables containing Media data are inserted into the corresponding global transfer variables. The channel number global variable C#(N) is then set to zero or less. The calling job can detect that the Media

servicing is complete by noting the change in C#(N). Error detection is performed in the I/O servic routines. If an error has occurred, C#(N) will be less than zero. The value is the error code number.

Each link is checked in the same manner and sent off for further processing if necessary. When the scan of the links is complete, the driver is stopped unless a link had made a request. In this case the procedure begins again until no requests are found on a scan of the Media software links.

The global transfer variables which make up the links are comprised of many different variable types. As a result it is important that the Media Driver job priority is higher than any calling job's priority. Also by convention the global transfer variable C#() is always the last variable to be processed. These steps insure that the transfer variable %V#() is not corrupted. A read or write to a real variable requires a two byte transfer. Programs of differing priority may collide by trying to access a real global variable 'at the same time'.

E.9.10.3 Channel Selection

Having found a service request in the Communication section, the channel selection section branches program control to the appropriate input/output routine for the channel requested. The user must ensure that Media channel numbers do indeed allow the program to branch to the correct subroutine for a specific hardware configuration. In the prototype alarm handling system channel asssignments have been based on the configuration of the Media interface hardware. Channel numbers are assigned consecutively form 1 which represents the first available I/O port at the lowest available Media memory address of 160000 octal. Refer to

Table E.9-2.

The channel number is retrieved from the transfer variable C#(N). The link number N has been established by the Communication section. Program control is passed on to the appropriate subroutine.

An error can be generated by this routine when a non-existent Media channel number is requested. The error code is -1. No branching to a service subroutine occurs when a none existent channel number error is encountered. Program control returns to the Communication section.

E.9.10.4 Service Routines

Each Media interface card requires special software routines. Media cards are electrically located in the computer in the I/O page of memory. The exact location is selected by the user such that the Media interface does not conflict with any addresses used for other computer interfaces. The addresses selected for the prototype alarm handling system start at 160000 octal and continue upwards. To read or write data to the Media interface cards the program must perform a memory read or write command. Following are decriptions of the software service routines available. Refer to the flow charts and listing for more details.

E.9.10.4.1 AI Routine

There are two types of analogue input interface cards used in the prototype alarm handling system, individually accessed and multiplexed analogue inputs. Multiplexed inputs use a single analogue input card and an attached multiplexer card which selects one of up to 16 different input lines to be connected to the analogue card. Accessing

inputs therefore requires an input line selection 1 to 16 on the multiplexer, a wait until the analogue card settles, and then read the data from the analogue card. Based upon the Media Driver channel number the program service routine calculates the appropriate multiplexer and multiplexer line to be used. The multiplexer is notified. Then the routine waits for the 50 microsecond settling time of the analogue card and reads the input data.

Individually accessed analogue cards do not require settling time so that the data can be read immediately. This methos allows the fastest form of analogue input with the unfortunate difficulty of having to provide a separate card for each analogue variable to be measured.

The Media analogue input cards have a 10 bit resolution. When reading the card these 10 bits are located in the most significant bits of the 16 bit memory word corresponding to the analogue input card. The unused bits are set to 1 by the card. The service routine calculates the normalized value 0.0 to 1.0 as a proportion of the bit pattern obtained from the analogue card.

No errors are generated by this routine.

E.9.10.4.2 AOV & AI Routines

Analogue output cards are available with either voltage or current outputs. AOV are voltage outputs while AOI are current types. Both cards are dealt with in the same manner. The Media interfaces require a 10 bit pattern. The bit pattern is comprised of the most significant bits of the 16 bit word located in the memory location at which the interface card resides. The analogue output routine converts the normalized output value %V into a proportion of the output represented by the octal values 0 to 1777. The

memory location of the output cards is calculated and the output value is transferred to the address. Program control is returned to the Communication section with the local variables set to zero. In the event that the normalized variable %V is out of the range 0.0 to 1.0, an error -2 is generated and no service is performed.

E.9.10.4.3 DIF Routine

Fleeting digital input card are unique in that once the card is read the card resets all inputs to read 0. There are 8 inputs on each card. The condition of the inputs is represented by a 1 for true in the most significant 8 bits of the 16 bit word representing the card. The remainder of the bits are set to 1. The service routine calculates the input card to be read based upon the channel number requesetd. The card is read and, for convenience, the top 8 bits are placed in the least significant bits of the integer transfer variable V# resulting in an integer representation of the inputs on the entire card or group. The channel number requested is used to calculate which bit in the 8 bit pattern is to be returned in the logic transfer variable ?V. In this way both a group representation can be retrieved along with an individual channel status.

No errors are generated by this routine.

E.9.10.4.4 DIM Routine

Maintained digital input cards have 16 input channels. Unlike the fleeting card types, these input cards are not reset when read. The service routine calculates in which group of 16 the Media channel number requested is located and the appropriate card is read. Data retrieved is in negative logic, i.e., 0 = true. The program converts the values into positive logic. Since the cards have 16 inputs,

all inputs are read simultaneously and the 16 bit pattern is placed in the integer transfer variable V. The bit location of the individual channel requested is calculated, and the value is placed in the logic transfer variable ?V.

No errors are generated by this service routine.

E.9.10.4.5 DO Routine

Digital output cards used in the prototype system are 16 way open collector types. The 16 bit pattern sent to these cards is translated into ON for a 1. Since individual bits within the pattern can not be addressed separately, the service routine stores the current status of the outputs in a digital output store array A(0). Each word (2 bytes) in the array represent the output for a digital output card. When output requests are serviced, the individual bit location is calculated and inserted in the output store array. The array elements are then written to the digital output cards.

No errors are generated by this service routine.

E.9.10.4.6 WD Routine

The watchdog service routine writes a special bit pattern to the Media System Monitor card. No error messages are generated by this routine. More details concerning the watchdog card are described in the watchdog documentation.

TABLE E.9-1 - MEDIA I/O DEVICE NOMENCLATURE

AI	Analogue Input
AOI	Analogue Output Current Type
AOV	Analogue Output Voltage Type
DIF	Digital Inputs with Fleeting Contacts
DIM	Digital Inputs with Maintained Contacts
DO	Digital Outputs
WD	Watch Dog System Monitor

Table E.9-2 I/O Device Channel Allocation

	I/O Device	Ch. No.
AOV1	0-10V AN OUT	1
AOV2		2
AOV3		3
AOV4		4
AOV5		5
AOV6		6
AOV7		7
AOV8		8
AOV9		9
AOV10		10
AOV11		11
AOV12		12
AOV13	0-10V AN OUT	13
AOI1	0-10MA AN OUT	14
AOI2	0-10MA AN OUT	15
WD	WATCH DOG	16
DO1	DIG OUT OPEN COLLECTOR	17
DO2		18
DO3		19
DO4		20
DO5		21
DO6		22
DO7		23
DO8		24
DO9		25
DO10		26
DO11		27
DO12		28
DO13		29
DO14		30
DO15	DIG OUT OPEN COLLECTOR	31

DO16	DIG OUT OPEN COLLECTOR	32
DO17		33
DO18		34
DO19		35
DO20		36
DO21		37
DO22		38
DO23		39
DO24		40
DO25		41
DO26		42
DO27		43
DO28		44
DO29		45
DO30		46
DO31		47
DO32		48
DO33		49
DO34		50
DO35		51
DO36		52
DO37		53
DO38		54
DO39		55
DO40		56
DO41		57
DO42		58
DO43		59
DO44		60
DO45		61
DO46		62
DO47		63
DO48		64
DO49		65
DO50		66
DO51	DIG OUT OPEN COLLECTOR	67

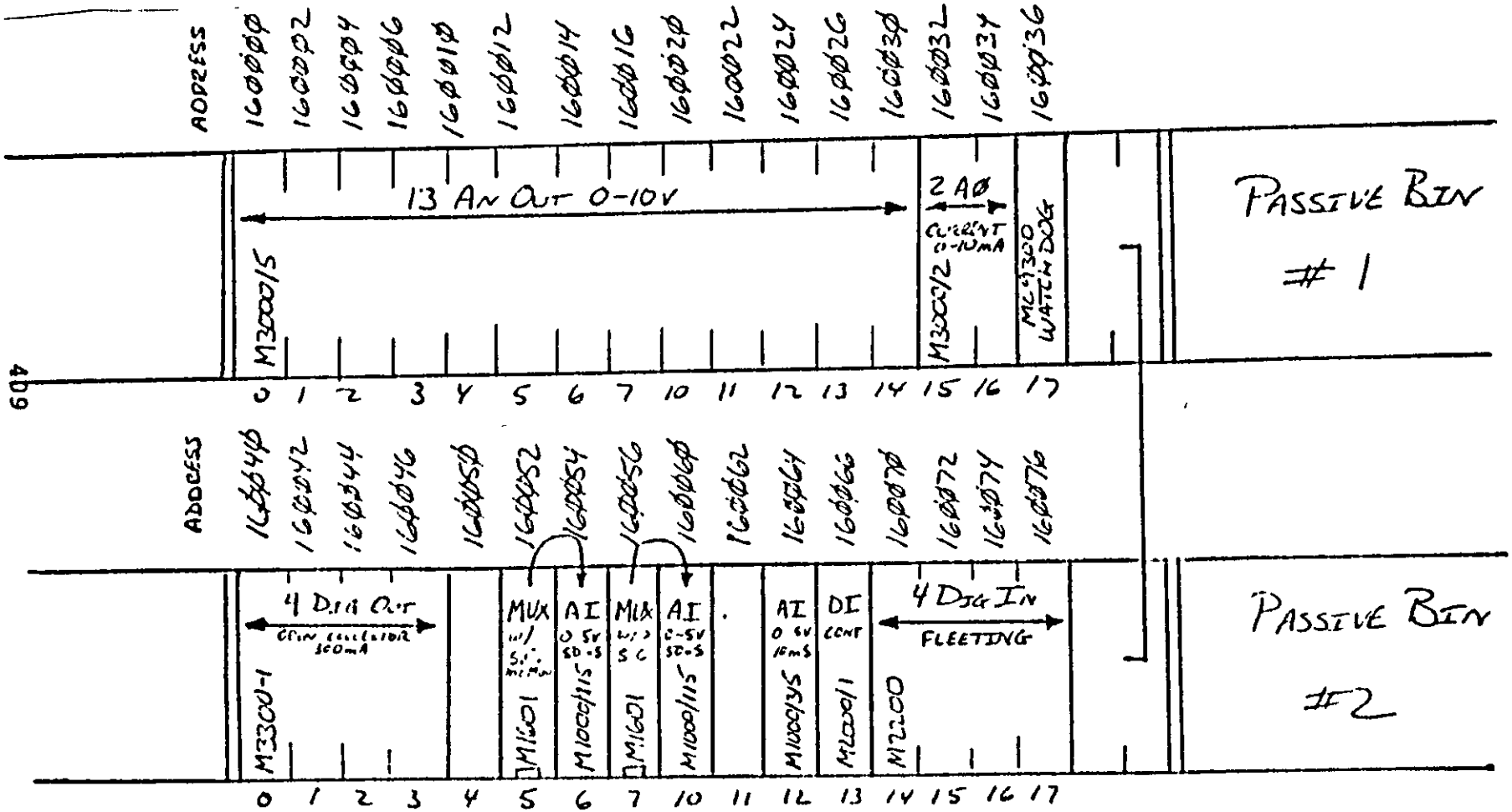
DO52	DIG OUT OPEN COLLECTOR	68
DO53		69
DO54		70
DO55		71
DO56		72
DO57		73
DO58		74
DO59		75
DO60		76
DO61		77
DO62		78
DO63		79
DO64	DIG OUT OPEN COLLECTOR	80
	RESERVED	81
		82
		83
		84
		85
		86
		87
		88
		89
		90
		91
		92
		93
		94
		95
	RESERVED	96
AI1	0-5V AN IN	97
AI2		98
AI3		99
AI4		100
AI5		101
AI6		102
AI7	0-5V AN IN	103

AI8	0-5V AN IN	104
AI9	0-20MA AN IN	105
AI10	0-20MA AN IN	106
AI11	0-10V AN IN	107
AI12		108
AI13		109
AI14		110
AI15		111
AI16	0-10V AN IN	112
AI17	0-5V AN IN	113
AI18		114
AI19		115
AI20		116
AI21		117
AI22		118
AI23		119
AI24		120
AI25		121
AI26		122
AI27		123
AI28		124
AI29		125
AI30		126
AI31		127
AI32	0-5V AN IN	128
	RESERVED	129
		130
		131
		132
		133
		134
		135
		136
		137
		138
	RESERVED	139

	RESERVED	140
		141
		142
	RESERVED	143
AI33	0-5V AN IN	144
DIM1	DIG IN MAINTAINED	145
DIM2		146
DIM3		147
DIM4		148
DIM5		149
DIM6		150
DIM7		151
DIM8		152
DIM9		153
DIM10		154
DIM11		155
DIM12		156
DIM13		157
DIM14		158
DIM15		159
DIM16	DIG IN MAINTAINED	160
DIF1	DIG IN FLEETING	161
DIF2		162
DIF3		163
DIF4		164
DIF5		165
DIF6		166
DIF7		167
DIF8		168
DIF9		169
DIF10		170
DIF11		171
DIF12		172
DIF13		173
DIF14		174
DIF15	DIG IN FLEETING	175

DIF16	DIG IN FLEETING	176
DIF17		177
DIF18		178
DIF19		179
DIF20		180
DIF21		181
DIF22		182
DIF23		183
DIF24		184
DIF25		185
DIF26		186
DIF27		187
DIF28		188
DIF29		189
DIF30		190
DIF31		191
DIF32	DIG IN FLEETING	192

Table E.9-2 Media Technical Information



Media Layout

4/11/81
GH

E.10.0 DATA ACQUISITION

E.10.1 INTRODUCTION

A principal characteristic of any device which requires data from the surrounding environment is the data aqcuisition task. In the alarm handling system the data acquisition task DA performs this function. The DA task allows the system to obtain process data from any task capable of generating such data having access to the queue communication system. In addition, the DA task has a software link with the Media Driver module. Directives are sent in the form of data packets to appropriate tasks. These commands initiate the acquisition of process data from individual plant sensors. The operation of the DA task is supervised by the DACON data acquisition controller task which is described elsewhere. The order and frequency with which the data points are sampled is determined by the system definition as defined by the contents of the alarm data base.

E.10.2 OPERATION SUMMARY

The DA task is a core alarm handling system task. System tasks are not accessible to the user in normal operation and therefore the functioning of the DA task does not require user intervention.

E.10.3 SOFTWARE DESCRIPTION

E.10.3.1 Setup

The DA task requires program support from the Media Driver, the Q Manager, and the Event Processor. The job

slot locations of these support tasks are assigned here.

E.10.3.2 Run

The program control section of DA coordinates the subroutine selection in the task. The software selects the locations in the alarm data base which correspond to the variable descriptions of the various process variables to be measured. Each variable description takes the same amount of space in the alarm data base. The DA tasks steps through the sets of data which are organised according to priority and scan group. Starting with the first variable description the DA task continues to step through the descriptions up to the limit set by the Data Acquisition Controller task.

Each variable description is examined and a process data request is sent to the appropriate I/O device. When the data is retrieved, the software examines the data type and calls up the appropriate subroutine to deal with the data. When complete the next variable description in the alarm data base is processed and so on.

When all variables are completed, the program looks to see if a significant change has occurred in any measured variable. If so the event processor is started.

With all data acquisition functions completed, the program stops. It will be started by DACON when the next data scan time period has elapsed.

Refer to the SETUP task documentation for the descriptions of the Data Acquisition storage arrays.

E.11.0 DATA ACQUISITION CONTROLLER

E.11.1 INTRODUCTION AND DESCRIPTION

The purpose of the Data Acquisition Controller (DACON) task is to supervise the operation of the Data Acquisition task. Process variables can be scanned at various rates with the maximum scan rate of 1 second. The alarm data base contains variable definitions describing where and how to obtain process data represented by these descriptions. These variable descriptions are ordered sequentially in the alarm data base according to scan rate and priority within each scan rate group. Each time the Data Acquisition task is executed the program starts at the beginning of the variable description list, i.e., the process variable with the highest scan rate and priority. After obtaining process data for this variable the task steps to the next variable definition and so on. The DACON task sets the upper limit to which the Data Acquisition task will step through the alarm data base variable definitions. The upper limit is set according to the scan group to be executed. For example, at one second intervals the maximum limit is set to the top or end of the one second scan group. Every five seconds the limit is set to the top of the five second scan group so that both the one second and the five second group are scanned during the data acquisition. The process continues for the remainder of the scan groups. The DACON task then resets this top limit to the appropriate level based upon the current real time. Once the limit is set the Data Acquisition task is started. Error messages are issued in the event that the Data Acquisition task has not completed a data scan during a one second interval.

The DACON task is started by the COMAH alarm handling command task. The command task also suspends the operation of DACON during some alarm handling systems functions.

E.11.2 SOFTWARE DESCRIPTION

The DACON task is a small supervision task. The operation of the software should be readily understood by examining the flow charts and program listings. Described here is a summary of the software functions.

E.11.2.1 Setup

The Setup section of the program contains the usual variable assignments relating to the job slot positions of supporting tasks. In this case the variable D is used for the job slot location of the Data Acquisition task.

E.11.2.2 Run

The program periodically checks the current real time clock values. Depending upon the time, the variable P0# is set to the maximum alarm data base variable definition to which the Data Acquisition task should scan. Once the scan group is determined based upon the cuurent real time, the maximum address (P0#) to be used in the Data Acquisition task is calculated. The value of P0# is dependent upon the number of variable definitions is each scan group. Information regarding the number of variable definitions in each scan group is obtained from the alarm data base header. See alarm data base documentation for further details. Additionally the DACON task will check the operation of the Data Acquisition task to ensure that the scanning is being completed within a one second scan interval.

E.12.0 EVENT PROCESSOR

E.12.1 INTRODUCTION

The event processor is one of the core alarm handling tasks. This task establishes a binary event status image representing the current mode of operation of the process plant. Using event definitions contained in the alarm data base, individual data acquisition units are converted into a true/false event status. The event definitions describe all conditions in which a particular data acquisition unit is to be considered as indicative of an event occuring on the plant. A wide variety of event types are available to suit most needs such as analogue, binary, or contact type data. In the present version of the alarm handling system, there is a one to one correspondence of events to data acquisition units. It is exspected that further additions such as derived events using multiple combinations of events and time related events can be added. The event processor software includes the provisions for such improvements.

E.12.2 OPERATION SUMMARY

The operation of the event processor is automatic and generally transparent to the operator. The event processor is started by the DA task whenever a significant change is noted in a data acquisition unit. Once started the event processor steps through event definitions as described in the alarm data base. Event definitions establish the measured variable data type and the variable event true band. The current value of the data acquisition units is compared with the event limits and pervious plant data. The processor then decides if an event has occurred. The event status image in E#() is updated as necessary and the alarm

generator is notified of the change.

E.12.3 SOFTWARE DESCRIPTION

The event processor task is relatively short however the functions in the task have been condensed, obscuring their operation.

E.12.3.1 Setup

The setup section assigns the alarm generator slot number, sets the busy flag, and defines the hysteresis truth table.

E.12.3.2 Find Change

When the processor is started by the DA task, the program tests the data change flag ?D# to see if there have been significant changes in the data acquisition data store. If so the D1#(0) array identifies which DA units have changed indicated by a 1 bit in the array.

E.12.3.3 Change

Once a change has been noted the program searches sequentially through the event definitions to locate all definitions containing the DA unit in question. Using the definition the event type is used to branch program control to the appropriate event type subroutine. Here the values of the DA unit are compared with event parameters to see if an event has occurred. If a change in event status is noted the event status image E#(0) is updated and the alarm generator is started. Program control is returned to the Find Change section to search for further significant changes in DA units.

E.12.3.4 Hysteresis Sub

Analogue event detection requires a method for detecting the relative change in process variables in order to establish which parameter range limits are to be used to evaluate the event. This is necessary since the hysteresis feature requires the event processor to know how the variable is changing. As described in the Off-Line User's Guide the range band limits are defined by the parameters 1, 2, 3, and 4. These range parameters define 5 regions in which the measured variable may be located.

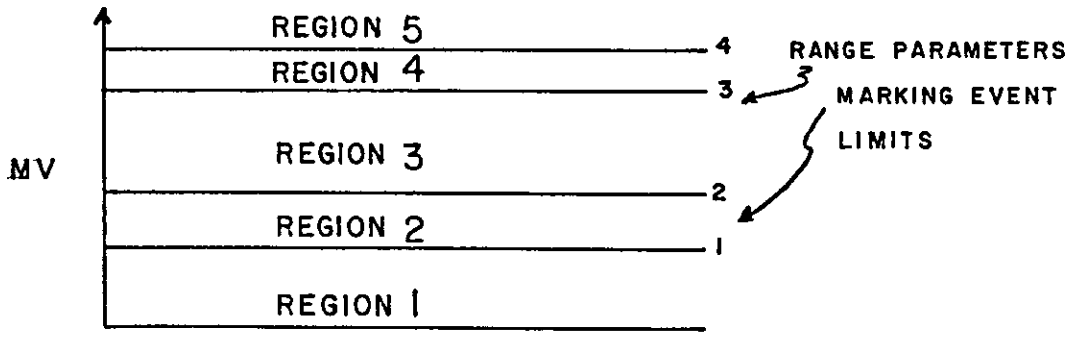


Figure E.12.1 Using regions for hysteresis calculations

As variables make excursions from one region to another the processor must select the range parameter to be used for the event detection. There are 25 different transitions that a variable can make from one region to another. From a transition table, a truth table can be developed as follows:

Transition Number	Region Old	Region New	Does this represent a true status		Octal Truth Table
0	1	1	N	0	
1	1	2	N	0	4
2	1	3	Y	1	
3	1	4	Y	1	
4	1	5	N	0	1
5	2	1	N	0	
6	2	2	Y/N	0	034614
7	2	3	Y	1	6
8	2	4	Y	1	
9	2	5	N	0	
10	3	1	N	0	4
11	3	2	Y	1	
12	3	3	Y	1	
13	3	4	Y	1	3
14	3	5	N	0	
15	4	1	N	0	0
16	4	2	Y	1	
17	4	3	Y	1	3
18	4	4	Y/N	0	
19	4	5	N	0	
20	5	1	N	0	4
21	5	2	Y	1	
22	5	3	Y	1	000143
23	5	4	N	0	1
24	5	5	N	0	

In the program the truth table is stored in bit form in H(0), H(1). The transition number is calculated as follows:

Transition = (Old -1)*5 + New - 1

The transition number is then used to determine if a change has occurred in the event status image.

E.13.0 ALARM GENERATOR

E.13.1 INTRODUCTION

The alarm generator is one of the core alarm handling system tasks. Event status images produced by the event processor are examined by the alarm generator each time a change occurs in the stauts image. Based upon enhanced Boolean expressions stored in the alarm data base, the alarm generator determines whether an individual alarm is now on or off. The Boolean expressions are comprised of combinations of events and Boolean operators stored in a coded Reverse Polish Notation as compiled by the off-line section of the alarm handling system. In the alarm data base the alarm definitions are comprised of two coded expressions one for the alarm ON condition and the other for the alarm OFF condition. Depending on the current status of a particular alarm, the alarm generator will examine the appropriate condition expression to see if the current status of an alarm should be changed.

E.13.2 OPERATION SUMMARY

The alarm generator operation is transparent to the operator. It's operation is fully automatic as the task is started by the event processor whenever a change in the event status image is detected. Using the event status image as input data the alarm generator evaluates alarm definitions in the alarm data base sequentially. There are no error messages generated by the alarm generator task.

E.13.3 SOFTWARE DESCRIPTION

The alarm generator software has been condensed into a closely packed program to maximize the efficiency of the program. The structure of the program is the same as others

in the alarm handling system, however several subroutines exploit the condensed nature of the alarm data base. It may be useful to examine the off-line documentation for a better understanding of the processing of the data base information.

E.13.3.1 Setup

The DISPlay task is required as a supporting task. The job slot location is assigned here.

E.13.3.2 Run

The Run section coordinates program control of the task. Firstly the section locates the beginning of the lists of alarm definitions in the alarm data base. The total number of alarm definitions is placed in S. The program next examines each alarm definition in the same manner by locating the first three records in the definition. These records represent 1) the alarm output code, 2) persistancy (not presently used), and 3) the length of the ON condition expression. The current status of the alarm is next checked. The Run section calls either the ON condition or OFF condition subroutine for processing the coded Boolean expression in the alarm data base. When the expression is processed and the alarm status image is updated, the program halts.

E.13.3.3 Check for ON or OFF

Depending upon the routine called these subroutines process the appropriate portion of the alarm data base representing the coded Boolean expression required. These expressions are coded in Reverse Polish Notation (RPN). The program processes these expressions using an RPN stack defined as ?S() with P as the stack pointer. Elements are

extracted from the Boolean expressions and temporarily placed in R. If R is less than 0 then the program recognises the element as a Boolean operator. Program control branches to the Boolean processor routine and performs the appropriate function on contents of the stack ?S(). If R is positive, then the program recognises the element as an event status image position bit location representing an individual event status. R is then inserted into the next available location in the stack ?S(). Once processing the Boolean expression, the result of the expression, if the expression has been correctly coded, will be at the top of the stack ?S(). Currently the stack size is set at 25, i.e. DIM ?S(25). Program control is returned to the Run section.

E.13.3.4 Check Result

This section checks the result of the Boolean expression. The result is compared with the present status of the alarm definition. If there has been a change in the alarm condition, the result of the Boolean expression will match the current alarm status. This is the case since when the status of an alarm changes, the expression chosen to evaluate its new status will represent the opposing condition statement result.

If a change is found, the alarm status image is updated and a message is sent to the DISPlay routine via the 'private software link' as described in the DISPlay task document. The message is a combination of the display function code either a 1 for OFF or 2 for ON and the alarm output code as obtained from the data base alarm definition.

E.14.0 DISPLAY TASK

E.14.1 INTRODUCTION

The DISPlay task coordinates all operator display functions generated by the alarm handling system. Display coordination is necessary in order to ensure that tasks within the alarm handling system which compete for the same display device, do not generate conflicting data and produce corrupted display information. In the prototype alarm handling system, the DISPlay Task coordinates display information for the VT-100 console terminal, the T-43 printer and display protocol for the Chromatics based display package. The DISPlay task has full access to the Alarm Handling System queue system thus allowing two way communication with the majority of tasks within the system. The DISPlay task also contains a communication task with the Media Driver. Function codes in incoming message packets are used to identify which output display device is to be implemented.

E.14.2 OPERATION SUMMARY

Generally the operation of the DISPlay task is transparent to the operation of the alarm handling system. Described here is a brief outline of the major functions of the DISPlay task. Further details of its operation are given in subsequent sections. The DISPlay task performs the following functions:

1) Supervises the communication links with the Chromatics display computer.

2) Supervises output to the T-43 printer log.

3) Deals with alarm output messages as received from the Alarm Generator.

4) Monitors queue system for output requests.

5) Coordinates all printed output to peripheral devices.

E.14.3 COMMUNICATION STRUCTURE AND PROTOCOL

The DISPlay task monitors three software communication lines:

1) The Q via INQ.

2) The Alarm Generation task via private link.

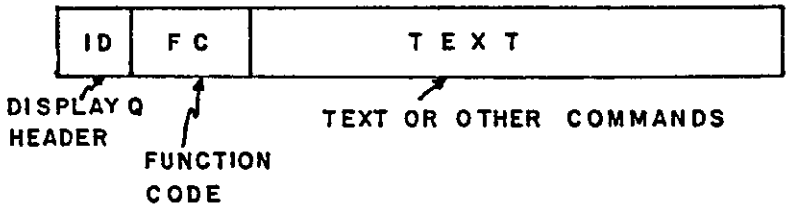
3) The Chromatics link via a private link to the Chromatics link task.

The protocol for all three links is the same, i.e. in the form of data packets comprised of strings. Private software link lines are used in some cases since the communication is exclusive to the DISPlay task. The data packets received by the DISPlay task are processed in a similar fashion as those in the Keyboard driver module. Leading function codes in the string are used to select the required display operation. The function code is a single numeric character 1 to 6. Following this function code operations the function operand. Depending on the function code selected the operand may be output text on further subfunction codes. The available display functions are as follow:

Function Code	Operation

1	Alarm OFF sends received alarm output code to Chromatics and requests a Media digital channel to be turned off.
2	Alarm ON sends received alarm output code to Chromatics, requests a Media digital channel to be turned on, and keeps the Keyboard audible device.
3	VT-100 prints text to console terminal
4	CHROM send text to Chromatics display computer.
5	T-43 send text to T-43 printer log.
6	Q send text to Q system.
7 8 9 0	Reserved for user specified functions.
L	Overlay LOAD task for transfer of alarm data base from OFF-line system.

The received data packet has the following format:



E.14.4 SUPPORT TASK PRIORITY ASSIGNMENTS

There are several priority and job slot conditions which must be set for the DISPlay task to function correctly. Firstly, the Chromatics link task CHROM must be in job slot 13. The Q Manager and Media Driver modules must be assigned a higher priority than the DISPlay task.

E.14.5 ERRORS

During normal operation no errors should be encountered. The program does not generate any of it's own error messages. During the OFF-line transfer of the alarm data base to the ON-line computer and during data base editing, the DISPlay task will be overlayed. In the event that the DISPlay task is not correctly overlayed back into its job slot after such an operation, unusual errors will occur. In this case enter the COMAH command task and re-execute the EDit mode which will make another attempt to overlay the task.

E.14.6 SOFTWARE DESCRIPTION

Careful examination of the flowcharts and listings of this task will help clarify the operation of this task.

Some explanation is required of the private communication link lines.

E.14.6.1 Private Software Links

The private software links are used for the exclusive communication between two jobs. The link is comprised of a data packet swapping routine similar to that used in the inter-computer link tasks. A unique global string variable is used for the transfer of the data packet through the one-way link. A unique global logical variable is used as the service request and busy flag. The operation of the link is as follows:

1) Sending job examines logical variable to see if link is busy. If it is, wait until flag is cleared then place data packet in global transfer variable and set busy flag.

2) Receiving job routinely polls the link request/busy flag. If set then a data packet is retrieved from the global link variable and the request/busy flag is cleared.

In the DISPlay task global variable assignments are as follows:

$N#, ?N#	Link from Chromatics link task
$M#, ?M#	Link to Chromatics link task
$M1#, ?M1#	Link from Alarm Generator task

E.14.6.2 Setup

The Setup section contains the usual job slot, link, and queue assignments.

E.14.6.3 Run

The Run section's principal function is to scan all incoming sources of data packets. These include the INQ Chromatics link, and the alarm generator link. If a service request is detected this sections transfers program control to the Function Select section. Otherwise if no service request is found the task stops.

E.14.6.4 Function Select

This section strips the leading function code from the data packet and branches program control to the appropriate subroutine. When complete the program control is returned to the Run section.

The function subroutines are simple and require little explanation. Subroutines retrieve the data as required from the data packets. Canned routines for the Media Driver, INQ, and OUTQ routines are used.

When an 'L' function code is encountered, the program attempts to overlay the LOAD overlay task located on tape storage over itself. When this occurs an 'X' is sent to the Chromatics link task to shut down the link. If the Chromatics link remains active during the data base transfer, errors will occur.

E.15.0 INTRODUCTION

Due to restricted memory space in the PDP 11/03 computer, two tasks LOAD and EDIT. are in overlay form. This means that the tasks are stored on the tape drive storage system until needed at which time the task is loaded into a job slot. This is the DISPlay task job slot as well. It is convenient to use this slot as the overlay slot since by stopping the DISPlay task the alarm handling system operation is suspended, not stopped, until the DISPlay task is re-installed. Since the DISPlay task must be reinstalled, there also exists an overlay file for the DISPlay task on the same tape drive.

It is important to note that the overlay tape must be properly inserted in the tape drive DD0: throughout the duration of the use of either LOAD or EDIT. Also, if for any reason the global variable table in the alarm handling system is modified. the system manager must INSTALL the LOAD the EDIT, and the DISPlay tasks. Refer to the SWEPSPEED user's guide for further details on overlay jobs and job slots.

Described in this section are software descriptions of the overlay tasks LOAD and EDIT. The DISPlay task although existing as an overlay file as well is described in a separate section.

E.15.2 EDIT

The EDIT task permits the user to make modifications to the alarm data base when installed in the on-line computer. The EDIT task is an overlay task called via a command ED in

the command task COMAH. When the EDIT task is called, the alarm handling system is halted and the EDIT task is overlayed from tape drive DD0: into job slot 2. Once in edit mode the prompt > appears signifying that the system is ready for edit commands. Modifying the alarm data base is not recommended since any changes must be executed with a full knowledge of the structure of the alarm data base and its functions. Incorrect data entries will result in the malfunctioning of the alarm handling system when restarted. Major changes made to the alarm data base should be performed in the off-line develpoment system.

E.15.2.1 Operation Summary

The EDIT task once evoked via COMAH exhibits the prompt >. The following commands are then available:

P Send a copy of the current alarm data base to TT1:, the T-43 printer.

L List the contents of the alarm data base to the console.

R Replace an element in the alarm data base with a new value.

I Insert a new element into the data base.

D Delete or remove an element from the alarm data base.

X Exit the edit mode, re-install the deisplay task in job slot 2, and return to the alarm handling command task.

Error messages occur during incorrect task overlays and when

the alarm data base needs to be resized. Refer to the on-line user's guide for a full explanation of the operation and the effects of the various edit commands.

E.15.2.2 Software Description

The EDIT task is an overlay task located on the Alarm Handling overlay tape. Being an overlay the task can only be loaded into a pre-selected job slot. In this case job slot 2 is use for all overlay tasks. The DISPlay task which is normally in job slot 2 is overwritten by the overlay. Disruption of the DISPlay task in this manner insures that the alarm handling system stops functioning while the overlay task is being executed. Being an overlay file the EDIT task can be loaded into the system by either an OVERLAY statement or a system INSTALL utility.

The program is simple in operation and requires little explanation. A monitor using a > prompt selects the appropriate subroutine as defined by the user response. The Setup section contains a definition of the legal inputs. The Run section selects the function subroutine. The various subroutine functions are best explained by examination of the program flowchart and listing.

E.15.0 LOAD

The LOAD task is an overlay task for use with the off-line development software. During the operation of TRANSFER in the off-line system, the Chromatics unit installs an alarm data base in the PDP 11/03 on-line alarm handling computer. The LOAD task when installed in the PDP 11/03, forms the receiving module for the data transfer. Communication protocol between the Chromatics off-line system and the on-line system consists of a simple data

packet exchange routine. The LOAD task sends an ASCII text string data packet in exchange for a data packet received from the Chromatics. Data packets sent by LOAD contain information concerning the status of the on-line system and also error information pertaining to the transfer of data. Refer to the off-line software document for additional information regarding the function of this task.

E.15.3.1 Operation Summary

The LOAD task operation is fully automatic once the task overlay has been installed by the DISPlay task. The DISPlay task installs the LOAD task when it receives an 'L' via the CHROM link task to TT2: serial line. The 'L' is sent by the off-line portion of the alarm handling system resident in the Chromatics computer when an alarm data base transfer is to occur.

The LOAD task replaces the on-line functions in the PDP 11/03 of the CHROM link task. The link is maintained by the LOAD task for transfer purposes. The task responses with the correct code to execute data base transfer. If an error is detected the program aborts and the normal on-line tasks are re-installed and started.

E.15.3.2 Software Description

Little explanation is required for this task since the program itself provides a good explanation. A few points must be made.

The off-line program first tests to see if the LOAD task is present by sending an 'L' at regular intervals until the LOAD task give the correct coded response. Once the link is established both program modules, the LOAD task and the off-line TRANSFER program, check to ensure that each

other is operational. With the link firmly established, the off-line program sends the new alarm data base array size. The LOAD task test this size to see if there is room. If there is no room, no transfer takes place and the program task is aborted. If all is well the data packet transfer rate is increased and the alarm data base is overwritten with incoming data. Any transfer error detected by either transfer task will cause a negative value error code down line. If this should occur, the process is aborted.

E.16.0 ALARM DISPLAY PACKAGE

E.16.1 INTRODUCTION

The Alarm Display Package is an independent hardware /software system intended to complement the operation of the prototype Alarm Handling System. Although the Alarm Display Package does not perform any alarm information generation, it does provide the primary means by which the alarm handling system displays alarm information to the operator. For prototype development, the software has been generalised in such a manner that the end user can readily 'add on' plant specific display formats without the modification to the display system's communication and data processing structures.

The display package runs in an intelligent colour graphics terminal which is connected to the alarm handling computer via a serial link. Software within the terminal is capable of identifying alarm output and other codes as distributed by the Alarm Handling Computer. The codes are interpretted as display commands to evoke appropriate alarm texts and/or mimics on the display screen. A data packet swapping protocol is used for communication between the display package and the alarm handling system. Although there is no alarm information generated by the display package, the data packet swapping enables the alarm handling computer to ascertain the health of the display package and to ensure that data transfer to the display package is complete and error free.

E.16.2 HARDWARE

The display package utilizes a Chromatics CG 1999 intelligent colour graphics terminal. This terminal is also

used for the off-line production of the alarm handling system data base. Operation of the Chromatics in the off-line mode is described elsewhere. The Chromatics is a Z-80 microprocessor based device with 28K of RAM memory. A Microsoft Basic interpreter is included in the unit in addition to a comprehensive selection of colour graphics operating system commands. A single 8 inch floppy disk unit is used as the mass storage device. Communications with the Alarm Handling System computer is by means of an RS232 serial line port. For a further description of the Chromatics unit refer to the Chromatics Operators Manuals.

E.16.3 OPERATION SUMMARY

This section describes in brief the use of the Alarm Display Package. Further details are given in the subsequent sections. the Alarm Display Package is used as follows:

1) The Chromatics display computer must be connected to the alarm handling computer. The connection is made via TT2: on the PDP11/03 and via SIO0: on the Chromatics.

2) Turn on the Chromatics and insert the Display floppy disk into the disk drive.

3) Press (RESET) (BOOT) and (BASIC.)

4) Enter Memory size? &HB000 (RETURN)

5) Type DOS"LOAD DISPLAY" (RETURN)

6) Type RUN (RETURN)

7) Enter the display data base number to be used.

8) The display package is now up and running ready for alarm data to be received from the alarm handling computer. No further operation on the Chromatics are required.

E.16.4 COMMUNICATION STRUCTURE AND PROTOCOL

The communication protocol between the Chromatics and the PDP11/03 alarm handling computer is in general transparent to the operation of the overal alarm handling system. The commuication structure is based on a simplified ring system. Data packets are passed between the Chromatics and the alarm handling computer. The data packets contain alarm output codes and other operating instructions. If no data is to be transferred, the data packets are sent regardless, filled with code indicating that the data packet is empty. In normal operation the Chromatics is waiting for input via the serial I/O link line. The serial line is connected to the PDP11/03 device TT2: and the Chromatics SIO0: port. The data packet is in the form of ASCII numeric codes which represent the type of function the display package is to perform.

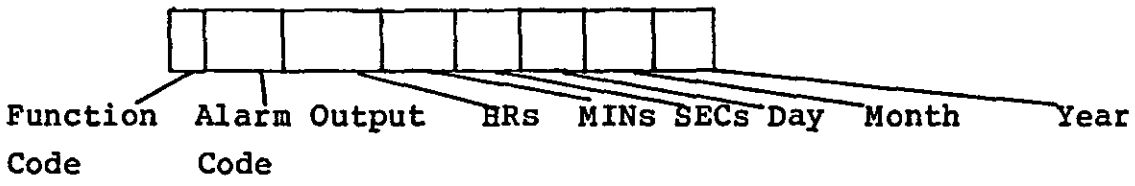
The data packet communication is bidirectional. The link task CHROM in the alarm handling computer sends a data packet which may or may not be empty. The alarm display unit responds with a data packet which again may or may not be empty. The procedure continues at a rate of approximately 3 exchanges per second. The constant exchange of packets is used by the computers to establish the health of each computer system. Data packets incoming to the display unit are executed immediately therefore no data packet is returned until the requested function has been completed error free.

The alarm display unit returns either an empty data packet or a packet containing alarm texts for printing. The

alarm text packets are headed by the appropriate function code required for printing via the DISP task in the alarm handling computer. See DISP documentation for further information. Empty data packets in either direction are coded as a '*'. Note: The alarm display linker task CHROM is used by the alarm handling display task DISP, both of which reside in the alarm handling computer. It should also be noted that since the TT2: I/O line has access to the DISP task via the link task CHROM, all functions available through the DISP task are available via the link line.

E.16.5 DATA PACKET STRUCTURE

The data packet structure as received by the alarm display package is a combination of an alarm output code and time/date information as follows:



The function code header identifies the display function to be evoked. The alarm output code immediately following the function code is used to identify the alarm to which the function code pertains. The alarm output code range is 000 to 999. Not all function codes require an alarm output code. In these cases all data following the function code is ignored if it is present. Time/date information optionally follows the alarm output code as shown in the above figure. Hours, minutes and secondes are in 24 hour format.

E.16.6 FUNCTION CODES

1+ALC+[time,date] Deactivate all alarms with alarm

	output code ALC, i.e., alarm condition no longer exists.
2+ALC+[time,date]	Activate a new alarm with alarm output code ALC. i.e., a new alarm has occured.
3+ALC+[time,date]	Accept the longest outstanding alarm with alarm code ALC.
4	Accept all outstanding active alarms.
5	Used for paging displays to roll display screen up a small amount and repack alarm list.
6	Used for paging displays to roll display sreen down a small amount. This function does not repack the alarm list.
7 8 9 0	Reserved for user defined functions.

E.16.7 ERRORS

The display package software does not generate error messages. If a software error does occur, the prototype

system uses the normal BASIC error trapping which stops the program and prints an error message on the display screen. The display system must be re-started in order for the alarm handling computer to function correctly. Obviously errors of this sort should not occur in normal operation, however, if and error should occur the BASIC software must be carefully examined for corruption. To re-start the system follow the start up procedure described in a previous section. The alarm display data present in the system before the failure can not be retreived.

E.16.8 SOFTWARE FUNCTIONAL DESCRIPTION

As previously discussed the alarm display package is an independent, stand alone device. The BASIC interpreter available in the Chromatics can run only one program at a time unlike the multi-tasking environment present in the alarm handling computer. The alarm display software must therefore incorporate all communication, processing, and display coding in one program. The software correspondingly is segmented into these three functions. The alarm display software is intended to be a core module upon which the user can readily add user defined display function. Software 'hooks' are provided for this purpose. The strictly modularized structured format aids in the addition of custom display modules.

Data communication is accomplished in the I/O subroutine. Time out error detection is used to detect the absence to responses from the link line. For details of the protocol of the link line see the previous section. Once a data packet is received the message is decoded. Depending upon the leading function code the program branches to the appropriate subroutine to process the alarm output data.

The core of the alarm display package is a data array

in which all current alarm display information is stored. Access to information in the array is organized in list processing format. As new alarm data is added to the display list the array pointer associated with each data record define the location within the display list. As entries are deleted due to the disappearance of an alarm, the associated alarm data in the display list is deleted simply by alteration of the list pointers. Dealing with the list in this fashion minimizes the processing time of the program. This is especially important to the operation of the overall alarm handling system since the single task display unit can not queue up display requests coming in from the link line.

Following the alarm list processing the program proceeds to display routines which can be user defined. In the case of the prototype system, a paging type of alarm display is available. Another display routine is also available to generate mimics of alarm annunciator panels on the VDU screen.

Once completing all display tasks, the program returns to the I/O routines for further instructions.

E.16.9 SOFTWARE DETAILED DESCRIPTION

The alarm display software is highly modularized. The major software sections are as follows:

SETUP
CONTROL
I/O ROUTINES
ALARM LIST PROCESSING
DISPLAY GENERATION

Refer to the flow charts and listings following this

section for further clarification of the text descriptions.

E.16.9.1 Setup

This initial routine is executed only at start up of the alarm display package. The Setup peforms two key functions

1) Initialize the maximum alarm list size.
2) Initialize the alarm list for list processing.

The alarm list size is defined in the variable w. The alarm list array is P(W,4) with it`s accompanying text array T$(W).

The alarm list array P(W,4) must be assigned with initial pointers and values as follows:

P(Wo,1) = PRECEEDING ARRAY LOCATION IN LIST

P(Wo,2) = NEXT ARRAY LOCATION IN LIST

P(Wo,3) = The alarm code as oobtained from alarm list entry

P(Wo,4) = The current status of the particular alarm list entry.

T#(Wo) = The associated text for the alrm code as defined in P(Wo,4).

All alarm list elements are prenumbered to establish the list processing network.

E.16.9.2 Alarm List Initialization

A summary of the variables associated with the alarm lists are initiated in the setup routine and are as follows:

N = Total number of entries in list

FI = First array element in list

LA = Last array element in list

P() = Alarm list

T#()= Text for alarm list

PN = Text available empty locations in list

W = Maximum size of alarm list

I = Current location in list, ie. the actual array element number.

Other variables associated with the list are transient in nature and therefor difficult to define. However, an examination of the program text should make these factors apprent.

The setup section is extended only once at startup, after which program control is passed on to the RUN control module.

E.16.9.3 Alarm List Status

Each entry in the alarm list is tagged with a variable whose value is dependent upon the current status of the list

element. The alarm status is located in the alarm list array P(W,4). There are four status values as follows:

0 = No entry in this alarm list element.

1 = This alarm has been accepted and will be removed from the list when the alarm conditions no longer exits.

2 = This alarm is active. It has not been accepted and the cooresponding alarm conditions still exists.

3 = List elements in the dondition are alarms which represent alarm conditions which no longer exists however the alarm has not been accepted.

The alarm list status information is used pricipally for identification of the manner by which the alarm should be displayed for example, an active alarm may be displayed in a different way from an accepted alarm.

E.16.9.4 Run Control

The run control module coordinates all activities of the alarm display package. Essentially the run control module is comprised of a series of GOSUB statements. Appropriate subroutines are called as necessary. The order in which the module executes is as follows:

1) Goto Loader routine - load in alarm text and data format arrays.

2) Goto Display Screen Initialisation routine - set up display.

3) Goto I/O routine - send and/or obtain data from the link line.

4) Examine the data received from the link to obtain function code.

5) Goto Decode routine - if appropriate to decode remainder of data package.

6) Locate if necessary the appropriate alarm list location.

7) Goto the appropriate alarm list processing routine .

8) Goto Display routine if required.

9) Return to step 3 and begin again.

E.16.9.5 I/O Routine

The I/O routine coordinates communication on the serial link line SIO:0. The routine uses the convention of nomenclature used throughout the alarm handling system as follows:

M$ = input data strings.
N$ = output data strings.

First the routine prints the contents of M$ out to the link line verbatim. If M$ is empty, I.e., is equal to a null string, a '*' is printed which is the normal indication of an empty data packet.

Next the output port is turned off to prevent echo. An

input statement is executed to input data into N$. The program is set up to 'time out' after one second in the event of no input. In this case the routine prints another '*' down the line and waits for input again. The procedure continues until a response is received from the link line, after which the output port is turned back on to re-establish the echo on the line.

Upon completion, program control is returned to the Run Control module. N$ contains the received data packet and M$ will have been emptied after having been sent down line.

E.16.9.6 Decode

The Decode module strips the alarm and other information from the incoming data packets. A predefined funcion FNCO(,) is used to select the appropriate characters from the data packet string. The decode routine returns program control to the Run Control module with the following variables. If a portion of the data string is missing the returned variables are equal to zero.

F = Function code 0 - 9.

C = Alarm output code 000 - 999.

T0 = Hours 0 - 23.

T1 = Minutes 0 - 59.

T2 = Seconds 0 - 59.

D0 = Day, day or month.

D1 = Month 1 - 12.

D2 = Year, eg. D2 = 82.

E.16.9.7 Alarm List Processing Functions

The available alarm list processing function are as follows:

ACTIVATE
ACCEPT
GENERAL ACCEPT
NO ALARM
INSERT
DELETE
LIST/REPACK

Each of these functions is comprised of a separate program module clearly defined in the program listing. The Run Control module calls the appropriate functions depending upon the alarm function code as received from the serial link line. In this section each of the list processing funcitons will be discussed.

A) ACTIVATE The activate routine is evoked with function F = 2. The routine adds the alarm output code to the next available location in the alarm list. The total number of entries in the alarm list is incremented. Finally the status code of 2 and the alarm code is assigned to the list location.

B) ACCEPT This routine changes the status code of the first entry in the alarm list which matches the alarm output coe as received form the link line. If the status code is 2 then the code is changed to 1. If the status code is 3 then the code is changed to 0. The function code F = 3 evokes this routine.

C) GENERAL ACCEPT The function code F = 4 selects the general accept routine which 'accepts' all outstanding active alarms in the alarm list. These are represented by the status codes 3 or 2 which are respectively changed to 0 or 1.

D) NO ALARM This routine is represented by the function code F = 1. When an alarm condition no longer exists the status code is changed for all alarm list elements containing the associated alarm output code. An active status code of 2 is changed to 3. If the alarm has been accepted, signified by the status code 1, then the status is changed to 0.

E) INSERT AND DELETE The insert and delete routines either add or remove an alarm list element from the alarm list. The insert routines initiates a display routine which will add the additional alarm to the display.

The delete routine removes the alarm from the list only if the status code for the alarm list element is zero. The pointers in the alarm list array are adjusted to compensate for the change in array element order. The deleted array element is cleared and becomes the new last element in the array. The variable I is the array element to be deleted. The deleted routine does not initiate any display routine, only performs the removal of an element in the alarm list.

E.16.10.0 DISPLAY PERSONALITY MODULES

The alarm display software discussed so far is the basic alarm display routine which executes functions required by the majority of alarm displays. Software specific to various types of display formats is dependent upon the type of hardware used and the formats themselves. The alarm display software is therefore structured in

modular form to facilitate the user to develop specific display function routines. The prototype alarm display system has two types of alarm display presently available:

1) VDU alarm annunciatorn panel mimics.

2) Conventional alarm paging display formats.

The software discussed in the subsequent sections deals with these display formats.

E.16.10.1 Alarm Paging Display

Alarm paging displays present alarm information to the operator in a chronological list format. Often the maximum available VDU screen area does not allow all alarms to be displayed at the same instance. Typical paging formats allow the operator to move the VDU display 'up and down' the list.

In the alarm display package commands received via the serial link line are available to execute the various paging functions. In particular these include rolling the screen display area up and down, repacking the list and relisting. Additional display information is placed at the top and bottom of the screen, primarily comprised of current alarm, total number of alarms, and the number of alarm entries above and below the display page.

The software module specific to the paging display are:

SCREEN INIT.
SCREEN UP
SCREEN DOWN
PRINT/ ADD
REMOVE

UPDATE

The Run Control module is modified to execute these routines after alarm list processing has occured. Note that alarm display commands F=5 and F=6 have been added to the core alarm list processing functions. Additional variables associated with the paging display include:

PG = Top of page location
L and X1 = List Locations for printing.

E.16.10.1.1 Screen Initialisation

This program module sets up the screen display for paging. In general the VDU screen is divided into three separate addressable windows 0, 1, 2. Printing colours and cursor operations is set. Refer to the Chromatics Operation manual for further explanation of the set up procedures.

E.16.10.1.2 Screen Up and Down

These display functions move the contiguous sections of the alarm list which is displayed up or down the list. Each time the functions are executed, the section displayed is moved up or down by ten lines.

E.16.10.1.3 Print/ Add and Remove

These routines either add, modify, or remove alarm texts from the screen with the following criteria:

Alarm Status

0 Remove
1 Print an 'A' preceeding the alarm text.
2 Print a '*' preceeding the alarm text.

3 Print a '*' preceeding the alarm text.

E.16.10.1.4 Update

The update routine adjusts the display data printed at the top and bottom of the display screen.

E.17.0 THE ALARM DATA BASE

E.17.1 INTRODUCTION

The alarm data base contains a definition of the functions to be executed in the on-line alarm handling system. Alarm system functions developed by the user in the off-line system are coded and condensed by the off-line compiler into a compact form for use in the on-line system. The on-line system uses the data base much like a program. The data base therefore defines the operation of the on-line system. In normal operation it is not important for the user to understand the structure and organisation of the data base. However, under certain circumstances it may be convenient user to examine or modify the contents of the alarm data base on-line without returning to the off-line system. Presented in this section are details of the alarm data base format as it would reside in the on-line system.

The alarm data base (ADB) is formed by the computer functions in the off-line system. A variety of data bases can be generated and stored on the off-line floppy disk unit. The ADB is stored in the form of a data file (.DAT) comprised of lists of real numbeers. When the ADB is transferred to the on-line system, these data files are conveyed verbatim to the on-line system. As the values are transferred, the on-line computer places the values consecutively into a real global variable array %A#() starting with location 1. Although real variables require twice the memory space as integers, it was found convenient to use reals since items such as range parameters are inevitably reals. No doubt this is an area which should be examined more closely. A real data base requires much more memory space and additional accessing time over integer based storage.

The alarm data base is organised into four major sections:

1) Data Base Header
2) Data Acquisition Variable Definitions
3) Event Definitions
4) Alarm Definitions

E.17.2 DATA BASE HEADER

The data base header is the only section of the ADB which remains consistent regardless of the structure of the remainder of the data base. The header contains key locations and other information in the data base required by the on-line system. The array elements 1 - 11 comprise the header as follows: NB In programing terms the element number should read as the array subscript.

Element Number	Contents Description
1	Alarm data base size, i.e., the maximum number of elements in the ADB.
2	The number of data acquisition variable definitions.
3	The number of variable definitions in data scan group 1.
4	The number of variable definitions in data scan group 2.
5	The number of variable definitions in data scan group 3.
6	The number of variable definitions in data scan

group 4.

7 The starting address of the first event description.

8 The starting address of the first alarm definition.

9 The required size of the data acquisition storage arrays.

10 The number of event status outputs. The size of the event processor arrays can be calculated from this value.

11 The number of alarm status outputs. The size of the alarm generator arrays can be calculated from this number.

E.17.3 DATA ACQUISITION

Data Acquisition variable definitions begin at element number 12. Variable definitions are all 6 elements long. The definitions are organized consecutively according to scan group and priority within the scan group. The lowest scan group number and the highest priority definition starts at element number 12. The format of each variable definition is as follows:

Element No.	Contents Description
12	Plant Code
13	Input device number - 1
14	Data type number

15 Range low

16 Range high

17 Significant change value

18 Start of next variable definition.

The values in the variable definition are discussed in the off-line documentation. Presented here is a brief summary of the contents. The plant code is the numeric code which is sent to the input device as a data request. The input device number directs where the data request is sent. The data base value is one less than the actual device number. The data type number represents the following:

0 Binary
1 Binary Inversion
2 Analogue Conversion 1
3 Analogue Conversion 2
4 Analogue Conversion 3
5 Analogue Conversion 4

The high and low range values defines the operational limits o the input device. Finally, the significant change value represents the amount analogue input value must change before the event process is started. Significant change and range values apply only to analogue input variables.

E.17.4 EVENT DEFINITIONS

Event definitions follow directly after the data acquisition variable definition. There are two types of event definitions, one binary type and the other analogue

type. The difference is the length of the definition record. Each event definition is organised as follows:

Element No.	Contents Description
N	EP Data Packet size
N+1	Data acquistion address location in data acquisition store
N+2	Event type
N+3	Band parameter 1 (analogue only)
N+4	Band parameter 2 "
N+5	Band parameter 3 "
N+6	Band parameter 4 "

The event types are defined as follows:

Value	Event Type
1	OFF
2	ON
3	XLO
4	LO
5	HI

6	XHI
7	TREND
8	DEVI
9	TDEVI

E.17.5 ALARM DEFINITIONS

Alarm definitions follow the event definitions. The order of the alarm definitions is based on the alphanumeric ordering of the alarm names assigned in the off-line development software. Each alarm definition is organized as follows:

Element No.	Contents Description
N	Alarm output code
N+1	Persistance value
N+2	Number of elements in ON condition Boolean expression
N+3 . .	ON condition Boolean expressions
N'	Number of elements in OFF condition Boolean expression
N'+1 . .	OFF condition Boolean expressions

The alarm output code is the alarm identification which is sent to the alarm display package. This code is sent along with status headers when the Boolean ON or OFF condition expressions are satisfied. The persistance number is not presently used.

The Boolean condition expressions are code in reverse Polish notations with operands being positively sized value indicating the location of events int the event status store. Negatively sized values are operators as follows:

Value	Operator
-1	NOT
-2	OR
-3	AND
-4	XOR
-5	NXOR
-6	SEQ
-7	TIL
-8	VOT

E.18.0 AN INTRODUCTION TO SWEPSPEED II

E.18.1 INTRODUCTION

SWEPSPEED II is a multi-tasking user oriented operating system and language primarily intended for real-time use. The software was developed by the Central Electricity Generating Board. Full details of the software package are given in the SWEPSPEED II User's Guide. The language is suitable for all PDP-11 type computers. The software is based on the DEC RT-11 operating system, however in operation SWEPSPEED II appears as a memory resident independent operating system and high level language. The SWEPSPEED II in the Alarm Handling System is a subset version, therefore some commands and facilities are not available as described in the user's manual. Normally a system is generated by the system manager to meet the requirements of a particular application. The purpose of this document is to give a brief introduction to the whole package. For further details refer to the SWEPSPEED II User's Guide.

E.18.2 CONVENTIONS

The SWEPSPEED II operating system is comprised of utilities and system commands. System commands are resident in memory. Utilities either are in memory or located on the TU58 tape storage and are automatically overlayed in memory when required. Any command or utility must be prefaced with a <control-c> character. No system command or utility will be accepted by the system until the $ prompt appears on the console. Any entry into the system must be followed by a <carrage return>. Only one utility or command may be evoked at a time.

Although the SWEPSPEED II language resembles BASIC in

syntax, care must be taken to insure that the SWEPSPEED syntax is strictly followed. As with many high level languages, few error messages are given in response to syntax errors.

E.18.3 LOG IN AND LOG OUT

SWEPSPEED II supports a single user environment with full user protection facilities. In order to enter the system a user must LOG in as follows:

LOG IN

1. Enter <control-c> and wait for $ prompt.
2. Enter 'LOG' <cr>. NB. <cr>= carrage return
3. 'name?'; enter your three letter user name then <cr>.
4. 'password?'; enter your three letter password then <cr>.
5. You are now logged in.

Once logged on the system responds to <control-c> with the command job dollar prompt. To log out of the system use the LOG command. only this time reply with a <cr> when prompted for the user name.

LOG OUT

1. Enter <control-c> and wait for $ prompt.
2. Enter 'LOG' <cr>.
3. 'name?'; <cr>.

E.18.4 OVERVIEW OF PROGRAM DEVELOPMENT

The system is organised into a number of jobs or job slots. The total number of available jobs is specified by

the system manager. Each job contains space for a program written in SWEPSPEED language. Selected job slots have been assigned to each user. A user only has access to his own jobs.

Each job is developed as follows:

1. Preparation of job source.
2. Compilation of job to produce a runable form.
3. Activation of a job.

E.18.4.1 Preparation of Job Source

New job programs are entered or existing jobs are modified through the use of the EDIT utility. The particular job number to be edited must be specified. Program text may now be entered. Editing is completed when terminated with the END edit sub-command. Some syntax errors are detected by the editor in which case the line is only displayed up to the point of the first error. The line must be retyped. Using the NAME edit sub-command gives new jobs an identification for future reference. Tip: Keep jobs short!

E.18.4.2 Compilation of Job

Jobs which have been edited or read in from a file on the TU58 tape unit require compilation before activation. The EDITor utility actually performs much of this function itself, however the COMpilation utility completes the process. During compilation variables are zeroed, data storage is allocated, and line number references are checked. The number of the job to be compiled is specified when entering the utility. All jobs which are to be activated or clock connected must be compiled. If a compiled job is stored, only the source is saved, that is,

when a stored job is reloaded it must be recompiled.

E.18.4.3 Activation of a Job

Compiled jobs may be activated, i.e., run, by using the system command ACTivate followed by the specified job number. Jobs can also be controlled by other jobs using the SWEPSPEED job interaction statements. The stop a job use the system command STOp followed by the specified job number.

E.18.5 FILE STORAGE AND LISTING

All files are stored on the TU58 tape unit. There are two drives in the unit. One tape contains the utility files and a bootable image of the system. This tape is located in the left hand drive and is referred to as drive DD0:. It is important that this tape is not removed during system operation since user jobs may be corrupted. The user file storage tape is located in the right hand drive and is referred to as drive DD1:.

Jobs can be stored on the user tape with the SAVe system utility. The job source code is written to a named file on the tape store for subsequent use.

Jobs stored on the tape can be reloaded using the system utility OLD. Only files belonging to the logged in user can be retrieved.

The DIR system command is used to obtain a listing of the user tape directory on the console VDU.

The utility LISt is used to obtain a listing of a job on the console VDU. A printed copy can be obtained by SAVing a job to device TT1: which is the T43 teletype. Make

certain that the TT1: is installed and powered up.

E.18.6 JOB MONITORING

Three system utilities are provided to help users monitor the execution of their jobs.

1. MONitor. Allows the user to examine variables in a specific job while the job is running, and if desired, change the value of the variable.

2. STAtus. Reports both static information about a job (such as its priority, etc.) and also reports the current status of a job, e.g. whether it is running, queued waiting for a device, idle, or uncompiled.

3. STReam. Allows the user to identify the current status of a general input/output such as which files are open and which jobs have access to those files and whether they are open for input or output.

E.18.7 GLOBAL VARIABLES AND REAL-TIME OPERATION

In many real-time program applications it becomes necessary for jobs to communicate with each other. SWEPSPEED II supports the use of global variables for this purpose. Global variables are accessable to those jobs specified by the system manager. Through the use of global variables data can be passed from one job to another. For example see Fig. E.18.1.

Job A obtains process data from plant transducers. The Retrieved data is placed in global variable G. Job B takes the data in the global variable G and converts it into engineering units.

The above principle is common in multi-tasking systems. However, it is important to recognize that when several jobs are running care must be taken to insure that one job does not write to a global variable at the same time that another job is trying to read the same variable. Steps must be taken to prevent data corruption through the use of flags or job priorities. (Note that an integer or logical can be used as a flag since it takes only one machine instruction to read or write these to a global.) N.B. As the priority number lowers, so does the priority of the job decrease.

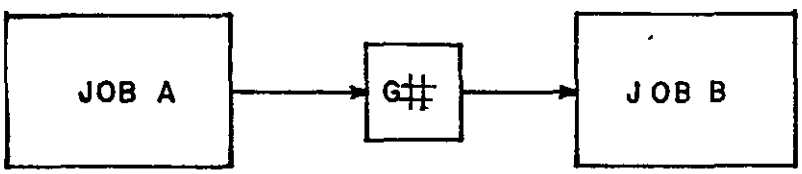


Figure E.18.1 Use of Global Variables

E.18.8 HARDWARE CONFIGURATION

The SWEPSPEED II in the Alarm Handling System is running on a PDP 11/03 supporting several peripheral devices as follows:

Device	Description
DD0:	A dual drive TU58 DECtape backing store.
DD1:	
TT0:	VT100 VDU used as the system console.
TT1:	T43 teletype printer.
TT2:	Chromatics display computer.
TT3:	Host computer if present.

Media Plant Interface System

The Media plant interface allows the user to output or input binary or analogue data to or from plant sensors. This system is accessable through the use of the SWEPSPEED job MEDIA.SPD located on the system tape drive. The use of the system is described elsewhere.

E.18.9 EXAMPLE

First Log in.....

```
<ctl-c>
$ LOG
name? ___<cr>
password? ___<cr>
```

```
<ctl-c>
$EDI3
*10 PRINT"THIS IS A TEST!"
*END

<ctl-c>
$LIS3
Job3
10 PRINT"THIS IS A TEST!"
End of listing

<ctl-c>
$COM3
Job compiled
0 errors
Job space = 888

<ctl-c>
$ACT3
THIS IS A TEST!

<ctl-c>
$SAV3
Save to file? DD1:TEST
done

<ctl-c>
$SCR3

<ctl-c>
$OLD3
From which file?  DD1:TEST
done
```

E.18.10 SWEPSPEED SYSGEN CONFIGURATION

Included in this section is the SWEPSPEED system generation required for the correct operation of the alarm handling system. Refer to the SWEPSPEED system manuals for further details on system generation. The listing presented here is the SWEPSPEED system generation command file required to generate the SWEPSPEED system used to support this alarm handling system software.

```
!SET TT QUIET
SET USR NOSWAP
!
@SYSMAC
!
@ MESAGE
Deleting old files
^Z
!
SET ERROR NONE
DELETE/LOG *.SS2,*.MAP,*.TMP,*.OB*
SET ERROR WARNING
!
@ MESAGE
Creating temporary files
^Z
!TME
!
COPY AMA.MAC+CS111.MAC NL:TPE.TMP
!
@ CREATE
SYSTEM.TMP

!TLE COBRO,5

!NCLUDE REALS,STRINGS,LOGICALS
!NCLUDE ABS,MOD,SIGN
!NCLUDE XOR,NXOR
!NCLUDE FOR,NEXT
!NCLUDE ASSIGN,CLEAR
!NCLUDE OCTAL,CHAR,LEN,VALUE,POS,SUBSTR
!NCLUDE BITS
!NCLUDE INPUT,PRINT
!NCLUDE OPEN,CLOSE
!NCLUDE READ,WRITE
!NCLUDE SQRT
!NCLUDE CONNECT,DISCONNECT,ELAPSE,SLEEP,WAKE
!NCLUDE START,ACTIVATE
!NCLUDE GOSUB,COMPGO
!NCLUDE TIME,DATE,SETTIME,SETDATE
!NCLUDE MEM
!NCLUDE FORMAT
!NCLUDE JOBOVERLAY
!NCLUDE CGOSUB,REM

SYSDVR  DD0
DEVDEV  DD1
```

```
INPSUB 0,BUFLEN=132.
PRISUB 0,BUFLEN=132.
INPSUB 1,BUFLEN=132.
INPSUB 2,BUFLEN=132.,TYPE=NOECHO

TERMINAL TT0,INPSUB=0,PRISUB=0,WIDTH=80.,VDU=1
TERMINAL TT1,INPSUB=1,PRISUB=0,IREG=176520,IVEC=310
TERMINAL TT2,INPSUB=2,PRISUB=0,IREG=176530,IVEC=320
TERMINAL TT3,INPSUB=2,PRISUB=0,IREG=176610,IVEC=330

DEVICE DD,DRIVS=2,BUFFS=3,INPSUB=1,PRISUB=0

FILES=4

JOB NUMBER=1,SIZE=2000,NAME=JOB1,OWNER=GOD,PRIOR1=100,PROTEC=10-
,<PRIVIL=GW,FC,FO,NF,AO,OU,DI,MP,JI,GS,MR,MW>
JOB SIZE=1000,PRIVIL=*

JOB SIZE=1000,PRIVIL=*
JOB SIZE=1000,PRIVIL=*
JOB SIZE=1000,PRIVIL=*
JOB SIZE=500,PRIVIL=*
JOB SIZE=500,PRIVIL=*
JOB SIZE=500,PRIVIL=*
JOB SIZE=500,PRIVIL=*
JOB SIZE=500,PRIVIL=*
JOB SIZE=500,PRIVIL=*
JOB SIZE=200,PRIVIL=*
JOB SIZE=200,PRIVIL=*
JOB SIZE=200,PRIVIL=*
JOB SIZE=100,PRIVIL=*
JOB     ;Jobs with no space allocated
JOB
JOB
JOB
JOB

USER NAME=GOD,PASSWORD=SSD,PRIORITY=32766,PROTECTION=127,PRIVILEGE=*
USER NAME=NE1,PRIORITY=100,PROTECTION=10-
,<PRIVILEGE=GW,FC,FO,NF,AO,OU,DI,MP,JI,GS,MR,MW,SM,GC>
USER
USER
USER

^Z
.
R CREATE
```

```
FILER.TMP
CYCOPT=0
^Z
!
R MACRO
FILER.OBJ=MCTYPE.TMP,FILER.TMP,FILER
^C
!
@FIS
!
*LSIGEN
!
COPY MINERR.LIB MINERR.OBJ
!
RENAME/NOLOG SWSPD2.(SSZ,MAP) LOBRO.*
!
R MESAGE
All done!
^Z
TIME
```

E.19.0 HARDWARE CONFIGURATION

Described in this section is the configuration of the PDP 11/03 computer. This includes details of the hardware setup of inputs/output cards, memory organisation, boot strap location, etc. For operational and further details refer to Digital Equipment Corporation publications.

E.19.1 PDP 11/03 PARTS

The processory, memory, device interface, backplane, and interconnecting hardware are all modular in design. Module selection, such as the type and size of memory and device interfaces, enables custom tailoring to meet specific application requirements. Following is a summary of the modules used in the alarm handling system PDP 11/03.

1) PDP 11/03 LSI11 CPU M7264 with 4K RAM

2) One KEV11, Floating point arthimetic chip

3) One MSV11-C, 16K word MOS Read-Write memory M7955

4) Two MXV11-AA, Dual asynchronous serial line interfaces with 4K word RAM memory and optional ROMs M8047

5) One DLV11, Serial Line Unit M7940

6) One Media Plant Interface Bus extension module

7) One H780-J, Master PDP Power Supply

8) One H9270, Backplane Assembly with logic cabinet

E.19.2 LSI PERIPHERIAL CONFIGURATIONS

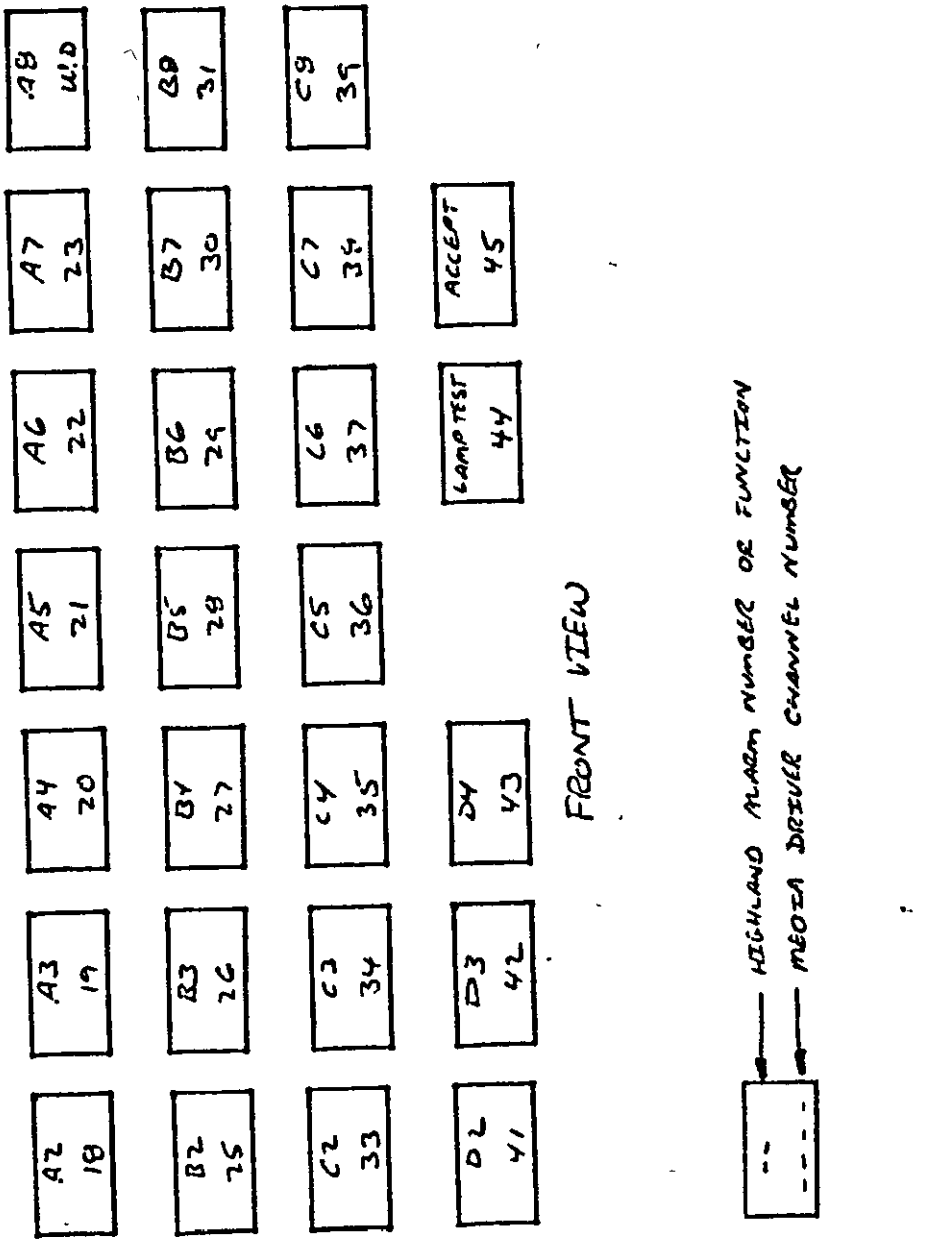
Device	I/O Module	Address	Vector	Function
TT0:	MXV11AA J2	177560	60	VDU console
DD0:/DD1:	MXV11AA J1	176500	300	TU58 Tape Drive
TT1:	MXV11AA J2	176520	310	T43 Printer
TT2:	MXV11AA J1	176530	320	Chromatics Link
TT3:	DLV11	175610	330	Host Link

Device	Tx / Rx Baud Rate	Interface Type
TT0:	9600	RS232
DD0:/DD1:	9600	RS232
TT1:	300	RS232
TT2:	9600	RS232
TT3:	9600	20MA Active Tx, Pasive Rx

All serial communication lines have the following bit pattern:

No parity, 1 stop bit, 8 data bits

E.19.3 HIGHLAND ANNUNCIATOR MEDIA CHANNEL NUMBERS



E.20.0 LISTINGS AND IMPORTANT FLOWCHARTS

```
NAME POW4
10 REM "POWERUP VER 4"
20 ON ERROR GO TO 100
100 REM "R"
110 FOR I=2 TO 20\ STOP I\ NEXT I
120 PROMPT "DAY   = " INPUT D1
130 PROMPT "MONTH = " INPUT D2
140 PROMPT "YEAR  = " INPUT D3
150 PROMPT "HOUR  = " INPUT T1
160 PROMPT "MIN   = " INPUT T2
170 SETDATE D1.D2.D3
180 SETTIME T1.T2.0
190 CONNECT 15 EVERY 1 SECS
200 CLEAR I#(0).0#(0)
210 FOR I=0 TO INT(ZA#(10))\ ZE#(I)=-1.00000E+06\ NEXT I
220 ?F9#=FALSE
230 C#(1)=48\ ?V#(1)=FALSE\ START 10
240 START 4
250 PRINT "LOG IN & ACT14"
END of listing
```

```
NAME DISP4
10 REM "DISPLAY DRIVE VER 1.4"
20 REM "SETUP"
23 START 13
25 ?F5#=TRUE
30 Q=7
40 Q1=2
50 Q2=7
60 M=10
70 M1=5
100 REM "RUN"
105 GOSUB 950\ GOSUB 200\ IF ?Q GOTO 105
110 IF NOT?N# GOTO 115\ $N=$N#\ $N#=""\ ?N#=FALSE\ GOSUB 200
115 IF NOT?M1# GOTO 120\ $N=$M1#\ ?M1#=FALSE\ GOSUB 200
120 ?F5#=FALSE\ STOP
200 REM "SEL FUNC"
210 L1=LEN($N)\ IF L1=0 GOTO 240
215 IF L1=1 AND $N="L" GOTO 990
220 A=VALUE(SUBSTR($N,1,1))
230 GOSUB( 300,350,400,450,500,550)A
240 RETURN
300 REM "AL OFF"
310 ?T=FALSE\ GOTO 370
350 REM "AL ON"
360 ?T=TRUE\ GOSUB 700
370 C=VALUE(SUBSTR($N,2,3))\ $M=$N\ GOSUB 850\ GOSUB 800\ RETURN
400 REM "VT100"
405 $M=SUBSTR($N,2,L1)\ PRINT $M
410 RETURN
450 REM "CHROM"
455 $M=SUBSTR($N,2,L1)\ GOSUB 800\ RETURN
500 REM "TT1:"
505 RETURN
510 OPENOUT @T:"TT1:DUMMY"
520 $M=SUBSTR($N,2,L1)\ PRINT @T:$M
530 CLOSE @T:\ RETURN
550 REM "@"
555 $M=SUBSTR($N,2,L1)\ GOSUB 900\ RETURN
700 REM "BEEP"
710 $M="060500"
720 GOSUB 900\ RETURN
800 REM "CHROMOUT"
810 IF $M<>"" GOTO 820\ RETURN
820 IF NOT?M# GOTO 830\ ELAPSE 2 TICKS \ GOTO 820
830 $M#=$M\ ?M#=TRUE\ RETURN
```

```
850 REM "MEDIA"
855 IF C#(M1)<=0 GOTO 860\ START M\ ELAPSE 2 TICKS \ GOTO 855
860 ?V#(M1)=?T\ C#(M1)=C\ START M\ RETURN
900 REM "OUT Q"
905 IF MOD(I#(Q1)+1,S1#)<>O#(Q1) GOTO 915
910 START Q\ PRINT "DIS Q WAITING!"\ ELAPSE 2 TICKS \ GOTO 905
915 I#(Q1)=MOD(I#(Q1)+1,S1#)\ L=I#(Q1)+S1#*Q1\ $Q#(L)=$M\ START Q\ RETURN
950 REM "IN Q"
955 IF MOD(O#(Q2),S1#)=MOD(I#(Q2),S1#) GOTO 970
960 ?Q=TRUE\ O#(Q2)=MOD(O#(Q2)+1,S1#)\ L=O#(Q2)+S1#*Q2\ $N=$Q#(L)\ RETURN
970 ?Q=FALSE\ $N=""\ RETURN
990 $M="X"\ GOSUB 800\ STOP 13\ OVERLAY .ERR=995:"DD0:LOAD2T" INTO 2,"R"\ STOP
995 PRINT "OVE ER"\ $N=""\ GOTO 10
DIM $N=20, $M=20
DIM $N#=20, $M#=20, $M1#=4
END of listing
```

```
NAME TALK3T
10 REM "11/34 OUT VER 1.3"
20 REM "SETUP"
30 Q2=5
35 ON ERROR GO TO 800
40 OPENOUT @T3:"TT3:DUMMY"
100 REM "RUN"
110 GOSUB 950\ IF $N<>"" GOTO 120\ CLOSE @T3:\ STOP
120 PRINT @T3:$N:
121 PRINT "T"\ GOTO 100
151 $N=SUBSTR($N.1.LEN($N)-1)\ PRINT "TALK =":$N
160 GOTO 100
800 REM "ERROR"
810 CLOSE @T3:\ GOTO 10
950 REM "IN Q"
955 IF MOD(O#(Q2).S1#)=MOD(I#(Q2),S1#  GOTO 970
960 O#(Q2)=MOD(O#(Q2)+1,S1#)\ L=O#(Q2 +S1#*Q2\ $N=$Q#(L)\ RETURN
970 $N=""\ RETURN
DIM $N=20
END of listing
```

```
NAME KBDRI6
10 REM "KEYBOARD DRIVER VER 1.6"
20 REM "SET UP"
30 M=10
40 M1=2
50 Q=7
60 Q1=1
70 Q2=6
100 REM "SCAN KEYS"
105 I=145
110 C#(M1)=I
115 START M
120 IF C#(M1)>0 GOTO 115
125 IF V#(M1)=0 GOTO 150
130 V=V#(M1)
135 FOR J=0 TO 11
140 IF BIT(J,V) GOTO 250
145 NEXT J
150 FOR I=161 TO 185 STEP 8
155 C#(M1)=I
160 START M
165 IF C#(M1)>0 GOTO 160
170 IF V#(M1)=0 GOTO 195
175 V=V#(M1)
180 FOR J=0 TO 7
185 IF BIT(J,V) GOTO 250
190 NEXT J
195 NEXT I
200 REM "POLL LAPSE"
210 ELAPSE 5 TICKS
215 GOSUB 950\ IF $N="" GOTO 220\ K=0\ $F#(0)=$N\ GOSUB 400
220 GOTO 100
250 REM "WITCH KEY"
255 C=I+J
260 IF C>=145 AND C<=154 GOTO 280
265 IF C>=161 AND C<=189 GOTO 290
270 K=0\ REM "NO KEY"
275 GOTO 100
280 K=C-144
285 GOTO 300
290 K=C-160+10
300 REM "KEY CONTROL"
305 GOSUB 350
315 IF NOT?M GOTO 330
```

```
320 GOSUB 575
325 GOTO 100
330 GOSUB 400
335 GOTO 100
350 REM "CK MASK"
355 IF ?M(K) GOTO 370
360 ?M=FALSE
365 RETURN
370 ?M=TRUE
375 RETURN
400 REM "FUNCTION"
405 IF $F#(K)<>"" GOTO 415
410 RETURN
415 P=1
420 GOSUB 780\ P=P+2\ L1=LEN($F#(K))\ IF P>L1 OR T<=0 GOTO 410
430 GOSUB( 500.510.530.550.570.590.625.645.700.720,790.790.790.790)T
440 P=P+2\ IF P>L1 GOTO 410
445 GOTO 420
500 REM "MASK"
505 GOSUB 780\ ?M(T)=TRUE\ RETURN
510 REM "UNMASK"
515 GOSUB 780\ ?M(T)=FALSE\ RETURN
530 REM "ON"
535 GOSUB 780\ IF T>=49 AND T<=60 GOTO 540\ RETURN
540 S=T-48\ ?S(S)=TRUE
545 ?V#(M1)=TRUE\ C#(M1)=T\ START M\ RETURN
550 REM "OFF"
555 GOSUB 780\ IF T>=49 AND T<=60 GOTO 560\ RETURN
560 S=T-48\ ?S(S)=FALSE
565 ?V#(M1)=FALSE\ C#(M1)=T\ START M\ RETURN
570 REM "BEEP"
572 GOSUB 780
575 IF NOT?S(11) GOTO 585
580 RETURN
585 X=0\ GOSUB 600\ ELAPSE 1 TICKS \ GOSUB 610\ RETURN
590 REM "CLEAR"
595 GOSUB 780\ GOTO( 596.597.598)T\ RETURN
596 CLEAR ?M(0)\ RETURN
597 CLEAR ?S(0)\ FOR T=48 TO 60\ GOSUB 565\ NEXT T\ RETURN
598 GOSUB 596\ GOTO 597
600 REM "BEEP ON"
605 ?V#(M1)=TRUE\ C#(M1)=59+X\ START M\ RETURN
610 REM "BEEP OFF"
615 ?V#(M1)=FALSE\ C#(M1)=59+X\ START M\ RETURN
625 REM "SEND"
626 GOSUB 780\ GOSUB( 632.633.634.635.640,641.642.643)T\ RETURN
630 GOSUB 900\ $B1=""\ RETURN
632 $M=$T#(K)+CHAR(13)\ RETURN 630
633 $M=$T#(K)+CHAR(27)\ RETURN 630
634 $M=$T#(K)\ RETURN 630
635 L2=LEN($T#(K))\ IF L2<3 GOTO 638\ IF B<>0 GOTO 636\ $B1=$T#(K)\ GOTO 637
```

```
636 $B1=SUBSTR($T#(K),3,L2)
637 $B=$B+$B1\ B=B+1\ PRINT $B1;\ RETURN
638 RETURN
640 $M=$B+CHAR(13)\ $B=""\ B=0\ RETURN 630
641 $M=$B+CHAR(27)\ $B=""\ B=0\ RETURN 630
642 $M=$B\ $B=""\ B=0\ RETURN 630
643 $X=CHAR(6)\ IF B>=1 GOTO 644\ $B=""\ B=0\ PRINT $X;\ RETURN
644 B=B-1\ $B=SUBSTR($B,1,LEN($B)-1)\ PRINT $X;\ RETURN
645 REM "HOLD ON"
650 GOSUB 780\ IF T>=46 AND T<=60 GOTO 655\ RETURN
655 IF T>=48 GOTO 660\ GOSUB 545\ GOTO 665
660 GOSUB 540
665 C#(M1)=C
670 START M\ ELAPSE 2 TICKS \ IF C#(M1)>0 GOTO 670\ IF ?V#(M1) GOTO 665
675 IF T>=48 GOTO 680\ GOSUB 565\ RETURN
680 GOSUB 560\ RETURN
700 REM "TOGGLE"
705 GOSUB 780\ IF T>=49 AND T<=60 GOTO 710\ RETURN
710 S=T-48\ IF NOT?S(S) GOTO 715\ GOSUB 560\ ELAPSE 10 TICKS \ RETURN
715 GOSUB 540\ ELAPSE 10 TICKS \ RETURN
720 REM "SET"
725 GOSUB 780\ GOTO( 730,731,732,733)T\ RETURN
730 ASSIGN ?M(11)=T,T,T,T,T\ RETURN
731 ASSIGN ?M(11)=F,F,F,F,F\ RETURN
732 FOR T=49 TO 58\ GOSUB 545\ NEXT T\ RETURN
733 FOR T=49 TO 58\ S=T-48\ IF NOT?S(S) GOTO 734\ NEXT T\ RETURN
734 GOSUB 565\ NEXT T\ RETURN
780 REM "GET FUNC"
785 $T=SUBSTR($F#(K),P,P+1)\ T=VALUE($T)\ RETURN
790 REM "END"
795 RETURN 410
900 REM "OUT Q"
905 IF MOD(I#(Q1)+1,S1#)<>O#(Q1) GOTO 915
910 START Q\ PRINT "KB Q WAITING"\ ELAPSE 2 TICKS \ GOTO 905
915 I#(Q1)=MOD(I#(Q1)+1,S1#)\ L=I#(Q1)+S1#*Q1\ $Q#(L)=$M\ START Q\ RETURN
950 REM "IN Q"
955 IF MOD(O#(Q2),S1#)=MOD(I#(Q2),S1#) GOTO 970
960 O#(Q2)=MOD(O#(Q2)+1,S1#)\ L=O#(Q2)+S1#*Q2\ $N=$Q#(L)\ RETURN
970 $N=""\ RETURN
DIM ?M(40), ?S(12)
DIM $N=20, $M=20, $T=2, $B1=4, $B=20, $X=1
DIM $F#(40)=20, $T#(40)=20
END of listing
```

```
NAME EP6
10 REM "EVENT PROC VER 1.6"
20 REM "SETUP"
30 ?F2#=TRUE
40 H(0)=^034614\ H(1)=^000143
50 A=6
60 ?X=FALSE
100 REM "FIND CHANGES"
105 IF ?D# GOTO 110\ IF ?X GOTO 107\ ?F2#=FALSE\ STOP
107 ?E#=TRUE\ ?F2#=FALSE\ IF ?F9# GOTO 108\ STOP
108 START A\ STOP
110 ?D#=FALSE
120 FOR I=0 TO INT(%A#(9))
125 IF D1#(I)=0 GOTO 150
130 FOR B=0 TO 15
135 IF NOTBIT(B,D1#(I)) GOTO 140\ E=16*I+B\ BIT( B,D1#(I))=FALSE\ GOSUB 200
140 NEXT B
150 NEXT I
155 GOTO 105
200 REM "CHANGE"
210 L=INT(%A#(7))
220 FOR J1=1 TO INT(%A#(10))
230 D1=L+1\ J=J1-1
240 IF INT(%A#(D1))<>E GOTO 250\ GOSUB 300
250 L=INT(%A#(L))+L\ NEXT J1\ RETURN
300 REM "EX CHANGE"
310 D=L\ D2=D+2\ D3=D+3\ D4=D+4\ D5=D+5\ D6=D+6
315 GOSUB( 325,340,400,400,400,400,500,600)INT(%A#(D2))
320 RETURN
325 REM "BINARY"
330 ?E=LOGIC(INT(%D#(E)))
335 IF ?E XOR BIT(J,E#(0)) GOTO 338\ RETURN
338 GOSUB 800\ RETURN
340 REM "BINARY INV"
345 ?E=LOGIC(INT(%D#(E)))
350 IF NOT?E XOR BIT(J,E#(0)) GOTO 355\ RETURN
355 ?E=NOT?E\ GOSUB 800\ RETURN
400 REM "ANA"
410 N=1\ N1=D3
420 IF %A#(N1)>%D#(E) GOTO 440
430 N1=N1+1\ N=N+1\ IF N<>5 GOTO 420
440 GOSUB 900
455 IF ?E NXOR BIT(J,E#(0)) GOTO 460\ GOSUB 800
460 RETURN
500 REM "TREND"
```

```
510 N=1\ N1=D3
520 IF %A#(N1)>%D2#(E) GOTO 540
530 N1=N1+1\ N=N+1\ IF N<>5 GOTO 520
540 GOSUB 900
550 IF ?E NXOR BIT(J.E#(0)) GOTO 560\ GOSUB 800
560 RETURN
600 REM "TIME OUT"
610 RETURN
800 REM "E CHANGE"
801 PRINT "H":N,"L0":L0,?E,"J":J
805 ?X=TRUE\ TIME T(0)
810 %T=60.0*FLOAT(T(0))+FLOAT(60*T(1)+T(2))
815 IF ?E AND NOTBIT(J.E#(0)) GOTO 825
820 BIT( J.E#(0))=FALSE\ %E#(J)=-1.00000E+06\ GOTO 830
825 BIT( J.E#(0))=TRUE\ %E#(J)=%T
830 RETURN
900 REM "HYSTERESIS SUB"
908 IF E1#(J)<>0 GOTO 910\ E1#(J)=N
910 IF E1#(J)=N GOTO 980
920 L0=(E1#(J)-1)*5+N-1
950 IF BIT(L0,H(0)) GOTO 960\ ?E=FALSE\ GOTO 970
960 ?E=TRUE
970 E1#(J)=N\ RETURN
980 ?E=BIT(J.E#(0))\ RETURN
DIM H(1), T(2)
END of listing
```

```
NAME AG6
1 PRINT "AG"
10 REM "ALARM GENERATOR VER 1.6"
20 REM "SETUP"
30 ?F3#=TRUE
40 B=2
100 REM "RUN AG"
105 S=INT(ZA#(11))
106 IF S=0 GOTO 160
110 D=INT(ZA#(8))
115 FOR I=0 TO S-1
120 D1=D+1\ D2=D+2\ D3=D+3
125 Z=INT(ZA#(D2))
130 Z1=D2+Z+1\ Z1=INT(ZA#(Z1))
135 IF BIT(I,G#(0)) GOTO 140\ GOSUB 200\ GOTO 145
140 GOSUB 300
145 GOSUB 500
150 D=Z+Z1+4+D
155 NEXT I
160 ?F3#=FALSE
165 STOP
200 REM "CK FOR ON"
205 IF Z<>0 GOTO 210\ RETURN
210 FOR K=1 TO Z\ J=D2+1
220 GOSUB 400
230 NEXT K
240 RETURN
300 REM "CK FOR OFF"
305 IF Z1<>0 GOTO 310\ RETURN
310 FOR K=1 TO Z1\ J=Z0+1
320 GOSUB 400
330 NEXT K
340 RETURN
400 REM "PROCESS EXP"
405 P=0
410 R=INT(ZA#(J))
415 IF R<0 GOTO 420\ P=P+1\ ?S(P)=BIT(R,E#(0))\ GOTO 425
420 GOSUB 600
425 RETURN
500 REM "CK RESULT"
505 PRINT "CK AG"
510 IF BIT(I,G#(0)) NXOR ?S(P) GOTO 540
520 ?G#=TRUE
530 BIT( I,G#(0))=?S(P)
535 GOSUB 700
```

```
540 RETURN
600 REM "BOOL OP"
610 R=ABS(R)\ P1=P-1
620 GOTO( 630,640,650,660)R
630 ?S(P)=NOT?S(P)\ RETURN
640 ?S(P1)=?S(P1) OR ?S(P)\ P=P1\ RETURN
650 ?S(P1)=?S(P1) AND ?S(P)\ P=P1\ RETURN
660 ?S(P1)=?S(P1) XOR ?S(P)\ P=P1\ RETURN
700 REM "OUT"
705 PRINT ?S(P)
710 IF ?S(P) GOTO 720\ $A="1"\ GOTO 730
720 $A="2"
730 IF NOT?M1# GOTO 740\ START B\ ELAPSE 2 TICKS \ GOTO 730
740 $M1#=$A+DECIMAL(INT(%A#(D)))\ ?M1#=TRUE\ START B\ RETURN
DIM ?S(25)
DIM $A=1
END of listing
```

```
NAME QMAN4
10 REM "Q MANAGER VER 1.4"
20 REM "SETUP"
30 Z=4
40 S1#=7
50 ASSIGN J(5)=3.4.2.8.17
100 REM "SCAN OUT Q'S"
110 ?T=FALSE
120 FOR Q=0 TO Z
130 IF MOD(O#(Q).S1#)=MOD(I#(Q).S1#) GOTO 150
140 GOSUB 300
150 NEXT Q
160 IF ?T GOTO 110
170 STOP
300 REM "GET DATA & DEV NO"
305 ?T=TRUE
310 O#(Q)=MOD(O#(Q)+1.S1#)
320 L=O#(Q)+S1#*Q
330 $N=$Q#(L)\ S=LEN($N)\ IF S>1 GOTO 340\ RETURN
340 $A=SUBSTR($N.1.2)
350 Q1=VALUE($A)\ IF Q1>9 GOTO 355\ $N=SUBSTR($N.3.S)\ GOTO 400
355 Q1=5
400 REM "XFER DATA"
405 IF Q1>=5 GOTO 410\ RETURN
410 IF MOD(I#(Q1)+1.S1#)<>O#(Q1) GOTO 500
420 REM "Q FULL"
430 START J(Q1)
440 PRINT "Q-";Q1;"WAITING!"
450 ELAPSE 20 TICKS
460 GOTO 400
500 REM "Q NOT FULL"
510 I#(Q1)=MOD(I#(Q1)+1,S1#)
520 L=I#(Q1)+S1#*Q1
530 $Q#(L)=$N
540 IF J(Q1)<>8 GOTO 545\ WAKE J(Q1)\ GOTO 550
545 START J(Q1)
550 RETURN
DIM J(20)
DIM $N=20. $A=2
DIM I#(10). O#(10)
DIM $Q#(70)=20
END of listing
```

```
NAME DA1B
10 REM "DATA ACQ VER 1.8"
20 REM "SETUP"
25 ?F4#=TRUE
30 M=10
40 M1=3
50 Q=7
60 Q1=3
70 Q2=8
80 E1=5
99 POINT=0
100 REM "RUN DA"
105 IF P0#=0 GOTO 170
110 FOR I=1 TO P0#
111 POINT=POINT+1\ PRINT POINT
115 D=12+(I-1)*6\ C=0
120 D1=D+1\ D2=D+2\ D3=D+3\ D4=D+4\ D5=D+5
125 ?F=FALSE
130 X=INT(%A#(D))
135 REM "ASK DEVICE"
140 GOSUB( 200,250,300,350)INT(%A#(D1))+1
145 IF C<0 GOTO 160
150 REM "TYPE"
155 GOSUB( 600,610,630,650)INT(%A#(D2))+1
160 NEXT I
165 IF NOT?F GOTO 170\ ?D#=TRUE\ START E1
170 ?F4#=FALSE
171 GOTO 100
180 STOP
200 REM "MEDIA"
210 C#(M1)=X
215 START M\ IF C#(M1)>0 GOTO 215
220 C=C#(M1)\ ?D=?V#(M1)\ %D=%V#(M1)
225 RETURN
250 REM "11/34"
255 $L="10"
260 GOSUB 500
265 GOSUB 950\ GOSUB 700\ RETURN
300 REM "TT1:"
305 $L="04"
310 GOSUB 500
320 GOSUB 950\ GOSUB 700\ RETURN
350 REM "EMPTY"
355 C=-1\ RETURN
500 REM "OUT CODE"
```

```
505 $T=DECIMAL(INT(ZA#(D)))\ T=LEN($T)
515 $M=$L+$T+CHAR(13)
520 GOSUB 900\ RETURN
600 REM "BINARY"
605 GOSUB 800\ RETURN
610 REM "BINARY INV"
615 ?D=NOT?D\ GOSUB 800\ RETURN
630 REM "ANA 1"
635 GOSUB 750\ RETURN
650 REM "ANA 2"
655 GOSUB 750\ RETURN
700 REM "ZD.?D"
705 $D="."\ N=LEN($N)\ IF N<>0 GOTO 707\ C=-1\ RETURN
707 N1=POS($D.$N)\ IF N1<>0 GOTO 710\ ?D=LOGIC(VALUE(SUBSTR($N.N.N)))\ RETURN
710 ZD=FLOAT(VALUE(SUBSTR($N.N-3.N)))*1.00000E-04\ RETURN
750 REM "CK RANGE"
755 IF ZD>ZA#(D4) OR ZD<ZA#(D3) GOTO 765
760 GOSUB 850\ RETURN
765 PRINT "RANGE CK";X\ RETURN
800 REM "UP DATE DIG D"
805 N=I-1
810 IF LOGIC(INT(ZD#(N))) NXOR ?D GOTO 815\ ZD#(N)=FLOAT(INT(?D))\ BIT( N,D1#(0)
815 RETURN
850 REM "UPDATE AN D"
855 N=I-1
860 IF ABS(ZD-ZD#(N))<ZA#(D5) GOTO 865\ ?F=TRUE\ BIT( N.D1#(0))=TRUE
865 ZD1#(N)=ZD#(N)\ ZD#(N)=ZD
870 ZD2#(N)=(ZD-ZD#(N)+ZD2#(N))/2.0
875 RETURN
900 REM "OUT Q"
905 IF MOD(I#(Q1)+1.S1#)<>O#(Q1) GOTO 915
910 START Q\ PRINT "DA1 Q WAITING!"\ ELAPSE 2 TICKS \ GOTO 905
915 I#(Q1)=MOD(I#(Q1)+1.S1#)\ L=I#(Q1)+S1#*Q1\ $Q#(L)=$M\ START Q\ RETURN
950 REM "IN Q"
955 IF MOD(O#(Q2).S1#)=MOD(I#(Q2).S1#) GOTO 970
960 O#(Q2)=MOD(O#(Q2)+1.S1#)\ L=O#(Q2)+S1#*Q2\ $N=$Q#(L)\ RETURN
970 SLEEP \ GOTO 955
DIM $L=2, $T=3, $M=20, $D=1, $N=20, $A=8, $B=8
END of listing
```

```
NAME DACON2
10 REM "DA CONTROLLER VER 1.2"
20 REM "SETUP"
30 D=8
100 REM "SELECT SCAN GROUP"
110 A=0
120 TIME T(0)
130 T=T(2)
140 IF MOD(T,60)=0 GOTO 180
150 IF MOD(T,15)=0 GOTO 190
160 IF MOD(T,5)=0 GOTO 200
170 S=1\ GOTO 300
180 S=4\ ?F9#=TRUE\ GOTO 300
190 S=3\ GOTO 300
200 S=2\ GOTO 300
300 REM "SET MAX ADDR"
310 FOR I=3 TO S+2
320 A=INT(%A#(I))+A
330 NEXT I
340 P0#=A
350 START D
360 ELAPSE 30 TICKS
370 TIME T(0)
380 IF NOT?F4# AND T<>T(2) GOTO 100\ ELAPSE 2 TICKS \ GOTO 370
DIM T(2)
END of listing
```

```
NAME LISN2T
10 REM "11/34 IN VER 1.2"
20 REM "SETUP"
30 Q=7
40 Q1=0
50 ON ERROR GO TO 800
60 OPENIN @L3:"TT3:DUMMY"
100 REM "LISTEN LOOP"
110 INPUT @L3:$M
111 PRINT "L"
120 IF $M="" GOTO 100
130 GOSUB 900
135 $M=""
140 GOTO 100
800 REM "ERROR"
810 CLOSE @L3:\ GOTO 60
900 REM "OUT Q"
905 IF MOD(I#(Q1)+1,S1#)<>O#(Q1) GOTO 915
910 START Q\ PRINT "LIS Q WAITING"\ ELAPSE 2 TICKS \ GOTO 905
915 I#(Q1)=MOD(I#(Q1)+1,S1#)\ L=I#(Q1)+S1#*Q1\ $Q#(L)=$M\ START Q\ RETURN
DIM $M=20
END of listing
```

```
NAME MEDIA6
5 REM "MEDIA DRIVER VER 1.6"
10 REM "FIND DEV"
15 ?F1#=TRUE
20 FOR I=1 TO 5
25 ?E=FALSE
30 IF C#(I)<=0 GOTO 45
35 C=C#(I)\ V=V#(I)\ %V=%V#(I)\ ?V=?V#(I)
40 GOSUB 100
45 NEXT I
50 ?F1#=FALSE
55 STOP
100 REM "SELECT I/O"
105 IF C=144 GOTO 550
110 IF C<16 GOTO 200
115 IF C=16 GOTO 300
120 IF C<81 GOTO 400
125 IF C<97 GOTO 160
130 IF C<129 GOTO 500
135 IF C<144 GOTO 160
140 IF C<161 GOTO 600
145 IF C<193 GOTO 700
160 PRINT "NON-EXISTENT MEDIA I/O ERROR"\ PRINT \ C#(I)=-1\ RETURN
165 ?V#(I)=?V\ %V#(I)=%V\ V#(I)=V
170 IF NOT?E GOTO 180
175 C#(I)=-2\ RETURN
180 C#(I)=0\ RETURN
200 REM "AOV&AOI"
205 IF %V<0.0 OR %V>1.0 GOTO 215
210 V=INT(FLOAT('001777)*%V)
212 V2=0\ FOR J=0 TO 9\ J6=J+6\ BIT( J6,V2)=BIT(J,V1)\ NEXT J
213 MEM('160000+C*2)=V2\ GOTO 165
215 PRINT "MEDIA ERROR A0=";%V\ ?E=TRUE\ GOTO 165
300 REM "WD"
305 V=MEM('160036)\ GOTO 165
400 REM "DO"
405 C=C-17
410 BIT( C,A(0))=?V
415 FOR J=0 TO 3\ MEM('160040+J*2)=A(J)\ NEXT J
420 GOTO 165
500 REM "AI"
505 C=C-97\ ?V=FALSE\ V=0
510 IF C>15 GOTO 520
515 MEM('160052)=C\ ELAPSE 3 TICKS \ V1=MEM('160054)\ GOTO 560
520 C=C-16
```

```
525 MEM('160056)=C\ ELAPSE 3 TICKS \ V1=MEM('160060)\ GOTO 560
550 V1=MEM('160060)
560 V=0
570 FOR J=0 TO 9\ J6=J+6\ BIT( J,V)=BIT(J6,V1)\ NEXT J
580 ZV=FLOAT(V)*9.76563E-04
590 GOTO 165
600 REM "DIM"
605 C=C-145\ ZV=0.0
610 V=MEM('160066)\ FOR Z=0 TO 15\ BIT( Z,V)=NOTBIT(Z,V)\ NEXT Z\ ?V=BIT(C,V)\ GOTO
700 REM "DIF"
705 C=C-161\ ZV=0.0
710 GOTO( 715,720,725,730)(C/8)+1
715 V=MEM('160070)\ GOTO 735
720 V=MEM('160072)\ C=C-8\ GOTO 735
725 V=MEM('160074)\ C=C-16\ GOTO 735
730 V=MEM('160076)\ C=C-24
735 W=(V-'000377)\ V=0\ FOR Z=0 TO 7\ Z1=15-Z\ BIT( Z,V)=BIT(Z1,W)\ NEXT Z\ ?V=BIT(C
740 GOTO 165
DIM A(3)
DIM C#(5), V#(5)
DIM ?V#(5)
DIM ZV#(5)
END of listing
```

```
NAME HELTH1
10 REM "HEALTH CHECK VER 1.1"
15 REM ""
20 TIME T(0)
30 OPENOUT @T:"TT1:DUMMY"
40 PRINT @T:"TIME =";T(0);":";T(1);":";T(2)
50 CLOSE @T:
60 ?V#(5)=TRUE\ GOSUB 65\ GOTO 100
65 FOR I=59 TO 59
70 C#(5)=I
80 START 10\ ELAPSE 2 TICKS \ IF C#(5)>0 GOTO 80
90 NEXT I
95 RETURN
100 ELAPSE 30 TICKS
110 ?V#(5)=FALSE
120 GOSUB 65
DIM T(2)
END of listing
```

```
NAME CHRON4
10 REM "CHRONATICS LINK VER 1.4"
20 REM "SETUP"
30 D=2
40 ON ERROR GO TO 300
50 CLOSE @D1:\ CLOSE @D2:
100 REM "RUN"
105 OPENIN @D1:"TT2:DUMMY"
110 OPENOUT @D2:"TT2:DUMMY"
200 INPUT @D1:$N
210 IF $N="*" OR $N="" GOTO 240
220 IF NOT?N# GOTO 230\ START D\ ELAPSE 2 TICKS \ GOTO 220
230 $N#=$N\ ?N#=TRUE\ START D
240 IF ?M# GOTO 250\ PRINT @D2:""\ GOTO 200
250 $M=$M#\ PRINT @D2:$M#\ $M#=""\ ?M#=FALSE\ GOTO 200
300 REM ""
310 CLOSE @D1:\ CLOSE @D2:\ IF $M="X" GOTO 320\ GOTO 100
320 STOP
DIM $N=20. $M=1
END of listing
```

```
NAME COMAH2
10 REM "AHS COMMANDS VER 1.2"
20 REM "SETUP"
30 ASSIGN $C(0)="ED","TI","RE","ST","RU","X"
100 REM "RUN"
105 C=0\ PROMPT "##" INPUT $N
110 FOR I=0 TO 5\ IF $N=$C(I) GOTO 120\ NEXT I\ GOTO 130
120 C=I+1\ NEXT I
130 GOSUB( 200,250,300,350,400,450)C
140 GOTO 100
200 REM "ED"
210 STOP 7.2.12
220 OVERLAY .ERR=800:"DD0:EDIT" INTO 2,"R"
230 STOP
250 REM "TI"
260 TIME T(0)\ DATE D(0)\ PRINT D(0):"/":D(1):"/":D(2),T(0):":":T(1):":":T(2)\ RETURN
300 REM "RE"
310 START 1\ STOP
350 REM "ST"
360 STOP 12.8.9\ RETURN
400 REM "RU"
410 START 12.9.8\ RETURN
450 REM "X"
460 STOP
800 PRINT "OVERLAY ERROR"\ GOTO 100
DIM T(2), D(2)
DIM $C(5)=2, $N=2
END of listing
```

```
NAME SETUP
DIM E#(20), E1#(100), G#(20), D1#(30)
DIM ZA#(260), ZE#(100), ZD#(100), ZD1#(100), ZD2#(100)
END of listing
```

```
NAME PCPMC2
10 REM "PCPMC VER 1.2"
20 REM "S"
30 Q1=4
40 Q2=9
50 M=10
55 M1=1
60 Q=7
70 ?F6#=TRUE
100 REM "R"
105 GOSUB 950\ IF NOT?Q GOTO 110\ GOSUB 200
107 GOSUB 900\ IF ?Q GOTO 105
110 ?F6#=FALSE\ STOP
200 REM ""
210 C=VALUE(SUBSTR($N,1,3))\ IF C<>0 GOTO 220\ RETURN
220 C#(M1)=C
230 START M\ IF C#(M1)>0 GOTO 230
235 V=V#(M1)
240 ZV=ZV#(M1)*10000.0\ $V=DECIMAL(INT(ZV))
245 IF LEN($V)>=4 GOTO 250\ $V="0"+$V\ GOTO 245
250 $M="12"+$N+"0."+$V+DECIMAL(V)+CHAR(13)
260 RETURN
900 REM "QQ"
905 IF MOD(I#(Q1)+1,S1#)<>O#(Q1) GOTO 915
910 START Q\ PRINT "P Q"\ ELAPSE 2 TICKS \ GOTO 905
915 I#(Q1)=MOD(I#(Q1)+1,S1#)\ L=I#(Q1)+S1#*Q1\ $Q#(L)=$M\ START Q\ RETURN
950 REM "IQ"
955 IF MOD(O#(Q2),S1#)=MOD(I#(Q2),S1#) GOTO 970
960 ?Q=TRUE\ O#(Q2)=MOD(O#(Q2)+1,S1#)\ L=O#(Q2)+S1#*Q2\ $N=$Q#(L)\ RETURN
970 ?Q=FALSE\ $N=""\ RETURN
DIM $N=20, $M=20, $V=4
END of listing
```

```
NAME WD3
10 REM "SYS WATCH DOG DRIV"
20 MEM('160036)='110000
30 TIME T(0)\ IF T(0)+T(1)+T(2)=0 GOTO 40\ STOP
40 FOR I=0 TO INT(ZA#(10))\ IF ZE#(I)<=-1.00000E+06 GOTO 50\ ZE#(I)=ZE#(I)-86400.0
50 NEXT I
DIM T(2)
END of listing
```

```
NAME EDIT2
10 REM "EDIT ADB VER 1.2"
20 REM "S"
30 ASSIGN $C(1)="I","R","D","X","L","P"
40 ON ERROR GO TO 900
100 REM ""
110 PROMPT "> " INPUT $F\ L=LEN($F)\ IF L=0 GOTO 100
120 $X=SUBSTR($F,1,1)\ FOR I=1 TO 6\ IF $X=$C(I) GOTO 200\ NEXT I
140 GOTO 100
200 IF L<2 GOTO 210\ A=INT(SUBSTR($F,1,L))\ IF A>0 GOTO 220
210 IF I>3 GOTO 230\ PROMPT "ARRAY ELEMENT NO = " INPUT $F\ A=VALUE($F)
220 IF A<=0 OR A>INT(%A#(1))+1 GOTO 100
230 GOTO( 300,350,400,450,500,550)I
240 GOTO 100
300 REM "IN"
305 IF A=1 GOTO 100\ GOSUB 700
310 GOSUB 750\ FOR J=INT(%A#(1))+1 TO A STEP -1\ J1=J-1\ %A#(J)=%A#(J1)\ NEXT J
315 %A#(A)=%B\ %A#(1)=%A#(1)+1.0\ A=A+1\ GOTO 300
350 REM "RE"
355 GOSUB 700\ IF $F<>"-" GOTO 360\ A=A-1\ IF A<=0 GOTO 100\ GOTO 355
360 IF $F<>"+" GOTO 365\ A=A+1\ IF A>INT(%A#(1)) GOTO 100\ GOTO 355
365 GOSUB 750\ %A#(A)=%B\ GOTO 350
400 REM "DE"
405 PRINT "DELETE ";\ GOSUB 700\ IF $F="Y" GOTO 410\ GOTO 100
410 IF %A#(1)=0.0 GOTO 100\ FOR J=A TO INT(%A#(1))\ J1=J+1\ %A#(J)=%A#(J1)\ NEXT J
415 %A#(1)=%A#(1)-1.0\ GOTO 100
450 REM ""
455 OVERLAY .ERR=100:"DD0:DISP2" INTO 2
500 REM ""
505 FOR J=1 TO INT(%A#(1))\ PRINT J;" => ";%A#(J)\ NEXT J\.PRINT \ GOTO 100
550 REM ""
555 TIME T(0)\ DATE D(0)\ OPENOUT @E:"TT1:DUMMY"
560 PRINT @E:D(0):"-":D(1):"-":D(2),T(0):":":T(1):":":T(2)
565 PRINT @E:\ FOR J=1 TO INT(%A#(1))\ PRINT @E:%A#(J);"  ";\ NEXT J\ PRINT @E:\ CLOSE
700 REM ""
705 PRINT A;"=>";%A#(A),\ INPUT $F\ IF $F="" GOTO 710\ RETURN
710 RETURN 100
750 REM ""
755 PRINT ,"= ";\ INPUT %B\ RETURN
900 REM "ER"
905 E=ERROR\ IF E=12 GOTO 910\ GOTO 450
910 PRINT "REDIM %A#()"\ GOTO 100
DIM T(2), D(2)
DIM $C(6)=1, $F=5, $X=1
END of listing
```

```
NAME LOAD2T
10 REM "ADB XFER VER 1.2"
20 REM "SETUP"
25 STOP 13
30 OPENOUT @C1:"TT2:DUMMY"
40 OPENIN @C2:"TT2:DUMMY"
100 REM "RUN"
110 PRINT @C1:"*"\ INPUT @C2:$N\ IF $N="READY" GOTO 120\ ELAPSE 20 TICKS \ GOTO 110
120 ON ERROR GO TO 850\ PRINT @C1:"0"\ INPUT @C2:%N\ N=INT(%N)
130 ON ERROR GO TO 800\ %A#(N)=0.0\ %A#(1)=%N
140 ON ERROR GO TO 850
150 FOR I=2 TO N\ PRINT @C1:"0"\ INPUT @C2:%N\ %A#(I)=%N\ NEXT I
700 REM "X"
705 CLOSE @C1:\ CLOSE @C2:
710 OVERLAY .ERR=720:"DD0:DISP2" INTO 2,"R"
720 PRINT "OVERLAY ERROR DISP"\ STOP
800 REM "ER1"
810 E=ERROR\ IF E<>12 GOTO 700\ PRINT "REDIM %A#"\ PRINT @C1:"1"\ GOTO 700
850 REM "ER2"
860 E=ERROR\ IF E<>1 GOTO 700\ PRINT "XFER ERROR"\ PRINT @C1:"2"\ GOTO 700
DIM $N=20
END of listing
```

```
$STR
Job   Name   Stream   Device   Filename   Status
2     DISP4  @T                          Closed
3     TALK3T @T3                         Closed
11    HELTH1 @T                          Closed
12    LISN2T @L3                         Closed
13    CHROM4 @D1                         Closed
13    CHROM4 @D2                         Closed
19    TEST   @T                          Closed
```

```
$GLO
?LIS
Name              Referenced by jobs
I#(10)            1.2.3.4,7.8.12.17
O#(10)            1.2.3.4.7.8.12.17
%A#(260)          1.5.6.8.9.15.16
%E#(100)          1.5.15.16
?F5#              2
?N#               2.13
$N#=20            2.13
?M1#              2.6
$M1#=4            2.6
?M#               2,13
$M#=20            2,13
C#(5)             1.2.4.8,10.11,17
?V#(5)            1.2,4.8,10,11
S1#               2.3.4.7,8,12,17
$Q#(70)=20        2.3,4.7.8,12,17
V#(5)             4,10.17
$F#(40)=20        4
$T#(40)=20        4
?F2#              5
?D#               5,8
?E#               5
D1#(30)           5.8,16
%D#(100)          5,8.16
E#(20)            5.6.16
%D2#(100)         5,8.16
E1#(100)          5,16
?F3#              6
G#(20)            6,16
?G#               6
?F4#              8,9
P0#               8,9
%V#(5)            8,10,17
%D1#(100)         8.16
?F1#              10
%Z#               18
?F6#              17
?F9#              1.5,9
?
```

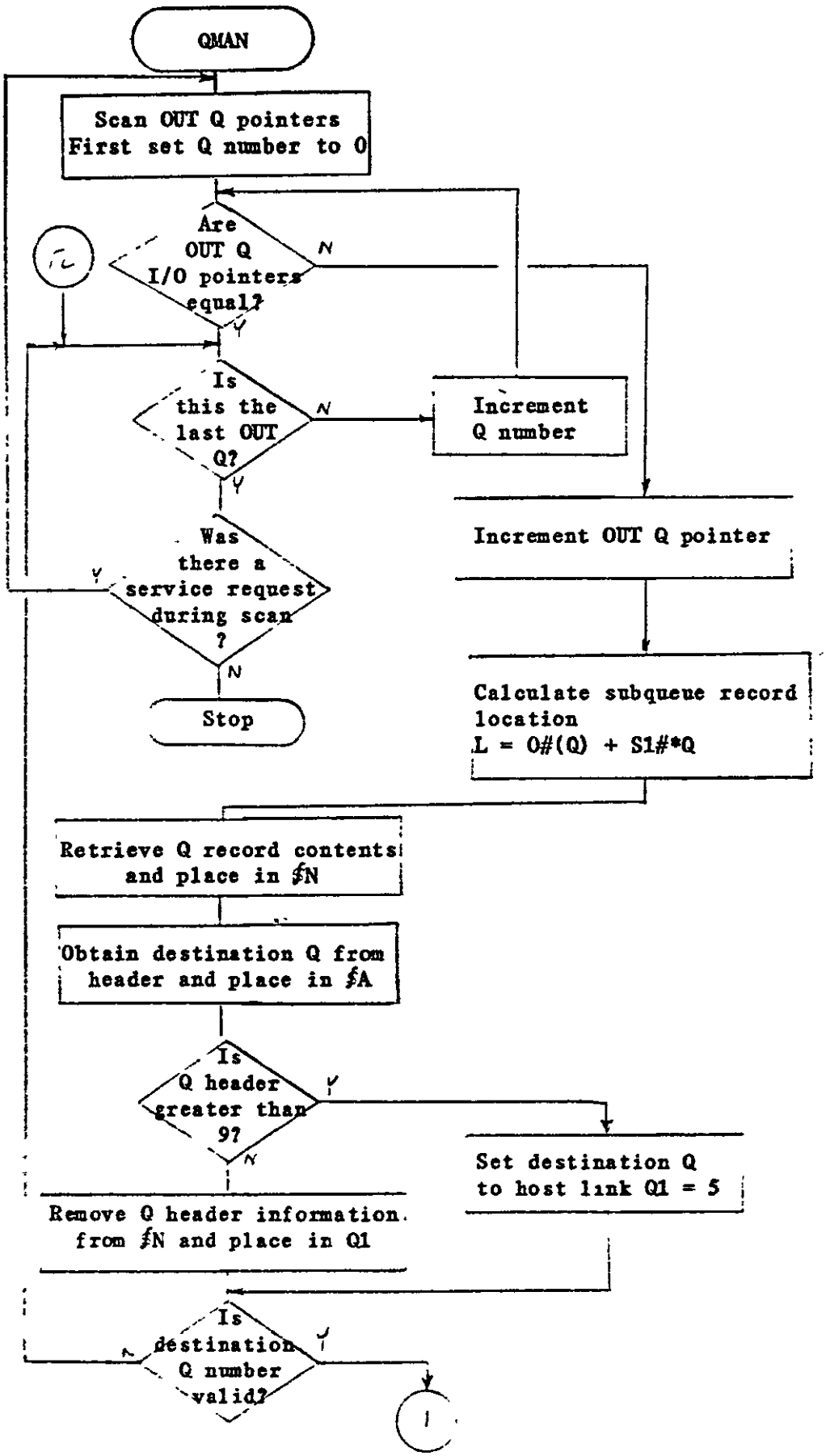
```
$SIZ

?LIS
Job   Name     Slot size   Free space   Job size   Unallocated
1     POW4     450         230          220        0
2     DISP4    750         50           700        0
3     TALK3T   250         30           220        0
4     KBDRI6   1600        30           1570       0
5     EP6      900         64           836        0
6     AG6      1000        398          602        0
7     QMAN4    500         67           433        0
8     DA18     1100        98           1002       0
9     DACON2   300         51           249        0
10    MEDIA6   1000        148          852        0
11    HELTH1   200         37           163        0
12    LISN2T   250         27           223        0
13    CHRON4   220         2            218        0
14    COMAH2   350         50           300        0
15    WD3      200         97           103        0
16    SETUP    20          2            18         0
17    PCPNC2   410         4            406        0
18    SIM1     60          2            58         0
19    TEST     140         19           121        0
20    JOB20    0           0            0          0
Globals        5017        1730
```

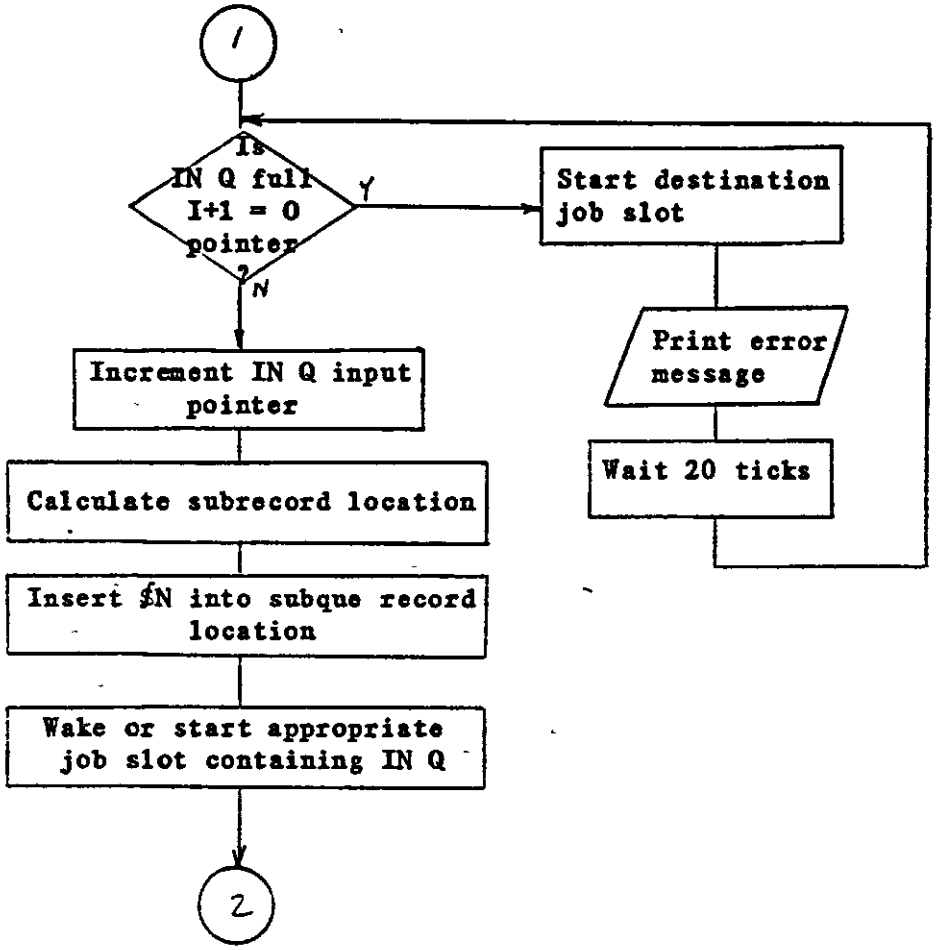
```
$PRO
Job no.  Owner   Prot    Prior   Privileges
1        GOD     10      100     GW GS JI MP DI OU FC FO AO NF MR MW
2                0       22      GW GS JI MP DI OU FC FO AO NF MR MW
3                0       44      GW GS JI MP DI OU FC FO AO NF MR MW
4                0       24      GW GS JI MP DI OU FC FO AO NF MR MW
5                0       25      GW GS JI MP DI OU FC FO AO NF MR MW
6                0       26      GW GS JI MP DI OU FC FO AO NF MR MW
7                0       41      GW GS JI MP DI OU FC FO AO NF MR MW
8                0       40      GW GS JI MP DI OU FC FO AO NF MR MW
9                0       29      GW GS JI MP DI OU FC FO AO NF MR MW
10               0       44      GW GS JI MP DI OU FC FO AO NF MR MW
11               0       31      GW GS JI MP DI OU FC FO AO NF MR MW
12               0       44      GW GS JI MP DI OU FC FO AO NF MR MW
13               0       33      GW GS JI MP DI OU FC FO AO NF MR MW
14       TOP     10      34      GW GS JI MP DI OU FC FO AO NF MR MW
15               0       45      GW GS JI MP DI OU FC FO AO NF MR MW
16               0       36
17               0       37      GW GS JI MP DI OU FC FO AO NF MR MW
18               0       38      GW GS JI MP DI OU FC FO AO NF MR MW
19               0       39      GW GS JI MP DI OU FC FO AO NF MR MW
20               0       40      GW GS JI MP DI OU FC FO AO NF MR MW

$USER
Owner Name?
Owner           Prot    Prior   Privileges
GOD             127     32766   SM GC GW GS JI MP DI OU FC FO AO NF MR MW
NE1             10      100     SM GC GW GS JI MP DI OU FC FO AO NF MR MW
TOP             10      5
```

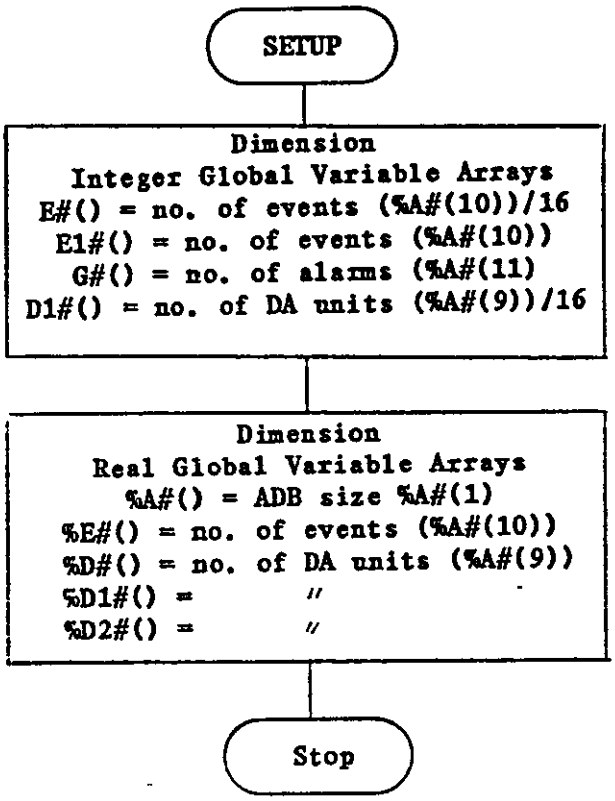
```
10 '--- ALARM DISPLAY PAGING PACKAGE
20 '--- G. HOENIG, LUT, MAR 1982
30 '--- VER 2.0
40 '
50 '--  SETUP
60 DEFINTP.L.D.S.W.X.I.T.N.F.C
63 W=100
65 DIMP(W,4),T$(W)
70 FORX=0TOW-2:P(X+1,1)=X:P(X+1,2)=X+2:NEXT
75 P(W,1)=W-1:P(W,2)=9999
80 LA=W:FI=1:N=0:PN=1:PG=0:D=1
85 DEF FNCO(W,X)=VAL(MID$(N$,W,X))
90 ONERRORGOTO900
95 PRINTCHR$(27);"ROC";
100 '--  RUN CONTROL
110 GOSUB2000
120 GOSUB1000
130 GOSUB200:L=LEN(N$):IFL=0THEN130ELSEF=FNCO(1,1)
140 IFF<4THENGOSUB300
144 NP=FI
145 IFF<>2THENGOSUB170
150 ONFGOSUB450,350,400,430,1200,1250
155 ONFGOTO160,130,160:GOTO130
160 IFD1<>1THEND=1:GOTO130
165 D1=0:D=X1+2:IFD>NTHEND=1:GOTO130
166 GOTO145
170 IFN=0THENRETURNELSEFORX=0TON:X1=X-1
175 IFP(NP,4)=0THEN180ELSEIFP(NP,3)=CTHENI=NP:X=N
180 NP=P(NP,2):NEXTX:RETURN
200 '--  I/O ROUTINE
205 ONERRORGOTO295
210 TIMEOUT25
215 IFM$=""THENM$="*"
220 PRINT#1,M$:M$=""
230 PRINTCHR$(27);"OBF";
235 LINEINPUT#1,N$
240 PRINTCHR$(27);"OB4";
245 ONERRORGOTO900
246 GOSUB1120:PRINTN$;CHR$(27);"OA1";
250 RETURN
295 IFERR=25THENPRINTCHR$(27);"OB4";:PRINT#1,"*":RESUME230ELSEONERRORG
OTO0
300 '--  DECODE
305 F=FNCO(1,1)
```

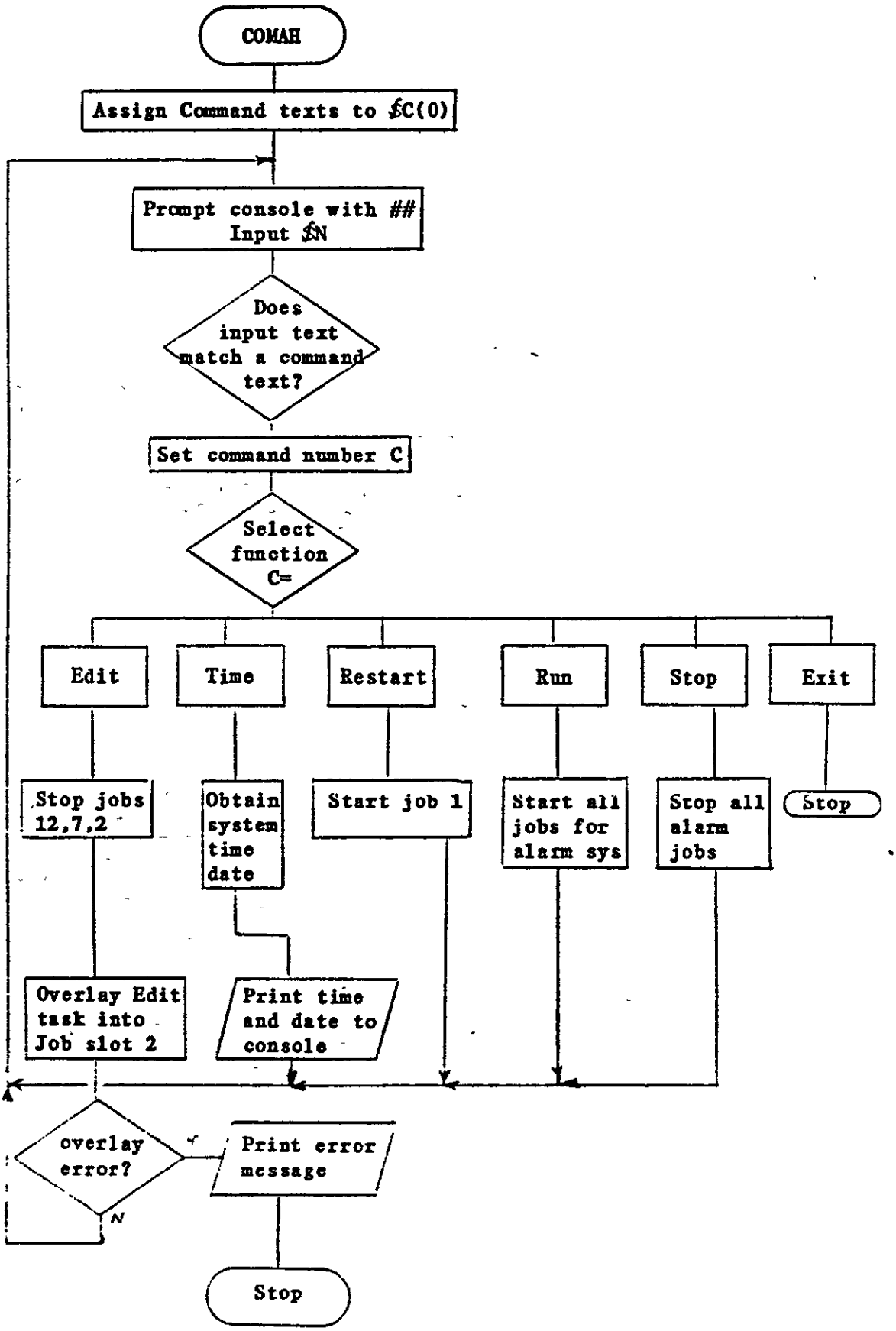
```
310 C=FNCO(2,3)
315 T0=FNCO(2,3):T1=FNCO(5,2):T2=FNCO(9,2)
320 D0=FNCO(11,2):D1=FNCO(13,2):D2=FNCO(15,2)
325 RETURN
350 '--  ACTIVATE
355 GOSUB500
360 RETURN
400 '--  ACCEPT
405 ONP(I,4)GOTO410,415,420:RETURN
410 D1=1:RETURN
415 GOSUB1300:P(I,4)=1:RETURN
420 GOSUB1300:P(I,4)=0:RETURN
430 '--  GEN ACC"
435 IFN=0THENRETURNELSEI=FI:FORL=1TON:X1=L-1:GOSUB400:I=P(I,2):NEXTL
440 RETURN
450 '--  NAL
455 ONP(I,4)GOTO460,465,470:D1=1:RETURN
460 D1=1:GOSUB1300:P(I,4)=0:RETURN
465 D1=1:P(I,4)=3:RETURN
470 D1=1;RETURN
500 '--  INSERT
505 IFN+1>WTHENRETURN
510 N=N+1:P(PN,3)=C:I=PN:P(PN,4)=2:PN=P(PN,2):L=N:GOSUB650:RETURN
550 '--  DELETE
555 IFI=FITHENFI=P(I,2)
560 P(LA,2)=I:P(P(I,2),1)=P(I,1):P(P(I,1),2)=P(I,2):P(I,1)=LA:LX=P(I,2
):P(I,2)=9999:N=N-1:LA=I:RETURN
600 '--  LIST/REPK
605 GOSUB1100
610 IFN=0THENRETURN
630 LL=FI:L1=N:FORL=1TOL1:I=LL
633 LX=P(LL,2)
635 IFP(LL,4)=0THENGOSUB550
640 LL=LX:NEXTL:IFN=0THENRETURN
645 LL=FI:FORL=1TON:I=LL:LL=P(LL,2):GOSUB650:NEXTL:RETURN
650 '--  PRINT/ADD
655 IFL<PGORL>PG+45THENRETURN
660 PRINTL;CHR$(9);:ONP(I,4)GOSUB670,675,675
665 PRINTT$(P(I,3)):RETURN
670 PRINT"A";:RETURN
675 PRINT"*";:RETURN
900 '--  ERROR
905 ONERRORGOTO0
1000 '--  SCREEN INT
1003 PRINTCHR$(12)
1005 PRINTCHR$(27);"DA0~P~W511511000480";
1010 PRINTCHR$(27);"DA1~P~W511013000478";
1015 PRINTCHR$(27);"DA2~P~W0,0,511,11";
1020 PRINT"~C1~G'511,12,0,12,511,479,0,479":CHR$(21);"~C2";
1100 '-- BLANK1
1105 PPRINTCHR$(27);"DA1":CHR$(12):RETURN
```

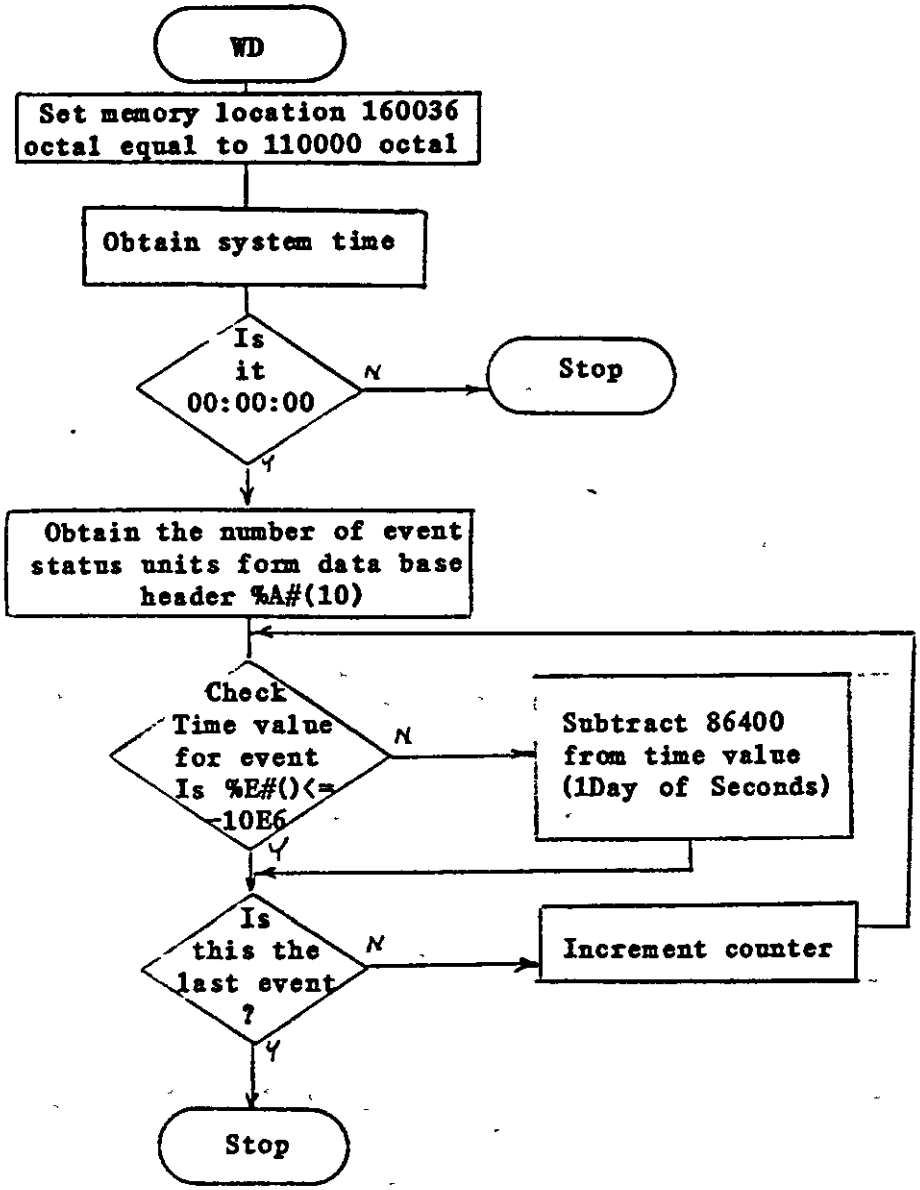
```
1120 '--  BLANK 0
1125 PRINTCHR$(27);"0A0";CHR$(12):RETURN
1140 '--  BLANK 2
1145 PRINTCHR$(27);"0A2";CHR$(12):RETURN
1200 '--  SRC UP
1205 PG=PG-10:IFPG<0THENPG=0
1210 GOSUB600:RETURN
1250 '--  SRC DOWN
1255 PG=PG+10:IFPG>=NTHENPG=N
1260 GOSUB600:RETURN
1300 '--  REMOVE
1305 IFX1<PGORX1>PG+45THENRETURN
1310 CX=CURSY(1):CY=CURSY(1):PRINTCHR$(28)
1315 IFX1-PG=0THEN1325
1320 FORJ=1TOX1-PG:PRINTCHR$(10);:NEXTJ
1325 ONP(I,4)GOTO1330,1335,1330
1330 PRINT"~@";:GOTO1350
1335 PRINTCHR$(9);"A";:GOTO1350
1350 PRINT"~U";:PLOTCX,PLOTCY:RETURN
1400 '--  UPDATE
1405 RETURN
2000 '--  LOAD
2010 RETURN
9000 FORH=1TO10:FORG=1TO4:PRINTP(H,G),:NEXTG:PRINT:NEXTH
=C2R0k
```

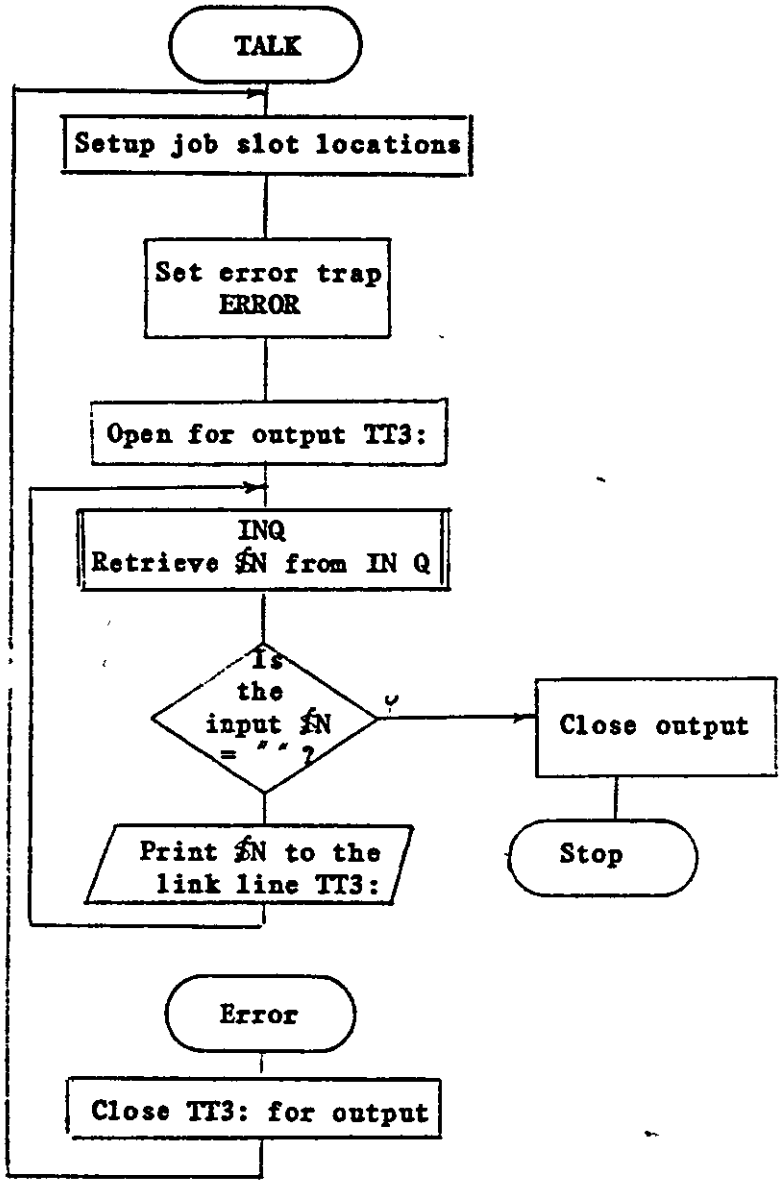

QMAN
Scan OUT Q pointers
First set Q number to 0
Are OUT Q I/O pointers equal?
N
Y
Is this the last OUT Q?
N
Y
Increment Q number
Increment OUT Q pointer
Was there a service request during scan ?
Y
N
Stop
Calculate subqueue record location
L = O#(Q) + S1#*Q
Retrieve Q record contents and place in $N
Obtain destination Q from header and place in $A
Is Q header greater than 9?
Y
N
Set destination Q to host link Q1 = 5
Remove Q header information from $N and place in Q1
Is destination Q number valid?
Y
N
1

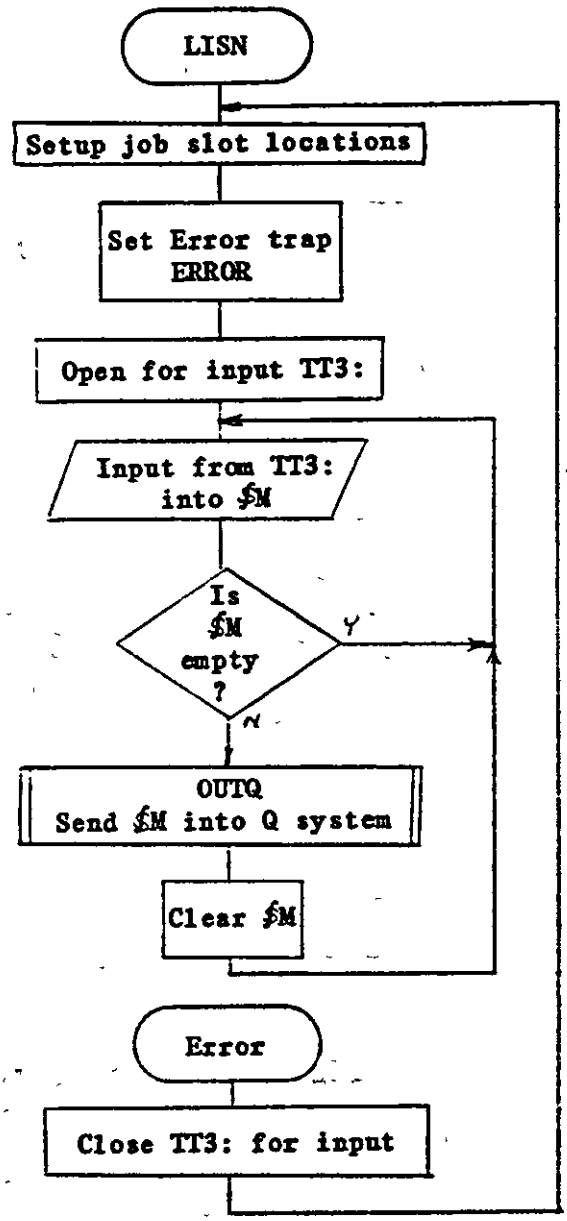

1
Is IN Q full I+1 = O pointer ?
Y
N
Start destination job slot
Print error message
Wait 20 ticks
Increment IN Q input pointer
Calculate subrecord location
Insert $N into subque record location
Wake or start appropriate job slot containing IN Q
2

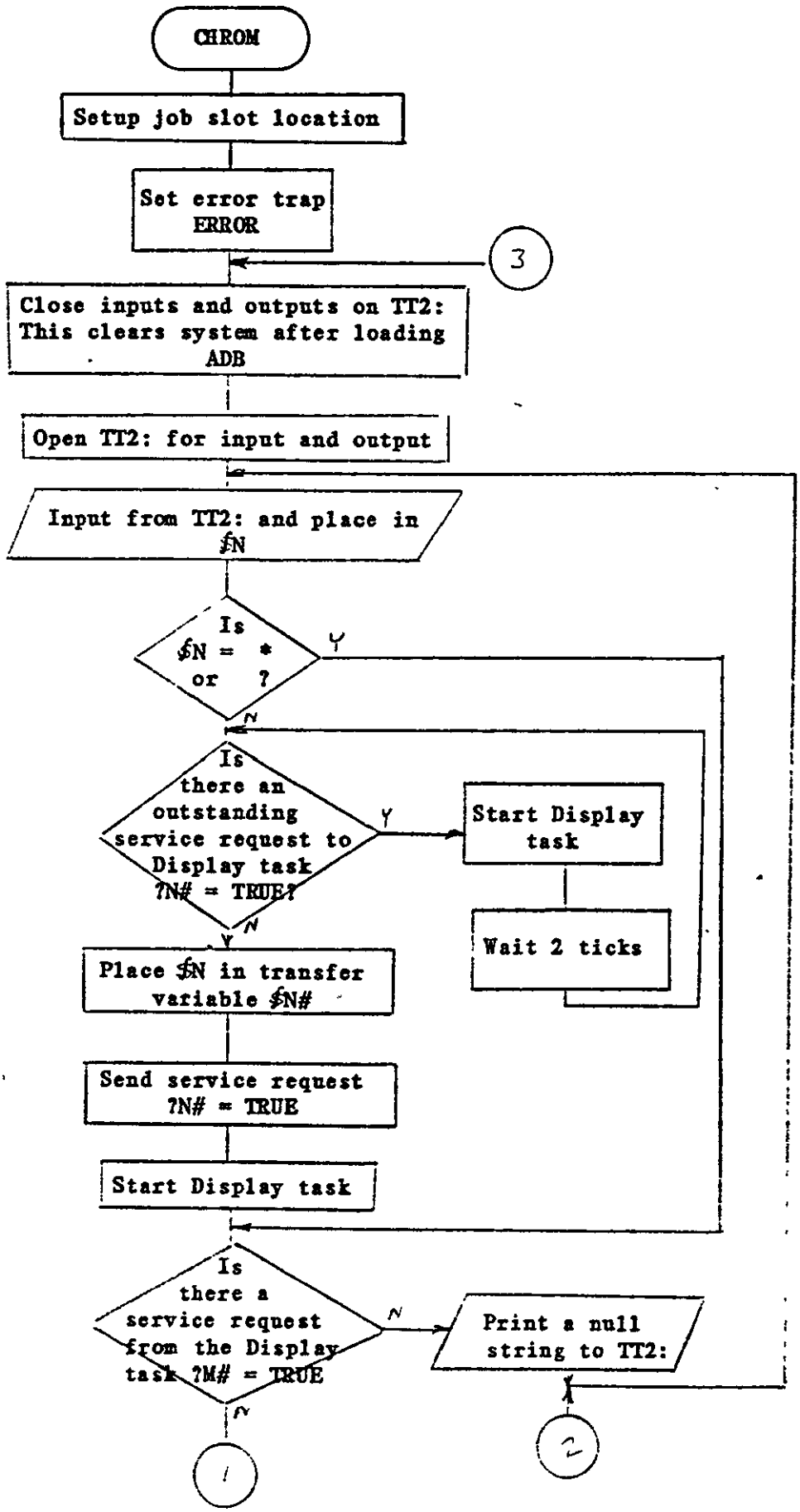

SETUP
Dimension
Integer Global Variable Arrays
E#() = no. of events (%A#(10))/16
E1#() = no. of events (%A#(10))
G#() = no. of alarms (%A#(11)
D1#() = no. of DA units (%A#(9))/16
Dimension
Real Global Variable Arrays
%A#() = ADB size %A#(1)
%E#() = no. of events (%A#(10))
%D#() = no. of DA units (%A#(9))
%D1#() = "
%D2#() = "
Stop

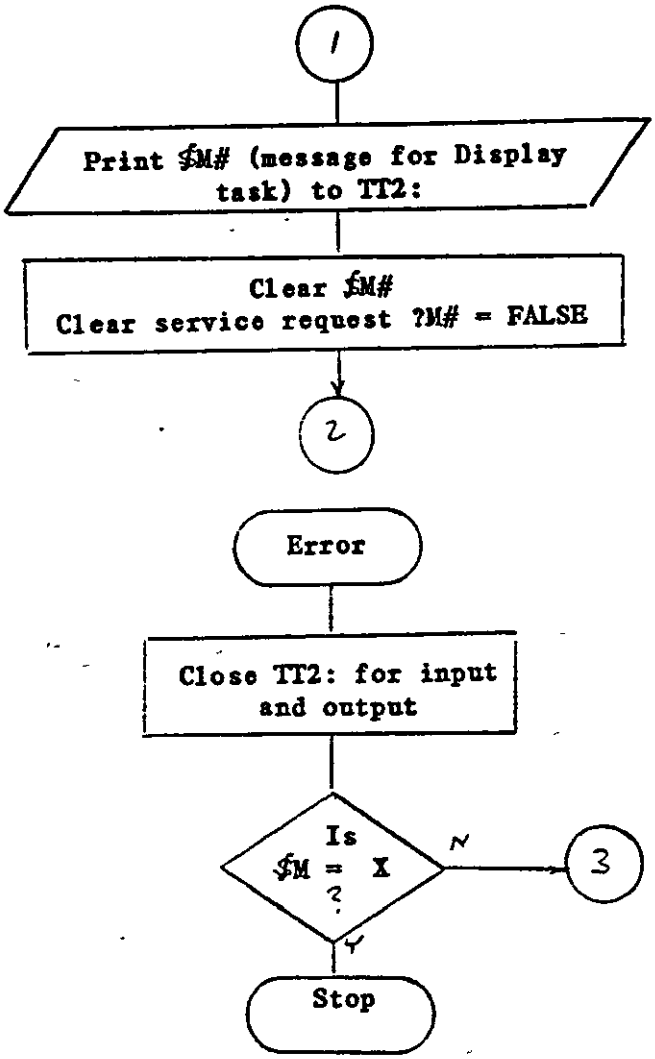

COMAH
Assign Command texts to $C(0)
Prompt console with ##
Input $N
Does input text match a command text?
Set command number C
Select function C=
Edit
Time
Restart
Run
Stop
Exit
Stop jobs 12,7,2
Obtain system time date
Start job 1
Start all jobs for alarm sys
Stop all alarm jobs
Stop
Overlay Edit task into Job slot 2
Print time and date to console
overlay error?
Y
N
Print error message
Stop

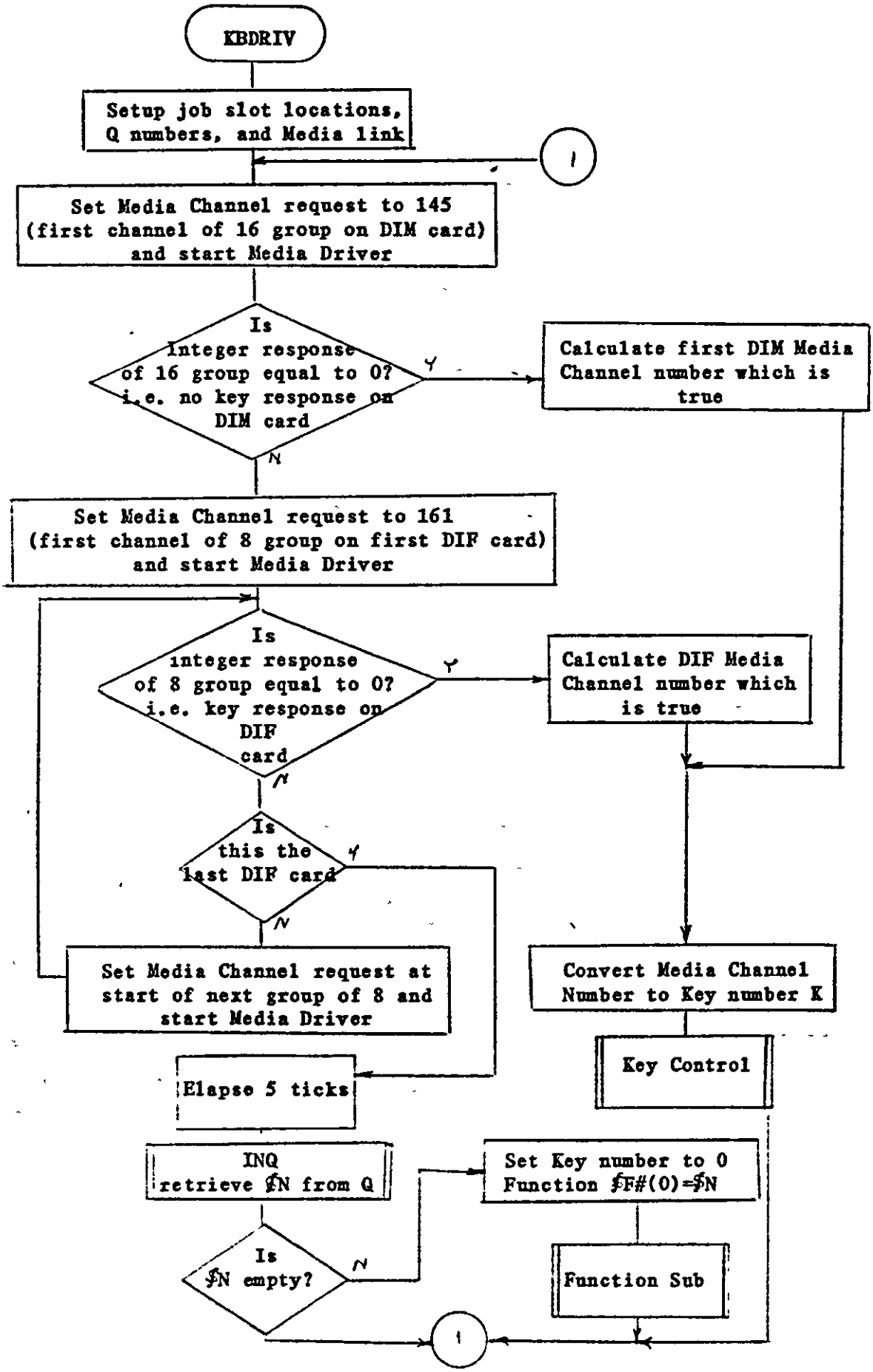

WD
Set memory location 160036 octal equal to 110000 octal
Obtain system time
Is it 00:00:00
N
Stop
Y
Obtain the number of event status units form data base header %A#(10)
Check Time value for event Is %E#()<= -10E6
N
Subtract 86400 from time value (1Day of Seconds)
Y
Is this the last event ?
N
Increment counter
Y
Stop

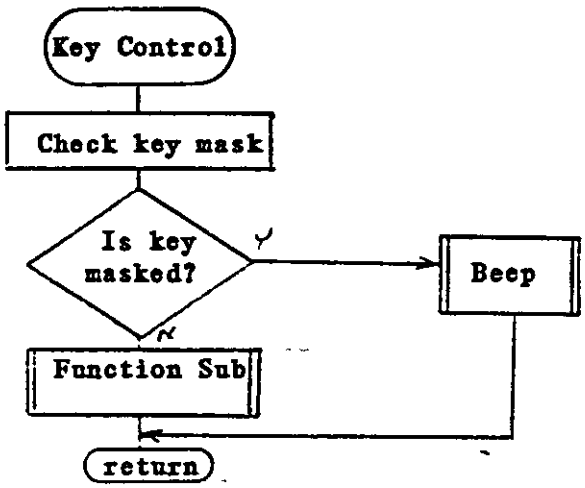

TALK
Setup job slot locations
Set error trap ERROR
Open for output TT3:
INQ Retrieve $N from IN Q
Is the input $N = "" ?
Y
Close output
Stop
Print $N to the link line TT3:
Error
Close TT3: for output

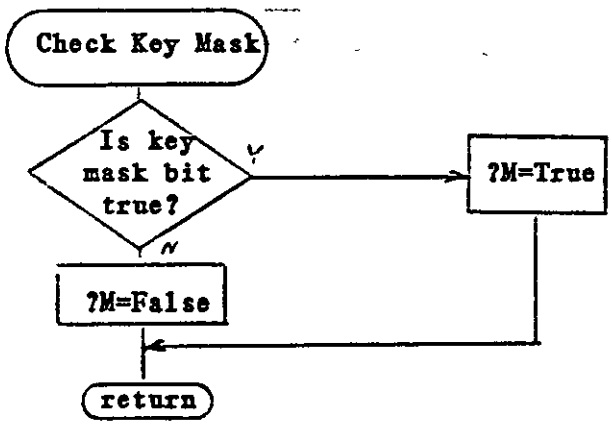

LISN
Setup job slot locations
Set Error trap ERROR
Open for input TT3:
Input from TT3: into $M
Is $M empty ?
Y
N
OUTQ
Send $M into Q system
Clear $M
Error
Close TT3: for input

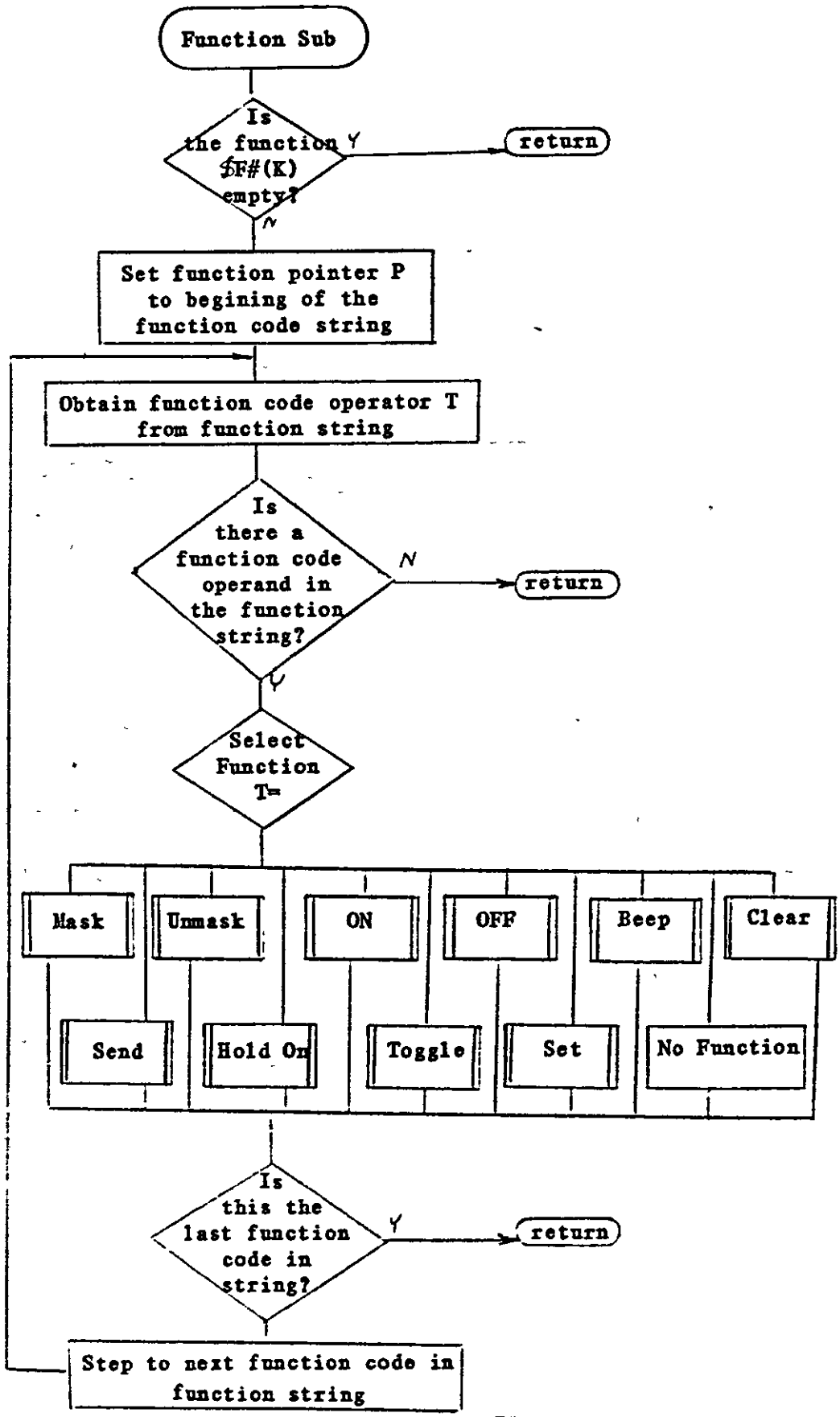

CHROM
Setup job slot location
Set error trap ERROR
3
Close inputs and outputs on TT2: This clears system after loading ADB
Open TT2: for input and output
Input from TT2: and place in $N
Is $N = * or ?
Y
N
Is there an outstanding service request to Display task ?N# = TRUE?
Y
N
Start Display task
Wait 2 ticks
Place $N in transfer variable $N#
Send service request ?N# = TRUE
Start Display task
Is there a service request from the Display task ?M# = TRUE
N
Y
Print a null string to TT2:
1
2

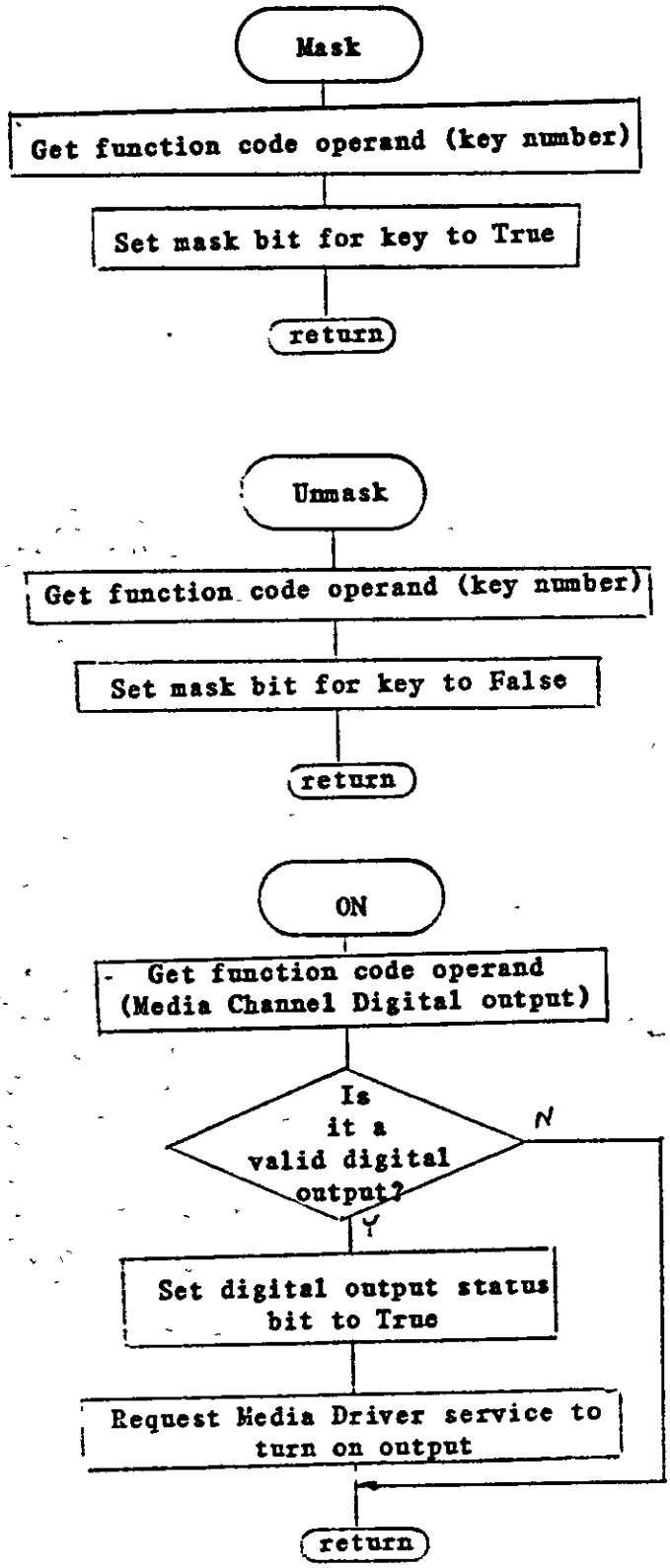

1
Print $M# (message for Display task) to TT2:
Clear $M#
Clear service request ?M# = FALSE
2
Error
Close TT2: for input and output
Is $M = X ?
N
3
Y
Stop

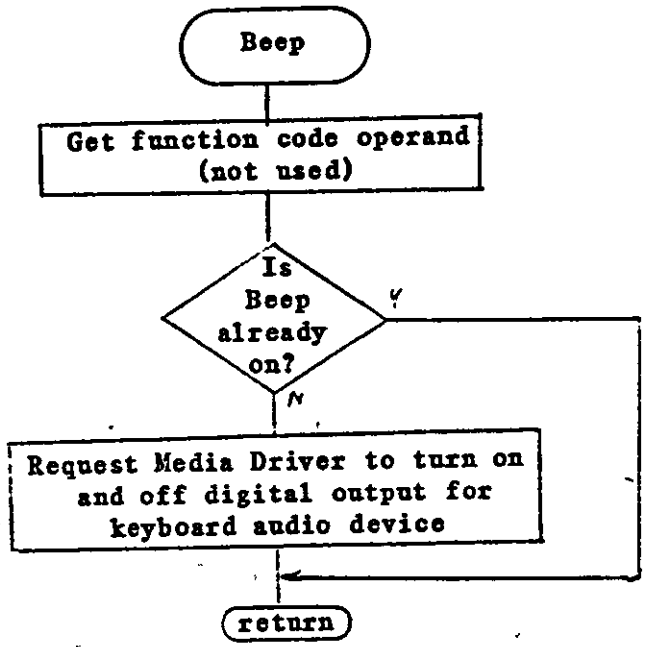

KBDRIV
Setup job slot locations, Q numbers, and Media link
1
Set Media Channel request to 145 (first channel of 16 group on DIM card) and start Media Driver
Is integer response of 16 group equal to 0? i.e. no key response on DIM card
Y
Calculate first DIM Media Channel number which is true
N
Set Media Channel request to 161 (first channel of 8 group on first DIF card) and start Media Driver
Is integer response of 8 group equal to 0? i.e. key response on DIF card
Y
Calculate DIF Media Channel number which is true
N
Is this the last DIF card
Y
N
Set Media Channel request at start of next group of 8 and start Media Driver
Convert Media Channel Number to Key number K
Key Control
Elapse 5 ticks
INQ retrieve $N from Q
Set Key number to 0 Function $F#(0)=$N
Is $N empty?
N
Function Sub
1

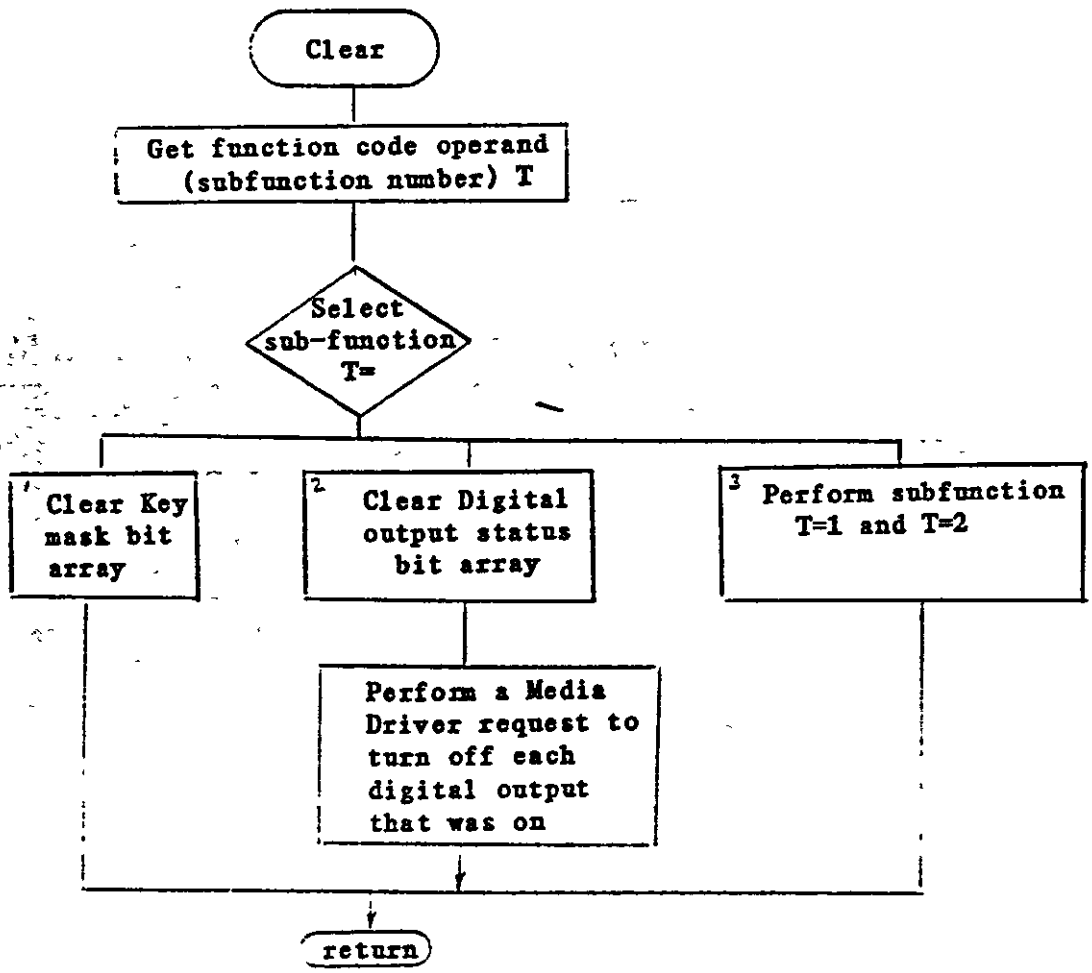

Key Control
Check key mask
Is key masked?
Y
N
Beep
Function Sub
return

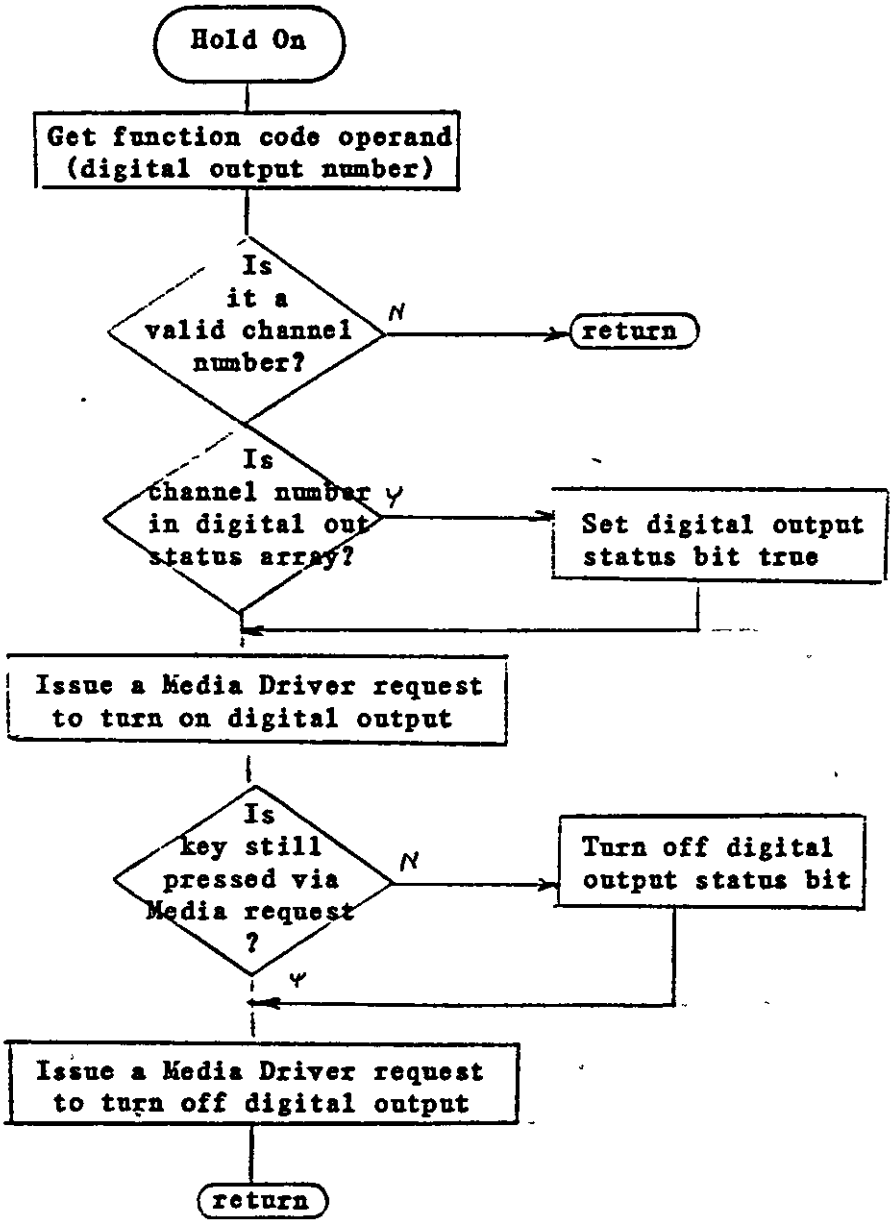

Check Key Mask
Is key mask bit true?
Y
N
?M=True
?M=False
return

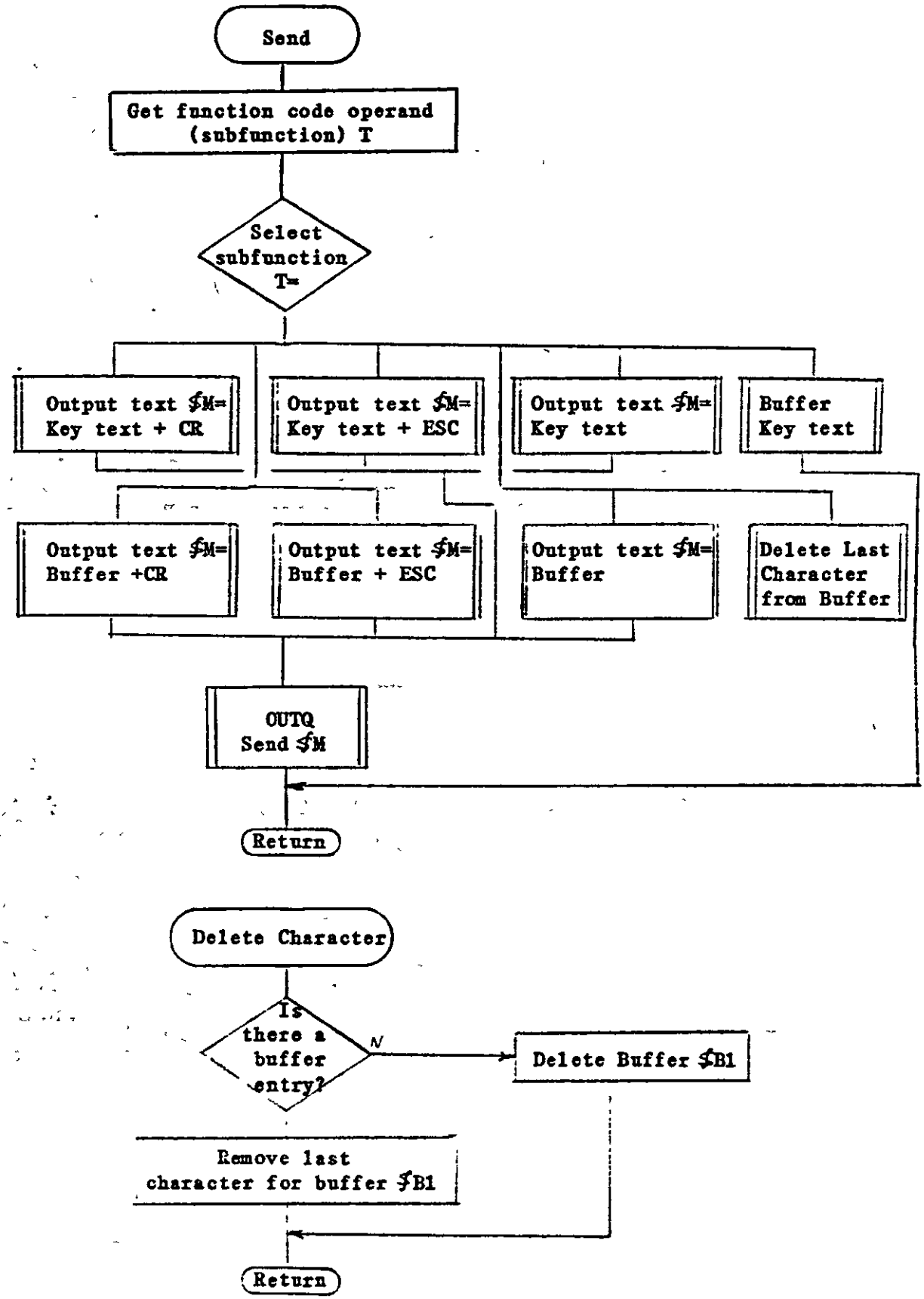

Function Sub
Is the function $F#(K) empty?
Y
return
N
Set function pointer P to begining of the function code string
Obtain function code operator T from function string
Is there a function code operand in the function string?
N
return
Y
Select Function T=
Mask
Unmask
ON
OFF
Beep
Clear
Send
Hold On
Toggle
Set
No Function
Is this the last function code in string?
Y
return
Step to next function code in function string

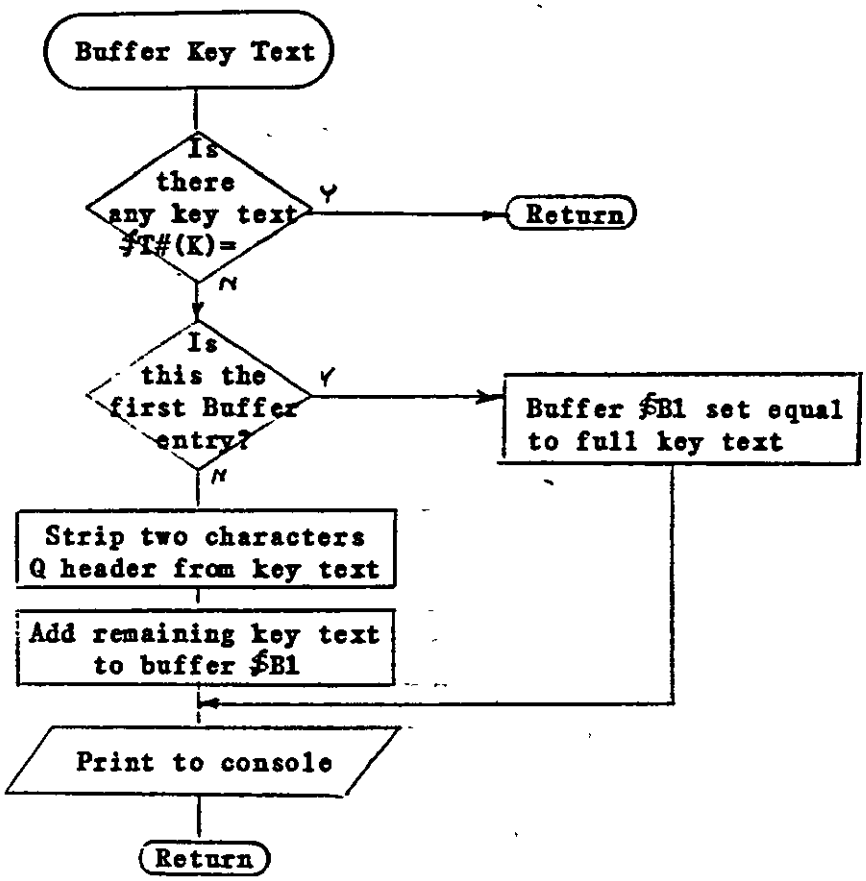

Mask
Get function code operand (key number)
Set mask bit for key to True
return
Unmask
Get function code operand (key number)
Set mask bit for key to False
return
ON
Get function code operand (Media Channel Digital output)
Is it a valid digital output?
N
Y
Set digital output status bit to True
Request Media Driver service to turn on output
return

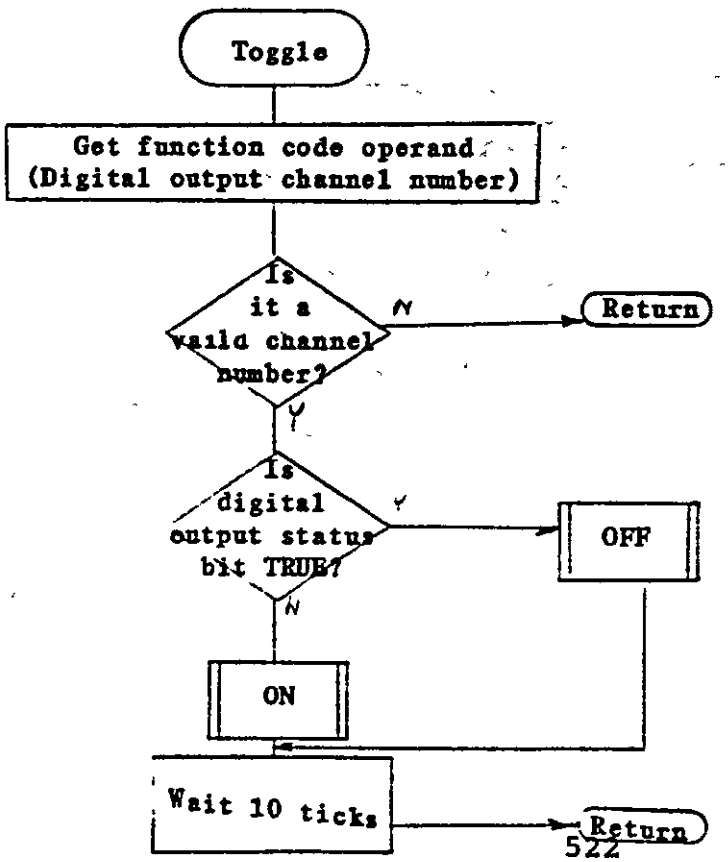

Beep
Get function code operand (not used)
Is Beep already on?
Y
N
Request Media Driver to turn on and off digital output for keyboard audio device
return

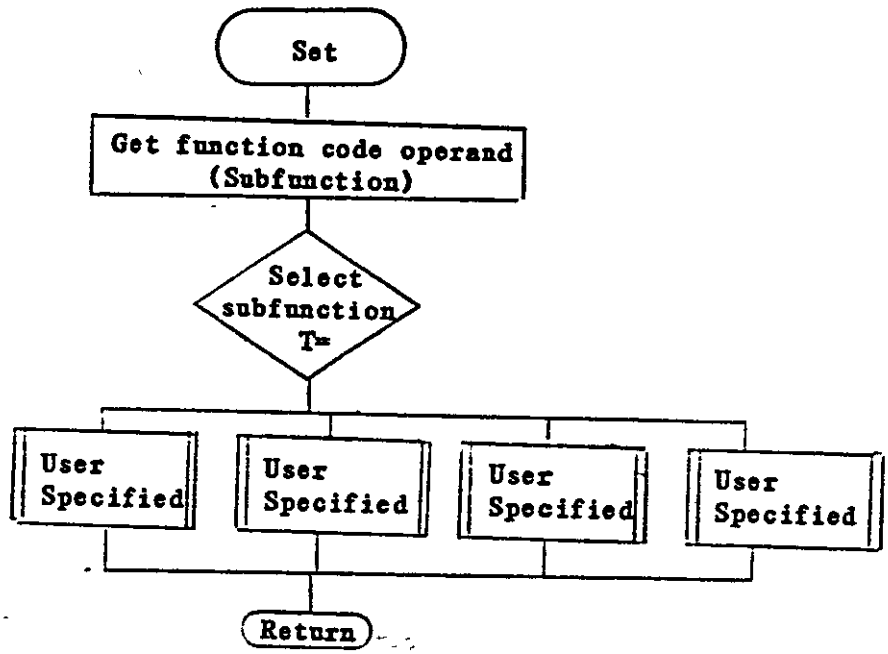

Clear
Get function code operand (subfunction number) T
Select sub-function T=
1
Clear Key mask bit array
2
Clear Digital output status bit array
3
Perform subfunction T=1 and T=2
Perform a Media Driver request to turn off each digital output that was on
return

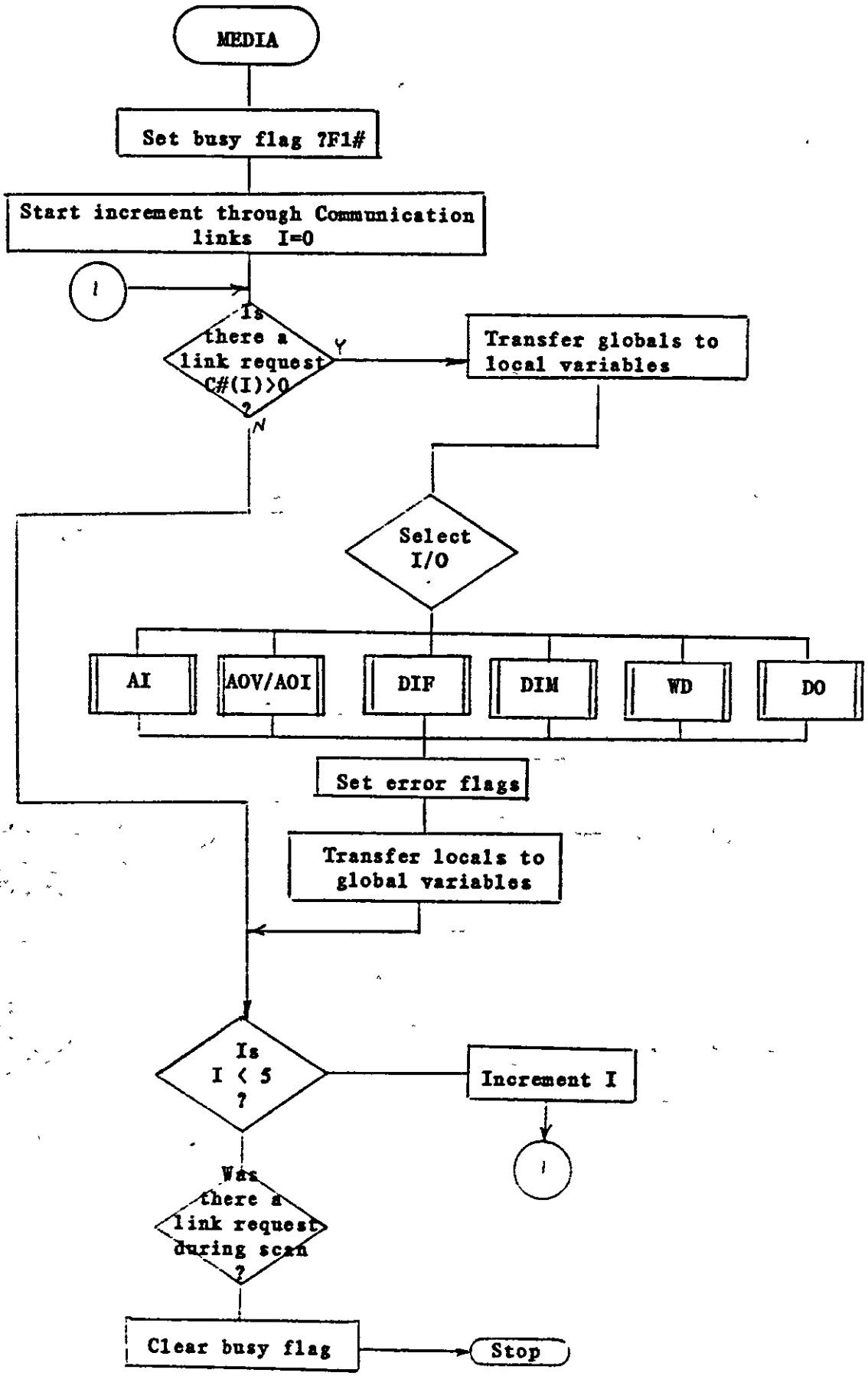

Hold On
Get function code operand (digital output number)
Is it a valid channel number?
N
return
Is channel number in digital out status array?
Y
Set digital output status bit true
Issue a Media Driver request to turn on digital output
Is key still pressed via Media request ?
N
Turn off digital output status bit
Y
Issue a Media Driver request to turn off digital output
return

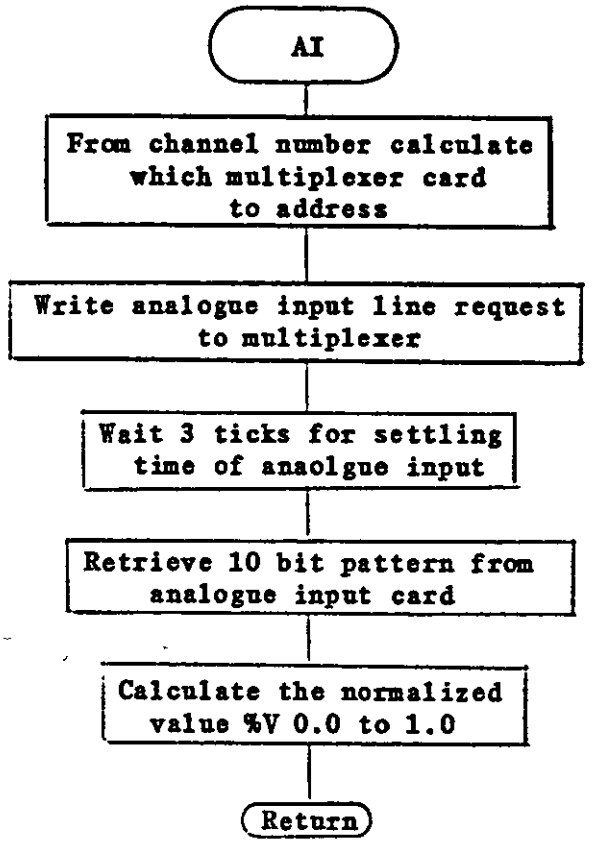

Send
Get function code operand (subfunction) T
Select subfunction T=
Output text $M= Key text + CR
Output text $M= Key text + ESC
Output text $M= Key text
Buffer Key text
Output text $M= Buffer +CR
Output text $M= Buffer + ESC
Output text $M= Buffer
Delete Last Character from Buffer
OUTQ Send $M
Return
Delete Character
Is there a buffer entry?
N
Delete Buffer $B1
Remove last character for buffer $B1
Return

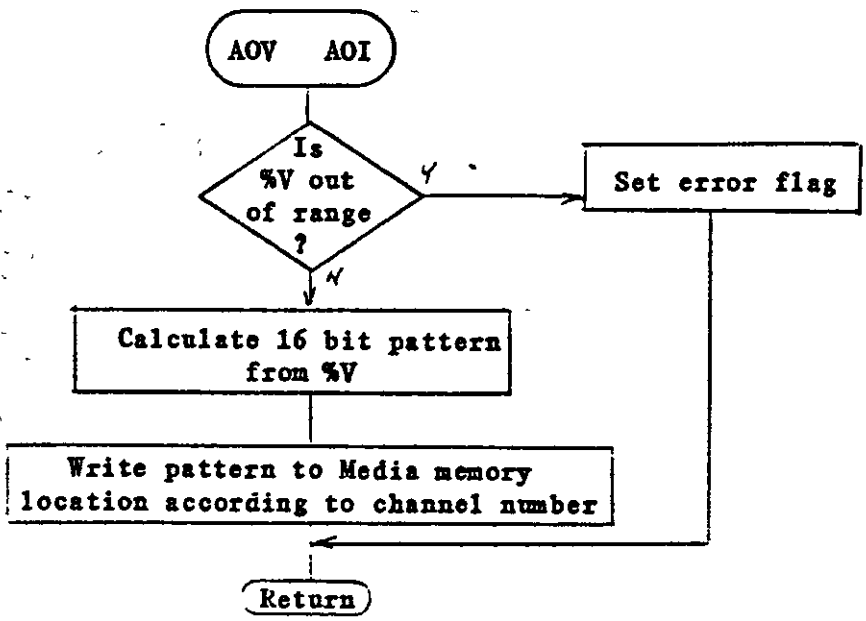

Buffer Key Text
Is there any key text $T#(K)=
Y
Return
N
Is this the first Buffer entry?
Y
Buffer $B1 set equal to full key text
N
Strip two characters Q header from key text
Add remaining key text to buffer $B1
Print to console
Return

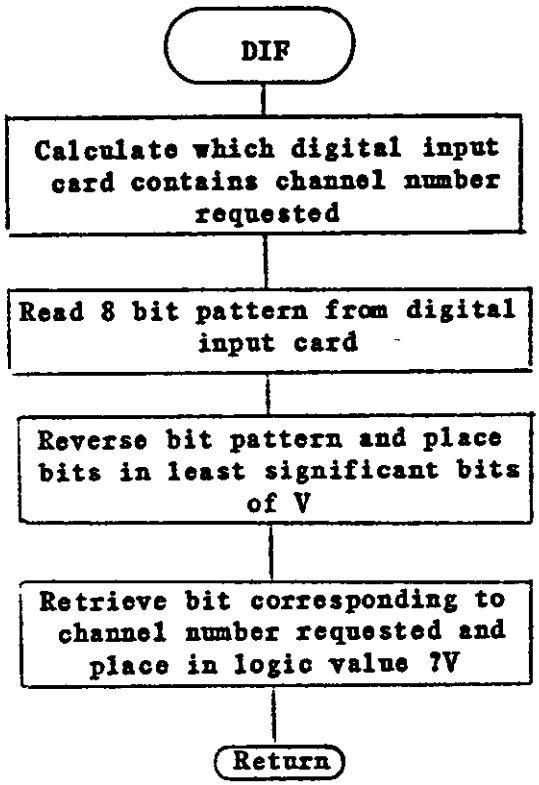

Toggle
Get function code operand (Digital output channel number)
Is it a valid channel number?
N
Return
Y
Is digital output status bit TRUE?
Y
OFF
N
ON
Wait 10 ticks
Return

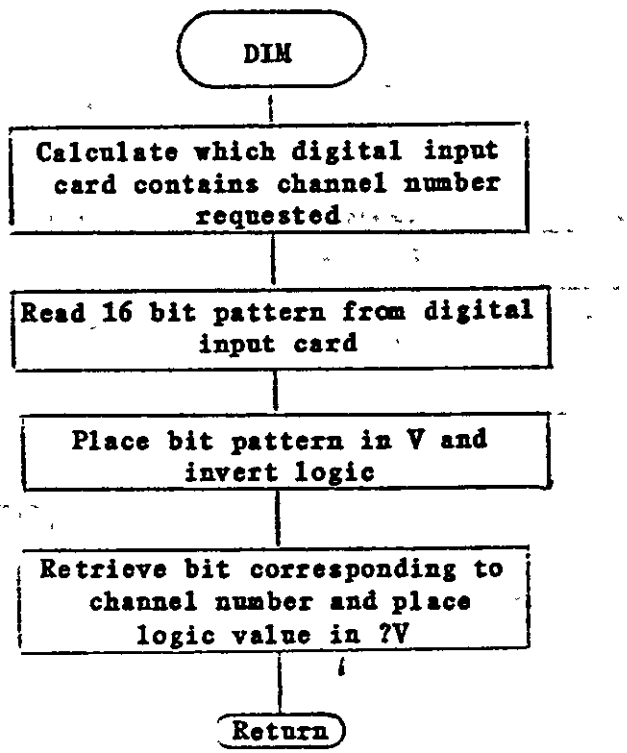

Set
Get function code operand
(Subfunction)
Select
subfunction
T=
User
Specified
User
Specified
User
Specified
User
Specified
Return

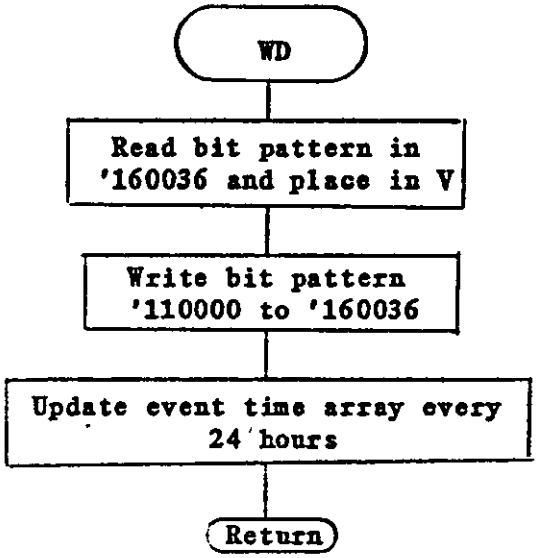

MEDIA
Set busy flag ?F1#
Start increment through Communication links I=0
1
Is there a link request C#(I)>0 ?
Y
N
Transfer globals to local variables
Select I/O
AI
AOV/AOI
DIF
DIM
WD
DO
Set error flags
Transfer locals to global variables
Is I < 5 ?
Increment I
1
Was there a link request during scan ?
Clear busy flag
Stop

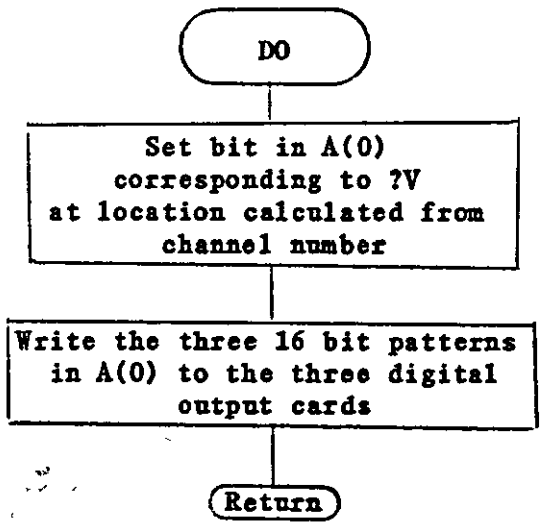

AI
From channel number calculate which multiplexer card to address
Write analogue input line request to multiplexer
Wait 3 ticks for settling time of anaolgue input
Retrieve 10 bit pattern from analogue input card
Calculate the normalized value %V 0.0 to 1.0
Return

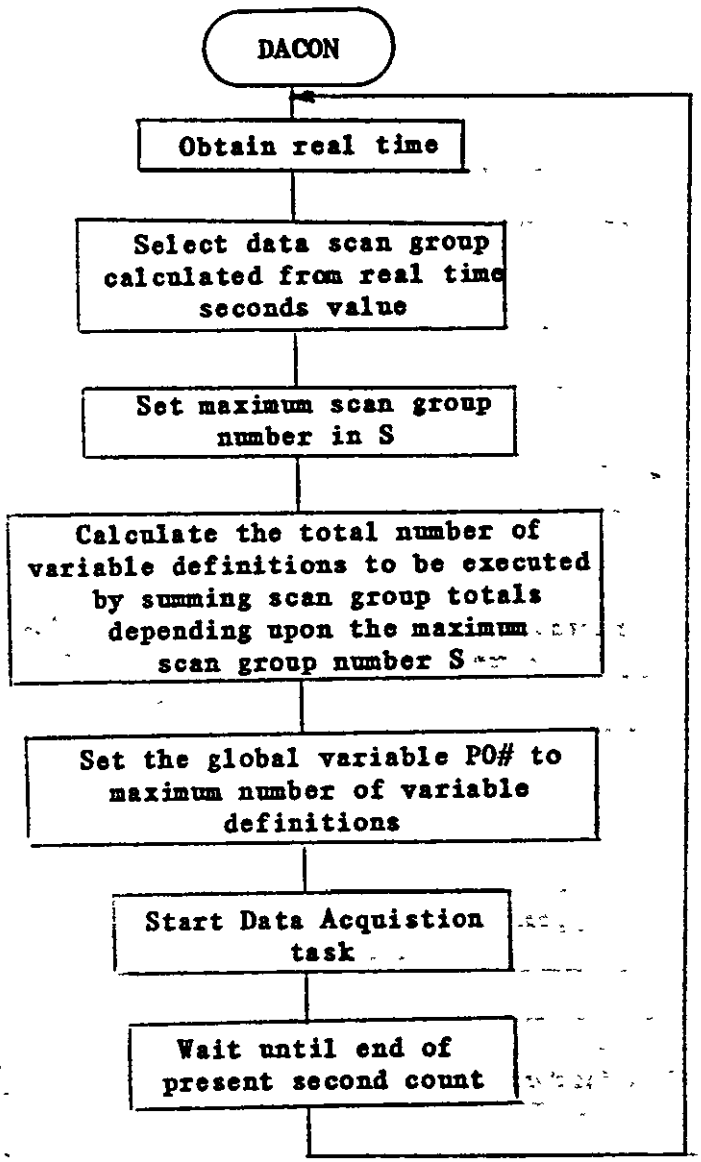

AOV AOI
Is %V out of range ?
Y
Set error flag
N
Calculate 16 bit pattern from %V
Write pattern to Media memory location according to channel number
Return

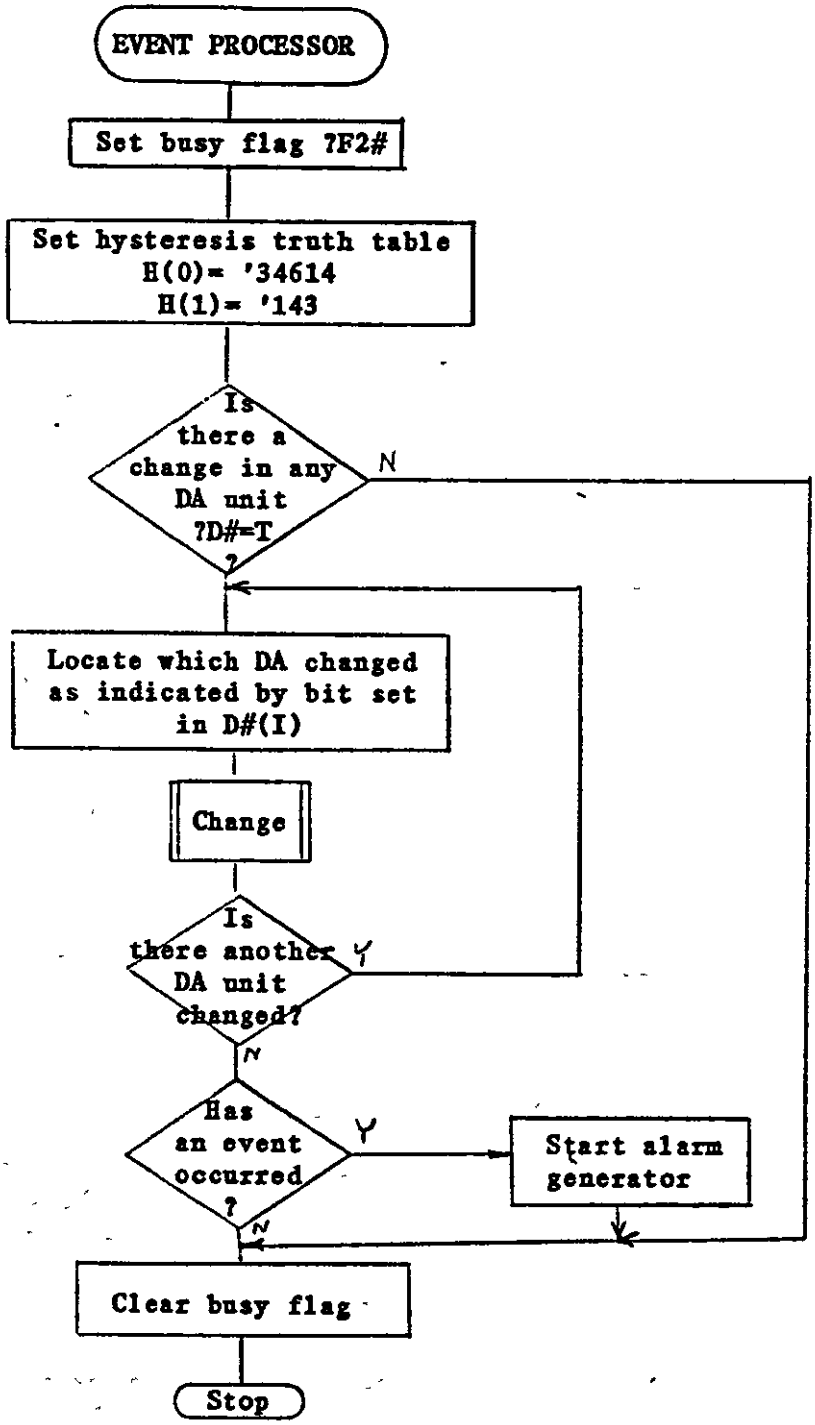

DIF
Calculate which digital input card contains channel number requested
Read 8 bit pattern from digital input card
Reverse bit pattern and place bits in least significant bits of V
Retrieve bit corresponding to channel number requested and place in logic value ?V
Return

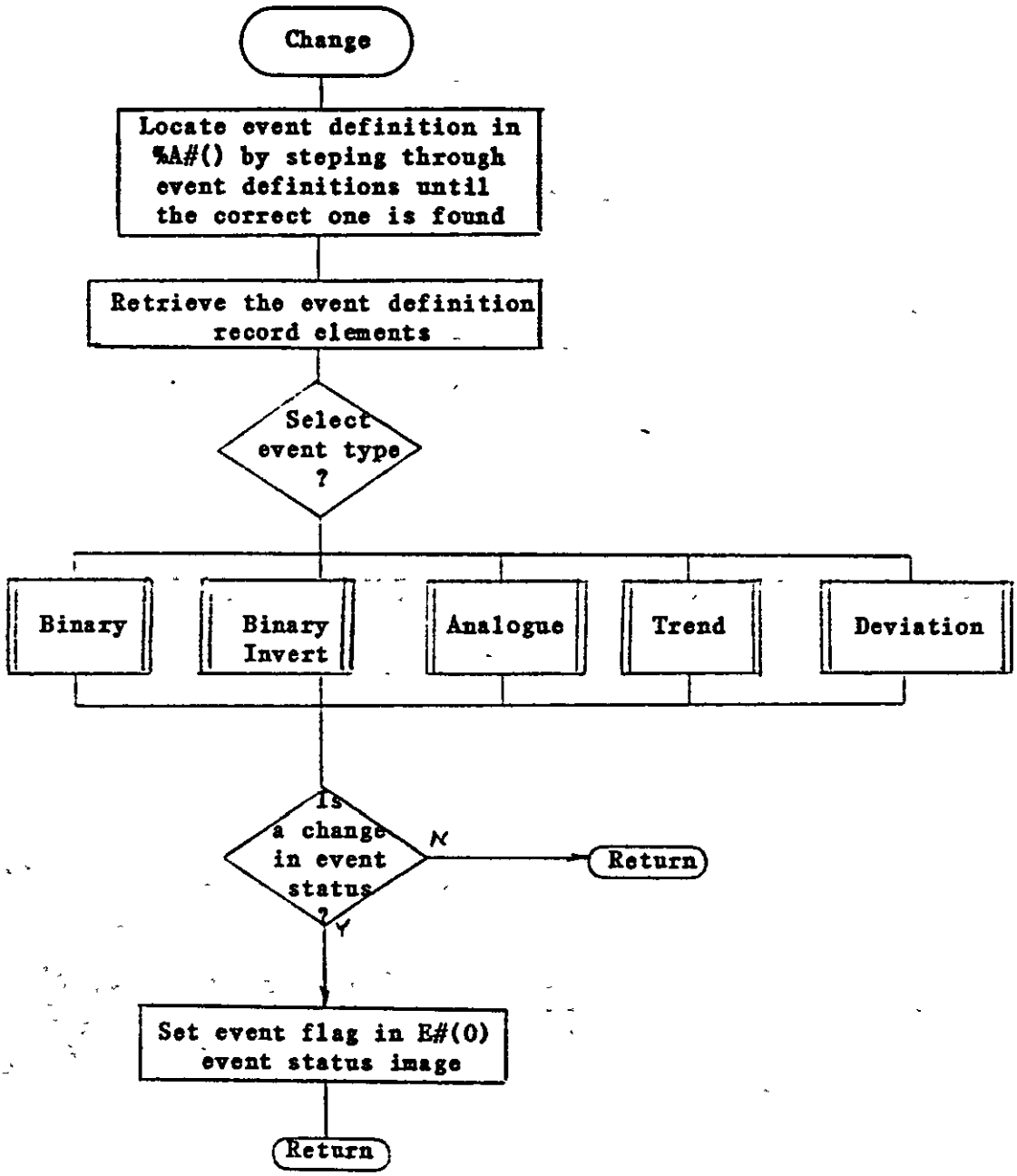

DIM
Calculate which digital input card contains channel number requested
Read 16 bit pattern from digital input card
Place bit pattern in V and invert logic
Retrieve bit corresponding to channel number and place logic value in ?V
Return

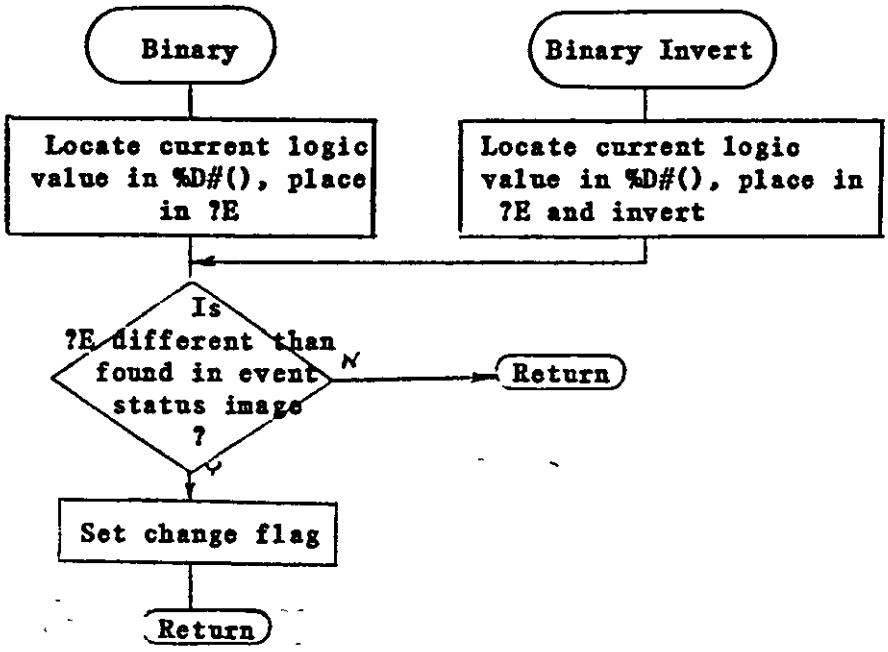

WD
Read bit pattern in '160036 and place in V
Write bit pattern '110000 to '160036
Update event time array every 24 hours
Return

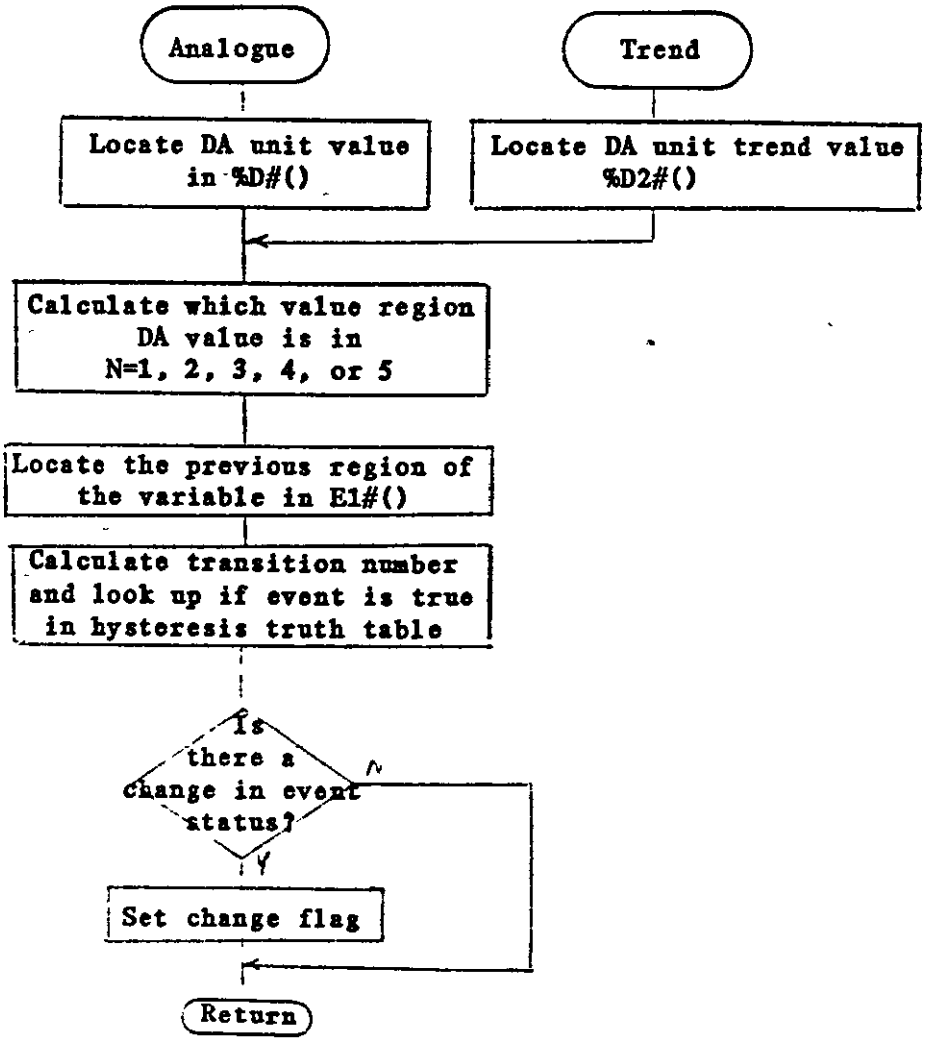

DO
Set bit in A(0) corresponding to ?V at location calculated from channel number
Write the three 16 bit patterns in A(0) to the three digital output cards
Return

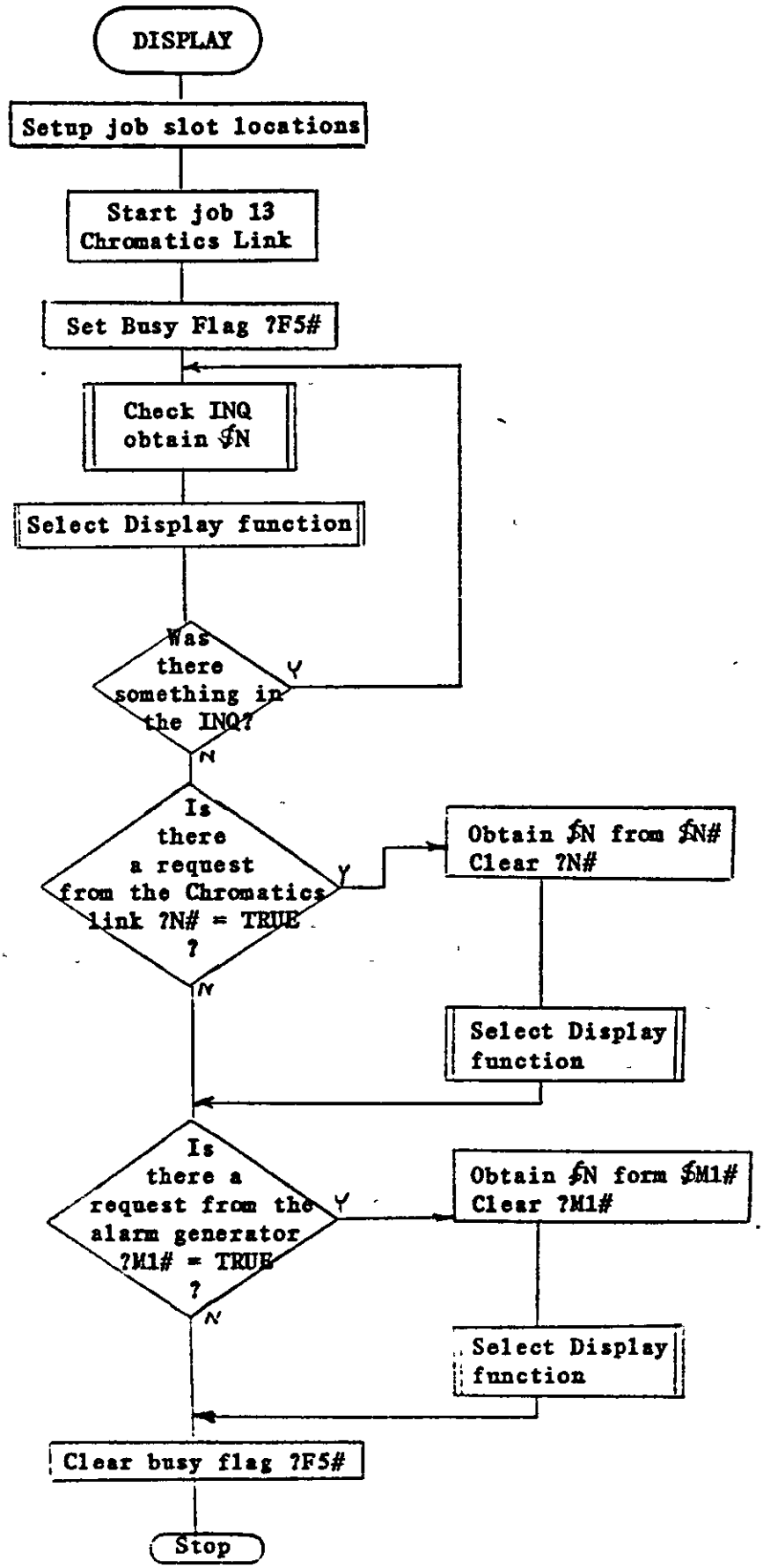

DACON
Obtain real time
Select data scan group calculated from real time seconds value
Set maximum scan group number in S
Calculate the total number of variable definitions to be executed by summing scan group totals depending upon the maximum scan group number S
Set the global variable P0# to maximum number of variable definitions
Start Data Acquistion task
Wait until end of present second count

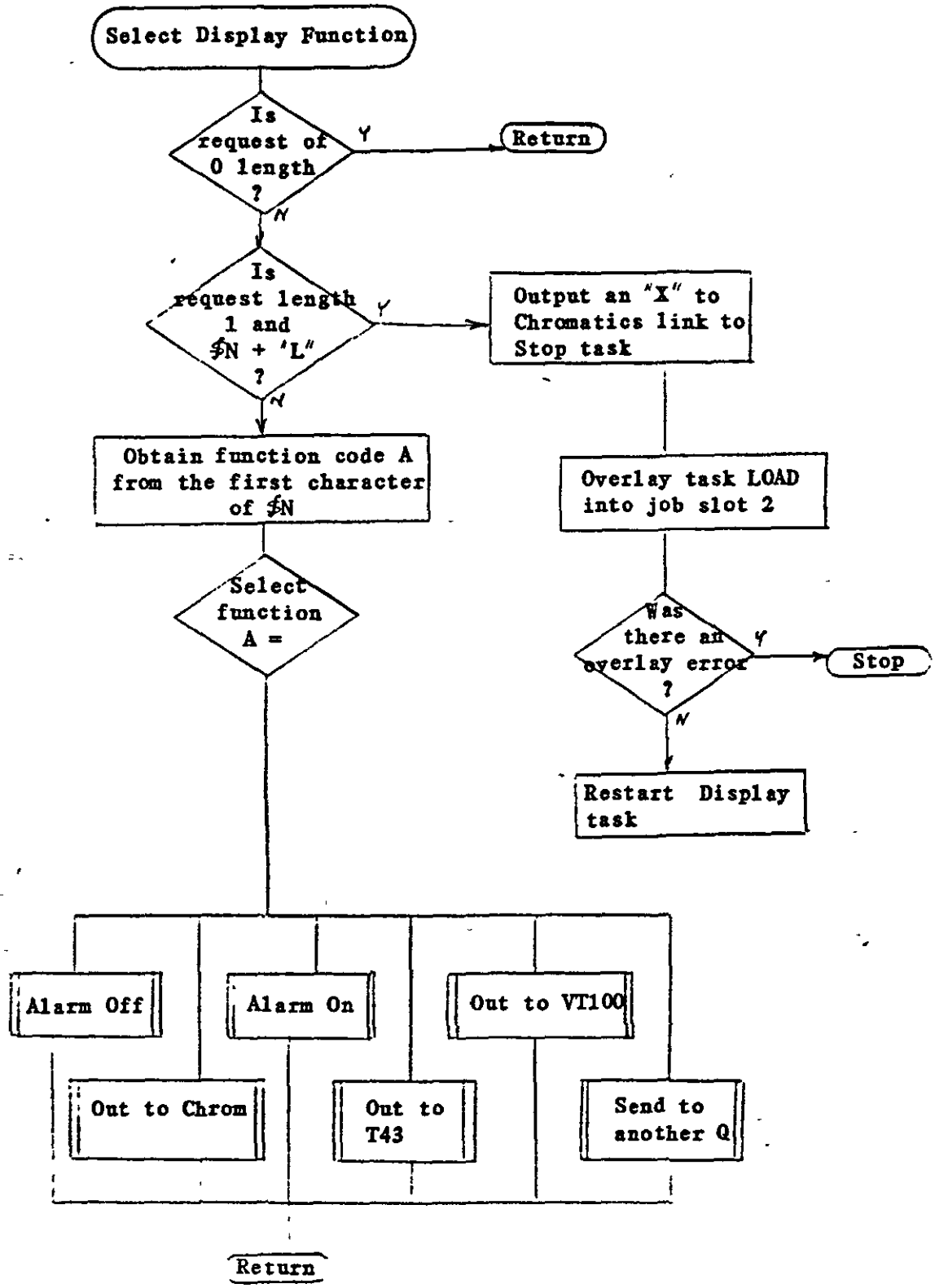

EVENT PROCESSOR
Set busy flag ?F2#
Set hysteresis truth table
H(0)= '34614
H(1)= '143
Is there a change in any DA unit ?D#=T ?
N
Locate which DA changed as indicated by bit set in D#(I)
Change
Is there another DA unit changed?
Y
N
Has an event occurred ?
Y
Start alarm generator
N
Clear busy flag
Stop

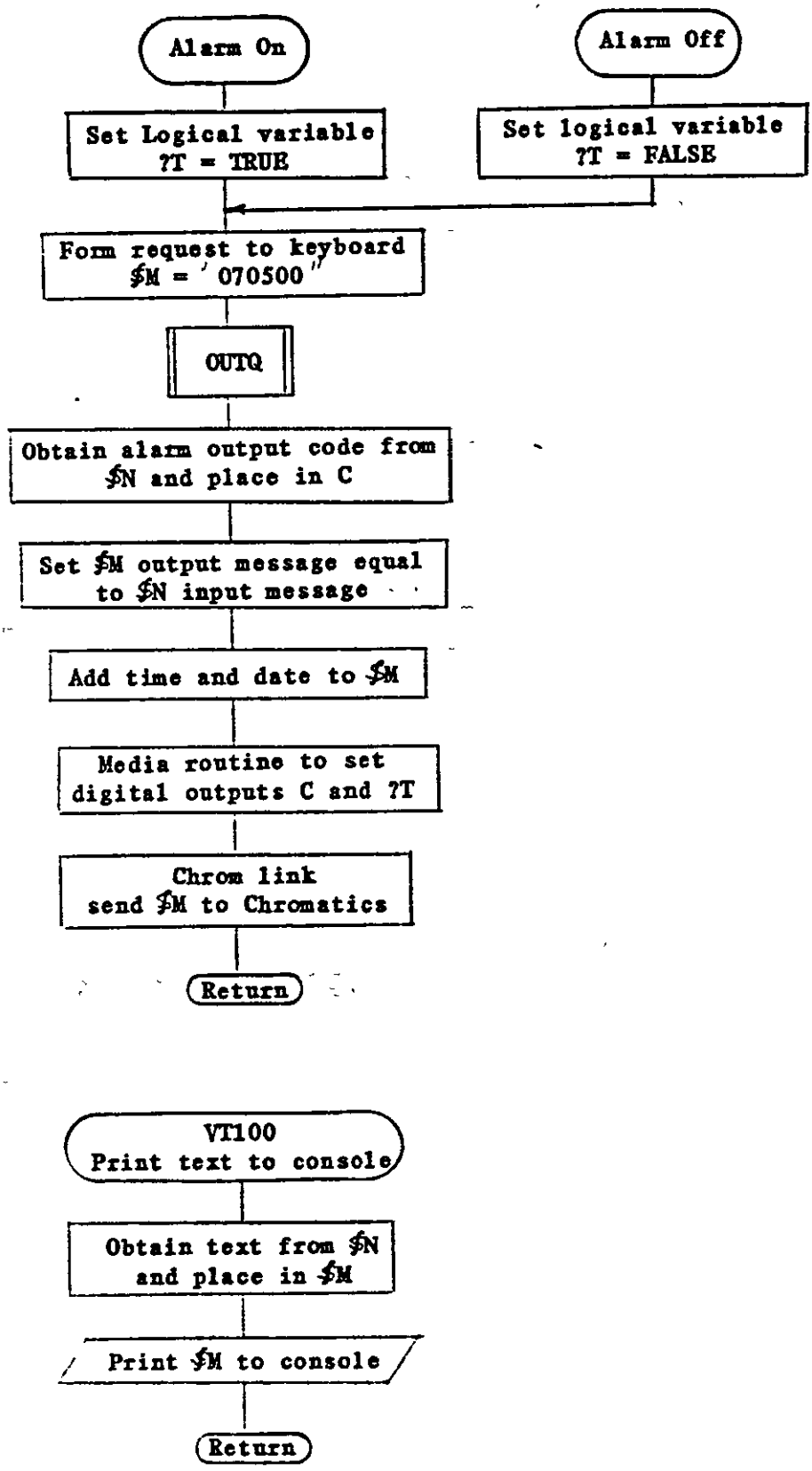

Change
Locate event definition in %A#() by steping through event definitions until the correct one is found
Retrieve the event definition record elements
Select event type ?
Binary
Binary Invert
Analogue
Trend
Deviation
Is a change in event status ?
N
Return
Y
Set event flag in E#(0) event status image
Return

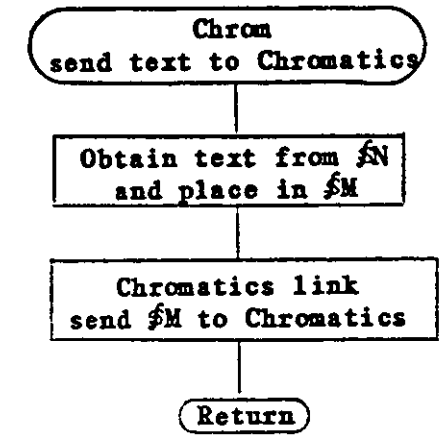

Binary
Binary Invert
Locate current logic value in %D#(), place in ?E
Locate current logic value in %D#(), place in ?E and invert
Is ?E different than found in event status image ?
N
Return
Y
Set change flag
Return

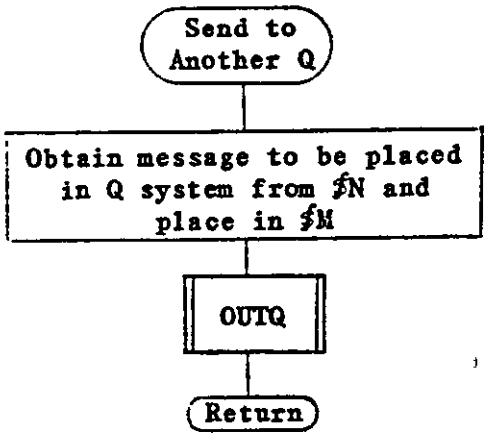

Analogue
Trend
Locate DA unit value in %D#()
Locate DA unit trend value %D2#()
Calculate which value region DA value is in N=1, 2, 3, 4, or 5
Locate the previous region of the variable in E1#()
Calculate transition number and look up if event is true in hysteresis truth table
Is there a change in event status?
N
Y
Set change flag
Return

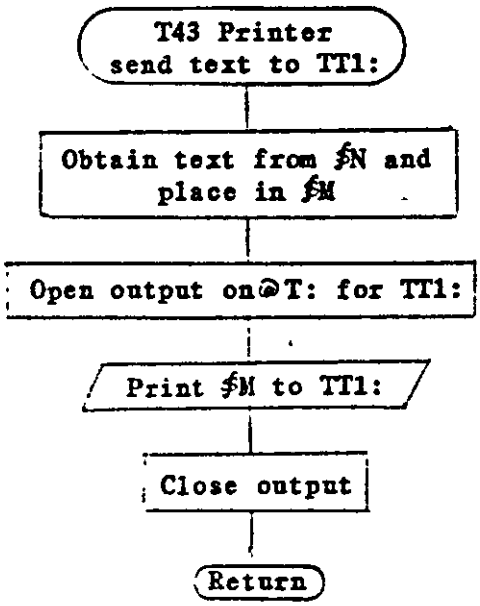

DISPLAY
Setup job slot locations
Start job 13
Chromatics Link
Set Busy Flag ?F5#
Check INQ
obtain $N
Select Display function
Was there something in the INQ?
Y
N
Is there a request from the Chromatics link ?N# = TRUE ?
Y
N
Obtain $N from $N#
Clear ?N#
Select Display function
Is there a request from the alarm generator ?M1# = TRUE ?
Y
N
Obtain $N form $M1#
Clear ?M1#
Select Display function
Clear busy flag ?F5#
Stop

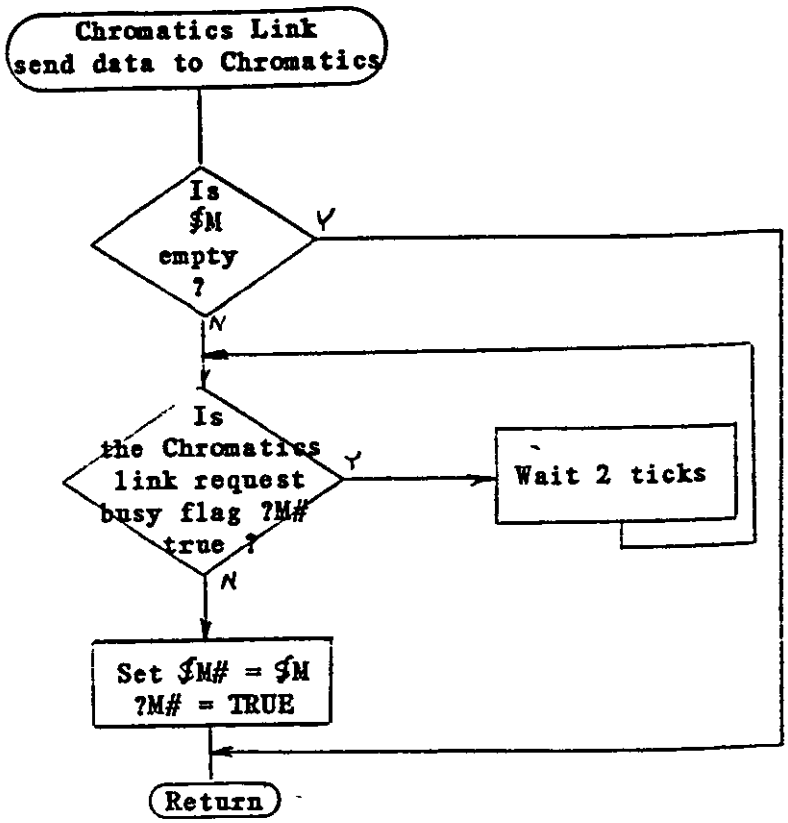

Select Display Function
Is request of 0 length ?
Y
Return
N
Is request length 1 and $N + 'L" ?
Y
Output an "X" to Chromatics link to Stop task
N
Obtain function code A from the first character of $N
Overlay task LOAD into job slot 2
Select function A =
Was there an overlay error ?
Y
Stop
N
Restart Display task
Alarm Off
Alarm On
Out to VT100
Out to Chrom
Out to T43
Send to another Q
Return

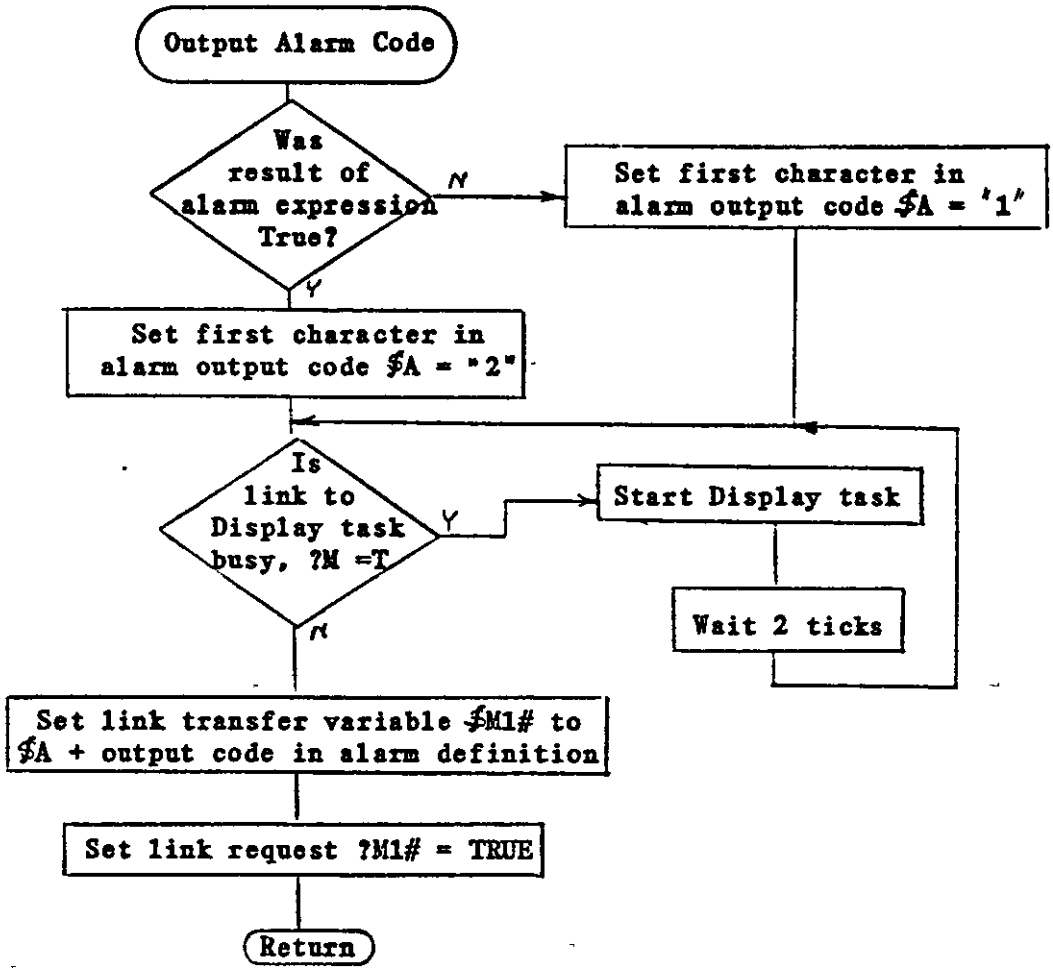

Alarm On
Alarm Off
Set Logical variable ?T = TRUE
Set logical variable ?T = FALSE
Form request to keyboard $M = ' 070500 "
OUTQ
Obtain alarm output code from $N and place in C
Set $M output message equal to $N input message
Add time and date to $M
Media routine to set digital outputs C and ?T
Chrom link send $M to Chromatics
Return
VT100 Print text to console
Obtain text from $N and place in $M
Print $M to console
Return

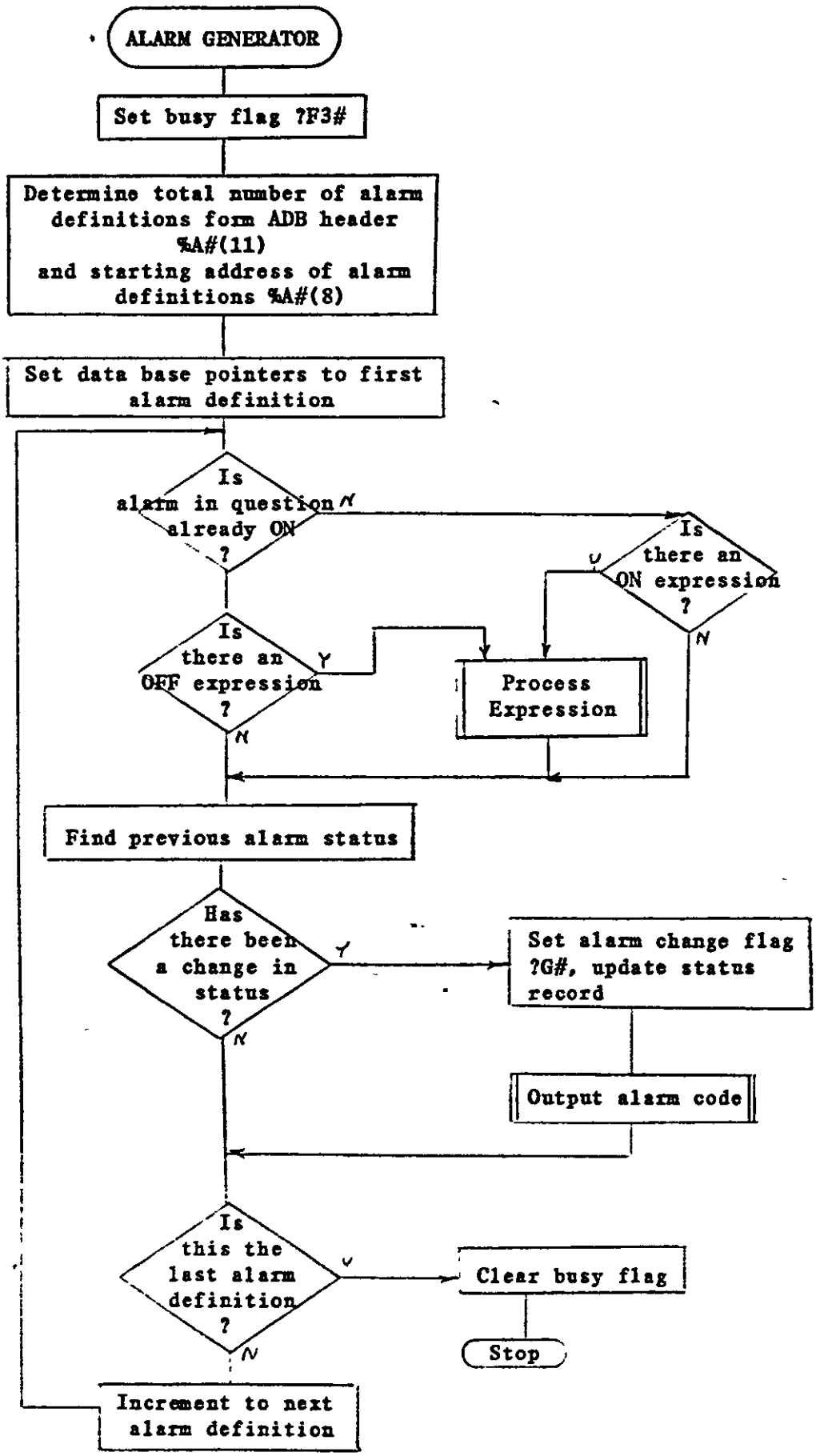

Chrom
send text to Chromatics
Obtain text from $N
and place in $M
Chromatics link
send $M to Chromatics
Return

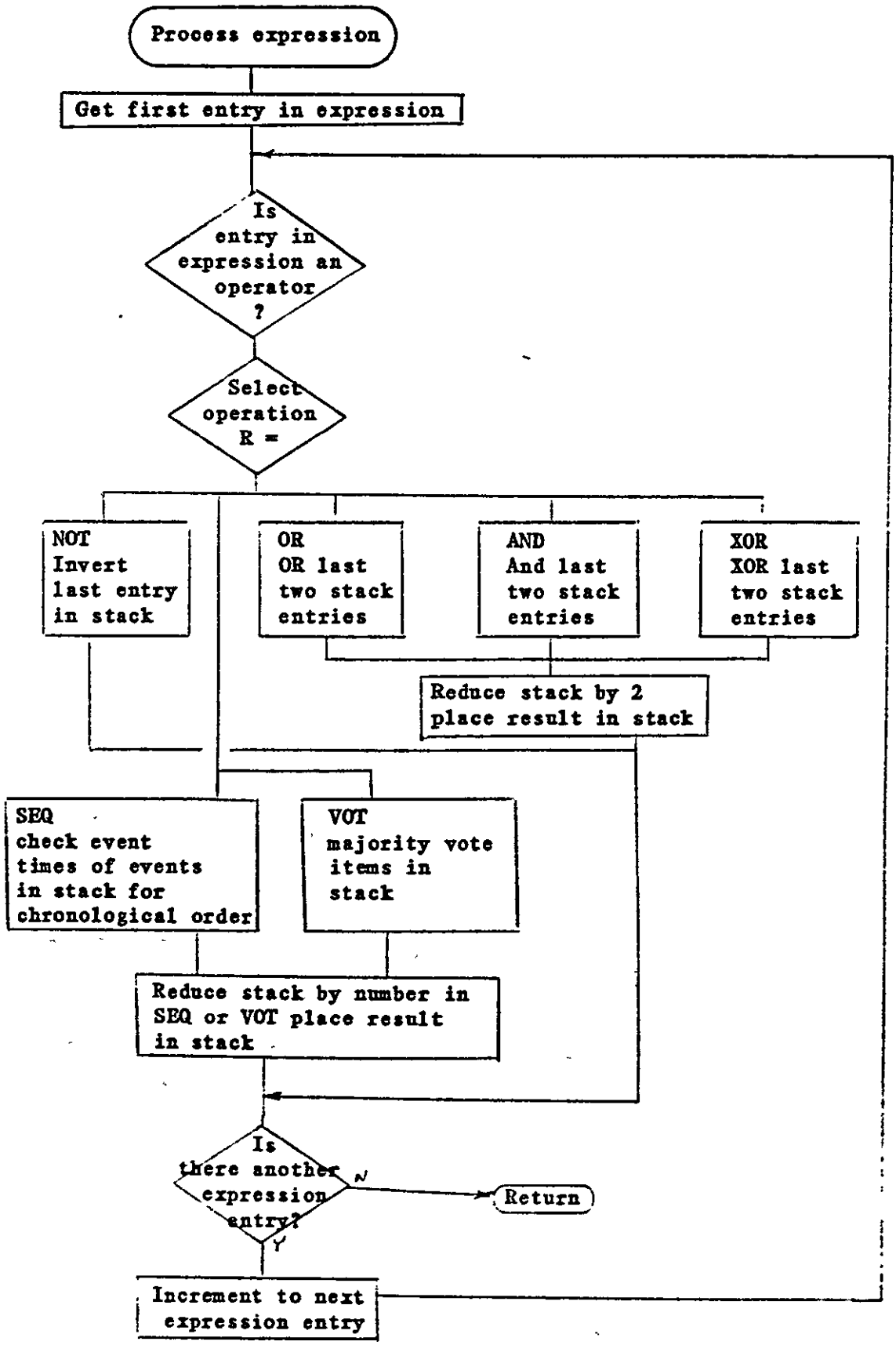

Send to
Another Q
Obtain message to be placed
in Q system from $N and
place in $M
OUTQ
Return

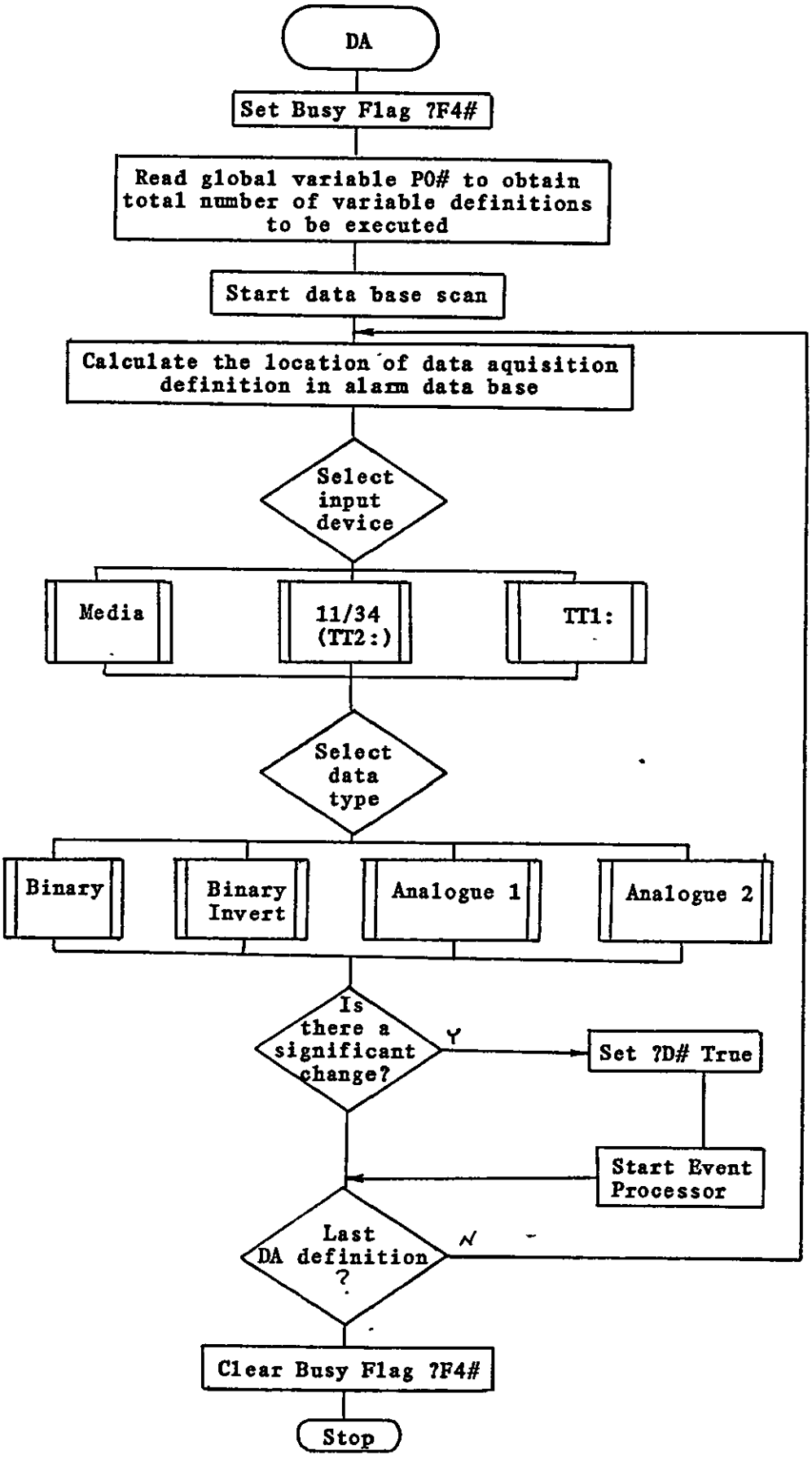

T43 Printer
send text to TT1:
Obtain text from $N and
place in $M
Open output on@T: for TT1:
Print $M to TT1:
Close output
Return

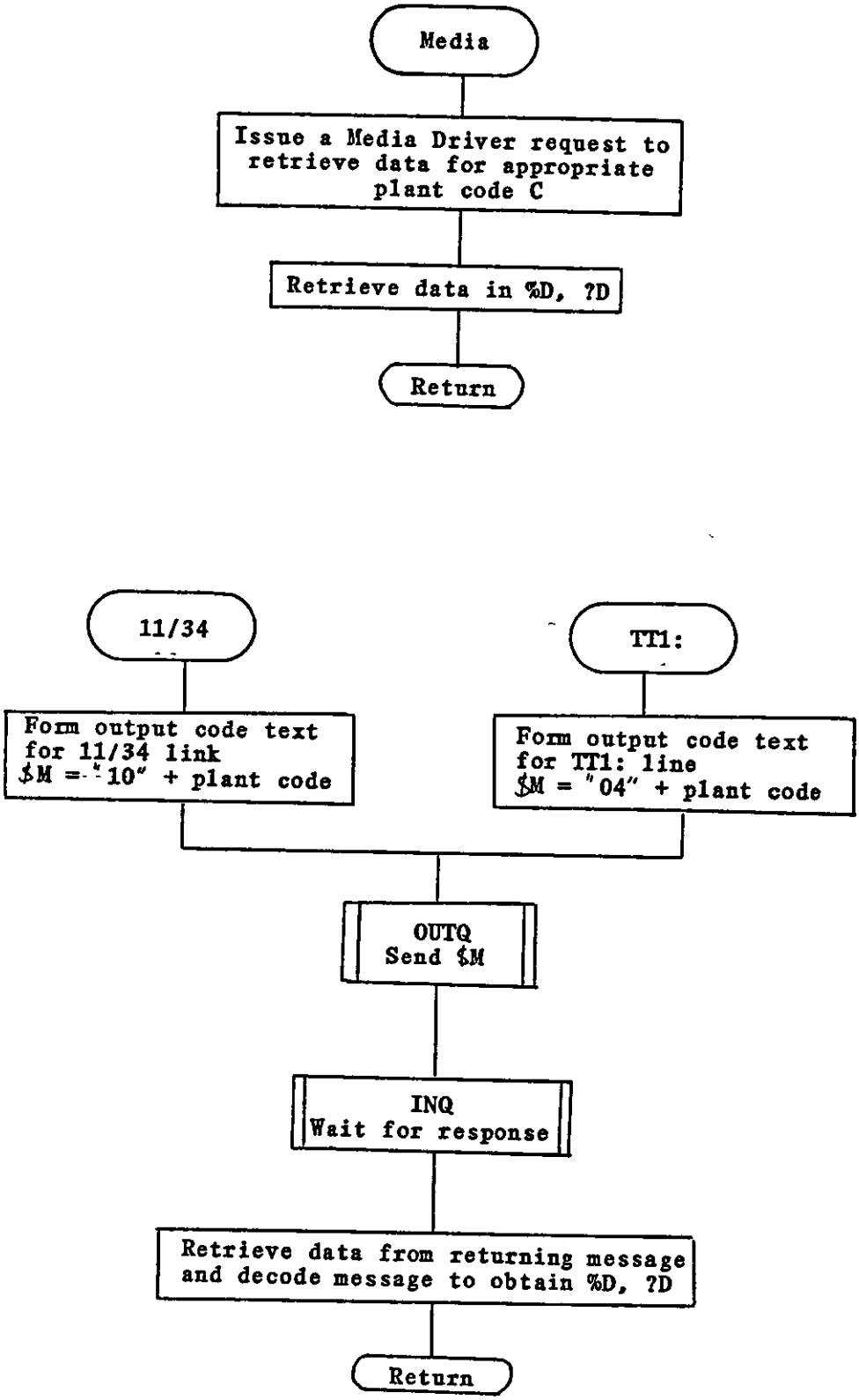

Chromatics Link
send data to Chromatics
Is
$M
empty
?
Y
N
Is
the Chromatics
link request
busy flag ?M#
true ?
Y
Wait 2 ticks
N
Set $M# = $M
?M# = TRUE
Return

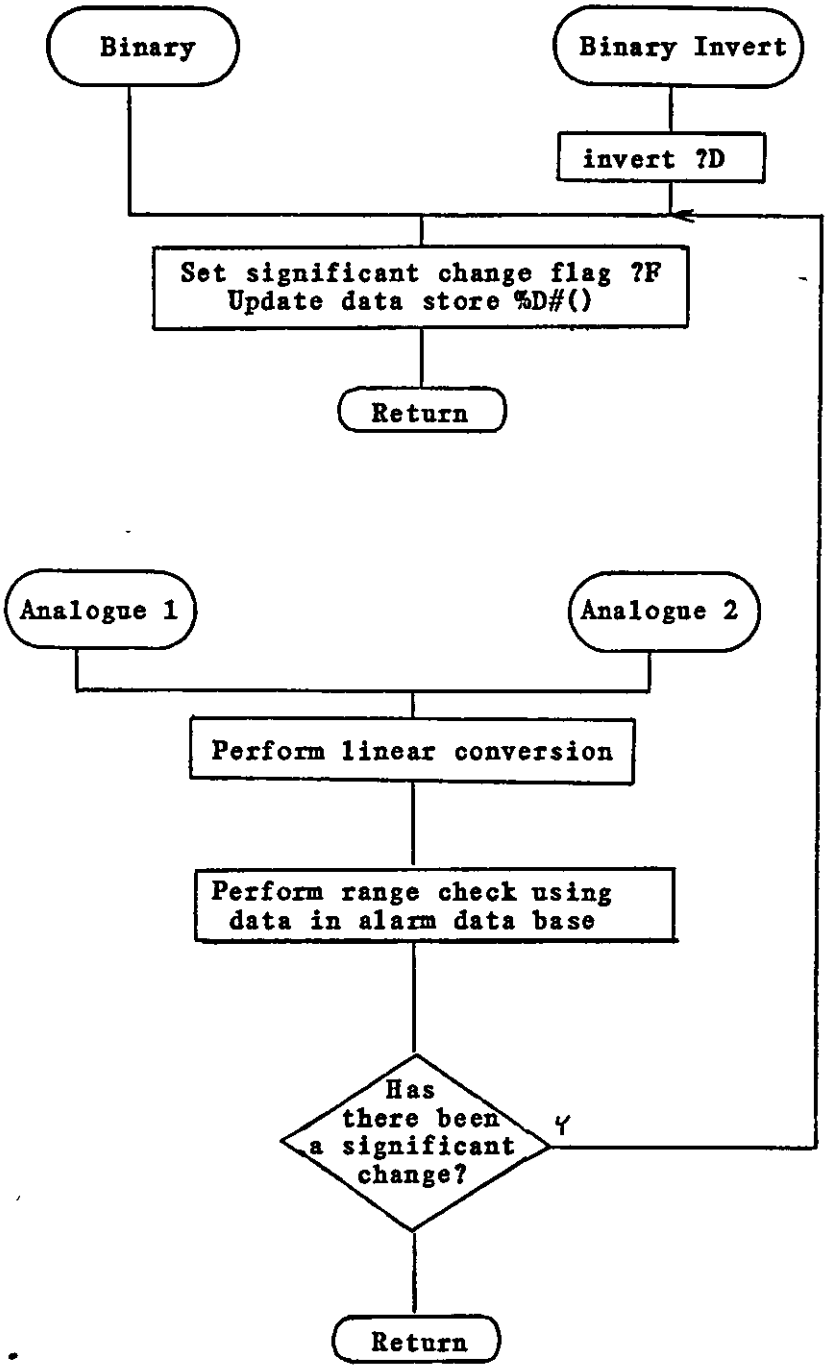

Output Alarm Code
Was result of alarm expression True?
N
Set first character in alarm output code $A = '1'
Y
Set first character in alarm output code $A = "2"
Is link to Display task busy, ?M =T
Y
Start Display task
Wait 2 ticks
N
Set link transfer variable $M1# to $A + output code in alarm definition
Set link request ?M1# = TRUE
Return


ALARM GENERATOR
Set busy flag ?F3#
Determine total number of alarm definitions form ADB header %A#(11) and starting address of alarm definitions %A#(8)
Set data base pointers to first alarm definition
Is alarm in question already ON ?
N
Y
Is there an ON expression ?
Y
N
Is there an OFF expression ?
Y
N
Process Expression
Find previous alarm status
Has there been a change in status ?
Y
N
Set alarm change flag ?G#, update status record
Output alarm code
Is this the last alarm definition ?
Y
N
Clear busy flag
Stop
Increment to next alarm definition


Process expression
Get first entry in expression
Is entry in expression an operator ?
Select operation R =
NOT Invert last entry in stack
OR OR last two stack entries
AND And last two stack entries
XOR XOR last two stack entries
Reduce stack by 2 place result in stack
SEQ check event times of events in stack for chronological order
VOT majority vote items in stack
Reduce stack by number in SEQ or VOT place result in stack
Is there another expression entry?
N
Return
Y
Increment to next expression entry


DA
Set Busy Flag ?F4#
Read global variable P0# to obtain total number of variable definitions to be executed
Start data base scan
Calculate the location of data aquisition definition in alarm data base
Select input device
Media
11/34 (TT2:)
TT1:
Select data type
Binary
Binary Invert
Analogue 1
Analogue 2
Is there a significant change?
Y
Set ?D# True
Start Event Processor
Last DA definition ?
N
Clear Busy Flag ?F4#
Stop


Media
Issue a Media Driver request to retrieve data for appropriate plant code C
Retrieve data in %D, ?D
Return
11/34
TT1:
Form output code text for 11/34 link $M = "10" + plant code
Form output code text for TT1: line $M = "04" + plant code
OUTQ
Send $M
INQ
Wait for response
Retrieve data from returning message and decode message to obtain %D, ?D
Return


Binary
Binary Invert
invert ?D
Set significant change flag ?F
Update data store %D#()
Return
Analogue 1
Analogue 2
Perform linear conversion
Perform range check using
data in alarm data base
Has
there been
a significant
change?
Y
Return