/N-61
33609
p.64

NASA Contractor Report 187171

Isothermal Thermogravimetric Data Acquisition Analysis System

Kenneth Cooper, Jr.
Transylvania University
Lexington, Kentucky

August 1991

Prepared for
Lewis Research Center

NASA
National Aeronautics and
Space Administration

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE August 1991	3. REPORT TYPE AND DATES COVERED Final Contractor Report

4. TITLE AND SUBTITLE
Isothermal Thermogravimetric Data Acquisition Analysis System

5. FUNDING NUMBERS

6. AUTHOR(S)
Kenneth Cooper, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Transylvania University
Brown 315
Lexington, Kentucky 40508

8. PERFORMING ORGANIZATION REPORT NUMBER
E-6474

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135-3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR-187171

11. SUPPLEMENTARY NOTES
Project Manager, James L. Smialek, Materials Division, NASA Lewis Research Center, (216) 433-5500.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The description of an Isothermal Thermogravimetric Analysis (TGA) Data Acquisition System is presented. The system consists of software and hardware to perform a wide variety of TGA experiments. The software is written in ANSI C using Borland's Turbo C^{++}. The hardware consists of a 486/25 MHz machine with a Capital Equipment Corporation IEEE488 interface card. The interface is to a Hewlett Packard 3497A data acquisition system using two analog input cards and a digital actuator card. The system provides for 16 TGA rigs with weight and temperature measurements from each rig. Data collection is conducted in three phases. Acquisition is done at a rapid rate during initial startup, at a slower rate during extended data collection periods, and finally at a fast rate during shutdown. Parameters controlling the rate and duration of each phase are user programmable. Furnace control (raising and lowering) is also programmable. Provision is made for automatic restart in the event of power failure or other abnormal terminations. Initial trial runs were conducted to demonstrate system stability. Extensive parameter variation between runs, many simultaneous runs, simulation of power outages have demonstrated system stability and reliability under a variety of operating conditions. This system has improved on the prior one in these main areas:

A. Recover from abnormal termination conditions with no loss of data.
B. Preprogramming all phases, allowing unattended startup and shutdown of a run.
C. Ease of operation - utilizing disk based systems as opposed to a tape based system.
D. Ready availability of data during a run.

14. SUBJECT TERMS
Thermogravimetry; Metal oxides; Oxidation; Data acquisition

15. NUMBER OF PAGES
64

16. PRICE CODE
A04

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRAC

TABLE OF CONTENTS

ISOTHERMAL THERMOGRAVIMETRIC ANALYSIS DATA ACQUISITION SYSTEM

Kenneth Cooper, Jr.[*]
Transylvania University
Lexington, KY 40508

ABSTRACT

The description of an Isothermal Thermogravimetric Analysis (TGA) Data Acquisition System is presented. The system consists of software and hardware to perform a wide variety of TGA experiments.

The software is written in ANSI C using Borland's Turbo C^{++}. The hardware consists of a 486/25MHz machine with a Capital Equipment Corporation IEEE488 interface card. The interface is to a Hewlett Packard 3497A data acquisition system using two analog input cards and a digital actuator card.

The system provides for 16 TGA rigs with weight and temperature measurements from each rig. Data collection is conducted in three phases. Acquisition is done at a rapid rate during initial startup, at a slower rate during extended data collection periods, and finally at a fast rate during shutdown. Parameters controlling the rate and duration of each phase are user programmable. Furnace control (raising and lowering) is also programmable. Provision is made for automatic restart in the event of power failure or other abnormal terminations.

Initial trial runs were conducted to demonstrate system stability. Extensive parameter variation between runs, many simultaneous runs, simulation of power outages have demonstrated system stability and reliability under a variety of operating conditions. This system has improved on the prior one in these main areas:

A. Recovery from abnormal termination conditions with no loss of data.
B. Preprogramming all phases, allowing unattended startup and shutdown of a run.
C. Ease of operation - utilizing disk based systems as opposed to a tape based system.
D. Ready availability of data during a run.

[*]NASA Lewis-ASEE Summer Faculty Fellow 1991.

DEFINITIONS

I. The purpose of the system is the acquisition of isothermal Thermogravimetric analysis (TGA) data. At the present time it is not designed for cyclic operation.

II. The system consists of software written in ANSI C, hardware consisting of a 486 clone machine, and an interface card to the data acquisition apparatus. A channel is a single apparatus and two voltage readings corresponding to weight and temperature values.

III. The data is analyzed offline. Typical analysis includes importing the resultant ASCII data files into a spreadsheet, i.e. Lotus 1-2-3, and preparing graphics. The data may also be imported into a number of data analysis programs for curve fitting, etc.

IV. There are two acquisition modes:

A. FAST ACQUISITION - The sampling period is measured in seconds and the total sampling time is measured in minutes. A fast acquisition mode precedes and succeeds a normal acquisition mode.

B. NORMAL ACQUISITION - The sampling period is measured in minutes and the total sampling time is measured in hours. The total sampling time may be preprogrammed to a specified value for completely automatic operation. In addition, the ability to 'look' at the data during a run allows the researcher to terminate a run at their discretion.

SYSTEM HARDWARE AND SOFTWARE

I. The working system directory is **C:\ACQUIRE** and contains **main.exe** and **startup.exe**. The data save subdirectory, **C:\ACQUIRE\SAVE_DAT**, is discussed in **SYSTEM OPERATIONS, Disk operations**. The working directory for program modification is **C:\BORLAND\TC\SOURCE**.

II. **HARDWARE:** The hardware implementation consists of three components, computer, data acquisition equipment, and interface board.

A. The computer is a 486/25 MHz machine with 4 MB of RAM, a 122 MB hard disk (C:), and 5 $^1/_4$" (A:) and 3 $^1/_4$" (B:) diskettes. DOS 5.0 is the operating system. The system, theoretically, could be implemented on a 286 machine. However, it is very unlikely that 16 channels could be supported. In addition, future modifications to the system, i.e. graphics, could not be supported. As a minimum, a 386/25 MHz machine is recommended.

B. The data acquisition equipment is a Hewlett Packard 3497A with two analog input (AI) cards and a digital actuator (DA) card. The first AI card, slot 0, acquires weight measurements. The second AI card, slot 1, collects temperature information and incorporates hardware temperature compensation. Typical input ranges are the order of hundredths of a volt, and less, for both cards. The DA card is in slot 3. Sixteen (0-15) of the twenty actuators are used. The relay systems for furnace control, controlled by the DA card, are designed inhouse.

C. The interface card is a Capital Equipment Corporation (CEC) PC488 IEEE-488 (HPIB) card. It provides for connection of an IEEE-488 device to a PC. CEC may be contacted at 99 South Bedford Street #107; Burlington MA 01803.

III. **SOFTWARE:** The software is written and compiled in ANSI C using Borland TURBO C^{++}, 2nd edition. In addition IEEE488.LIB and IEEE-C.H from CEC are required. Both of these files are supplied by CEC on their distribution diskette. The **LARGE** memory model as well as the 80x87 floating point and instruction set is used. The program is compiled as a PROJECT file, MAIN.PRJ, which contains **main.c** and IEEE488.LIB. The project file is contained included in **C:\BORLAND\TC\SOURCE**. IEEE488.LIB must be in the **C:\BORLAND\TC\LIB** directory. The IEEE-C.H header file must be in the **C:\BORLAND\TC\INCLUDE** directory. In addition to TURBO C^{++} the compilation requires access to TASM.EXE (Turbo Assembler). TASM.EXE should be in the **C:\BORLAND\TC\SOURCE\BIN** directory.

SYSTEM CAPABILITIES

I. The system has a 16 channel capacity.

II. The default times associated with each acquisition mode are identified and discussed in the **SYSTEM OPERATION** section of this manual. The default times may be respecified by the user on channel startup:

A. Actual data collection times are two seconds less than the sampling period.

B. Normal acquisition mode acquisition time is 28 seconds starting at the beginning of the sampling period.

III. The fast acquisition mode has two additional 'features' associated with it.

A. Fast acquisition is done at the start *and termination* of all runs. The parameters used for the termination fast acquisition are the same as for the startup fast acquisition.

B. A 'delay time' has been implemented into the fast acquisition mode. This is the number minutes after fast acquisition starts that a digital signal is issued which may be used to raise or lower the furnace for the apparatus. The default delay time is specified as the integer portion of $^1/_3$ of the total fast acquisition time. For example, a 16 minute fast acquisition time will have a default delay time of 5 seconds.

IV. The type thermocouple may be respecified on startup and up to 60 characters of general comments may be stored with the channel data. Six types of thermocouples are specifiable with no program modification: E, J, K, R, S, and T. Temperature ranges for thermocouples are specified in **SYSTEM OPERATIONS** section **S.IV.**

V. After the user specifies all startup parameters the system will automatically start itself, raise the furnace, run to completion, and lower the furnace. The preprogrammed normal sampling time may be manually overridden by means of the end channel option in the main menu.

VI. Data, as acquired during a run, is stored in individual files of the form **CHAN*cc*.DAT**. After a run terminates, the month, day, and hour are used to time stamp the data file giving it a 'unique' name, i.e. **CH*ccmmdd*.D*hh*.**

cc	- channel number
mm	- month run was started
dd	- day run was started
hh	- hour run was started

It is imperative for the system operator to maintain a log book of all runs started showing the starting date:time and other identifying information to facilitate finding a given data file.

VII. All data, as it is collected, is stored on the system hard disk. Data files for an individual channel, in process of acquisition or completed, may be copied to a diskette at any time. The resulting data files may imported as an ASCII file and analyzed by any spreadsheet.

VIII. In the event of abnormal termination, i.e. power failure, the system will restart all channels which were active and continue with the programmed parameters. An active channel has the name **CHAN*cc*.DAT.**

IX. The 3497A internal clock is synchronized with the PC CPU clock.

ADDITION OF DATA ITEM TO struct channel

To add an additional item to the structure **channel** the following functions must be considered. This may not be an all inclusive list depending on the purpose of the data item.

I. Change the *struct channel* definition, in the beginning of the program, to include the new data item.

II. Set the default value for the data item, along with the other defaults, in function **main()**.

III. To initialize the data item:
  A. If the data item requires no input/response from the user it is initialized in **get_startup_data()**. Typical of this type of data are filenames the ending time of the acquisition, etc.
  B. If the user must be prompted for input this is handled by the function **get_channel_data()**. Typical of this type of data is the length, in hours, of the run.
  C. If the data item is one which is written to the output data file for subsequent use by the automatic startup section of the program, the function **get_startup_data()** must be referenced.

IV. **setup_data()**, inputs data from the output data file during automatic startup, will need to be modified if **get_startup_data()** was used to output the data item to the channel's data file.

V. All channels have their default values reset in **terminate_channel()** to the same values specified in **main()**. Therefore, this function must be referenced.

SYSTEM OPERATION

The system consists of two programs - **main.c** and **startup.c**. Both of which are compiled by Borland Turbo C^{++} to their *.EXE files. Program **main.exe**, hereinafter referred as **main**, is the data acquisition program. **startup.exe** hereinafter referred to as **startup**, provides for unattended startup of **main** in the event of an abnormal system failure and may be used to start **main** upon system boot.

To implement **startup** the following lines must be placed in the **AUTOEXEC.BAT** file of the system machine:

> **CD *Absolute pathname to directory containing startup***
> **STARTUP**

For the present system the following lines:

```
CD \ACQUIRE
STARTUP
```

would be placed at the *end* of the **AUTOEXEC.BAT** file! In addition, **startup** and **main** MUST be in the same directory. Another method of starting the program would be to place the above two lines in a batch file, i.e. TGA.BAT, and simply execute the batch file.

In the event of a boot the program **startup** is always executed and displays the following message:

```
Press a/A to start data acquisition
```

If any key, other than a/A, is pressed the system returns the DOS prompt and you are in the directory containing the programs. If the a/A is pressed, or no key is pressed for 15 seconds **main** is invoked via execv() from **startup**.

Another way to start the data acquisition program is from the DOS command line. Switch to the directory containing **main**, via a CD DOS command, then invoke **main** from the command line by entering:

> C:*directory containing startup*> **main**

For example, the following DOS commands perform this function on the present system:

```
CD \ACQUIRE
MAIN
```

On startup, the first function performed by **main** is to check for the presence of the file **NORMAL**. This file is created by the orderly termination of **main**. If **NORMAL** is not present **main** assumes an abnormal termination, i.e. power failure, and invokes its restart recovery function using the file(s) **CHANcc.DAT**. If none of these files are present **main** displays the main menu and waits for user input. If any of these files are present, main will process the respective channels, *cc*, before displaying the main menu. Processing of channels on restart recovery may involve a single normal acquisition or a normal acquisition followed by a fast acquisition, in the event termination time for the channel expired during abnormal termination.

Abnormal system termination is indicated by the absence of the file **NORMAL** and the presence of the files **CHANcc.DAT**. Normal startup <u>always</u> occurs if the file **NORMAL** is present. Therefore, if a file with this name is in the **startup/main** directory normal startup will occur. If it is decided to manually create NORMAL a decision must also be made as to what to do with the files CHAN*cc*.DAT. They may be deleted, saved and dumped for subsequent analysis, or left as is. In the latter case the system will overwrite these files the next time channel *cc* is started. In general, this function should ONLY handled by the system operator.

Once **main** has been started and displays the MAIN MENU any one of the following four options may be selected.

D **Disk operations:** The option provides for the dumping of acquired data to a diskette for subsequent analysis. This option requires a preformatted 5 $^1/_4$" diskette to be placed in the A: drive, on the present machine. If the A: drive is changed to a 3 $^1/_2$" drive then the corresponding diskette must be used. Either high or double density diskettes may be used after being properly formatted. Capacities of the two sizes are as shown below:

DISK CAPACITIES (HOURS)*

Disk size	Double Density	High Density
5 $^1/_4$	690	2300
3 $^1/_2$	1380	2760

* - Assuming nor_delta_t = 6 min and fast_total_time = 30 min

The first prompt is:

WHICH CHANNEL DO YOU WANT(* for all):

At this point the user may request all files be dumped or only the files for a specified channel.

I. If an asterisk is entered, all files of the form CH*ccmmdd*.D*hh* and CHAN*cc*.DAT are dumped. As each file is dumped its name is displayed.

II. The second option is to simply enter the channel number of the desired channel. In this case all files are dumped wherein *cc* is the desired channel number. For example, if channel 5 is to be dumped then all files of the form CH5*mmdd*.Dhh and CHAN5.DAT will be dumped and their names displayed.

After the channel(s) has(have) been specified the following prompt is displayed:

INSERT A PREFORMATTED DISKETTE IN DRIVE A:
PRESS <ENTER> WHEN READY

After the diskette has been inserted in the A: drive and the <ENTER> key pressed all requested files are dumped. If an error occurs, unformatted diskette, diskette full, etc., a condition specific error message is displayed and the system returns to the MAIN MENU.

When a specified channel is dumped, see II above, all files of the form CH*ccmmdd*.D*hh* are dumped to the diskette and then moved to a subdirectory, SAVE_DAT, of the program directory. The files in SAVE_DAT may only be accessed from the DOS prompt. NO FILES MAY BE DELETED FROM WITHIN **main**.

Therefore, a point to be addressed is the 'cleaning out' of old data files. Again, this must be done from the DOS prompt. Care must be taken to not delete files which have not been backed up. It is suggested that all files be dumped as needed, backed up on a periodic basis and the system purged of all unneeded data files. Eventually, if this is not done, enough data files will accumulate so that a diskette will not be able to contain them. Whatever method is used, it should be done on a regular basis in a systematic manner as determined by the system operator. REMEMBER THERE IS NO SUBSTITUTE FOR AN ADEQUATE BACKUP SYSTEM!! It is a lot easier to recover data from a backup diskette than to rerun a sample.

E **End channel acquisition:** This option is used to terminate a channel prior to its scheduled termination time. As indicated in option S, section III.B. indefinite times may be specified for the normal acquisition. This option would be used to terminate a channel in this instance. The prompt:

Channel number to terminate:

is displayed and the channel number to terminate is entered. If an invalid channel or an inactive channel is specified, an error message is specified and the user is returned to the MAIN MENU.

S **Start channel acquisition:** Typically the first option selected. This invokes the channel startup function. For the following entries the characters accepted may only be a minus (-), and numeric characters (0-9). Note that a backspace does not fall into this category and will be ignored! Permitted values are specified in the table below:

VARIABLE VALUES

Variable name	Minimum	Maximum	Default
channel	0	15	none
fast_delta_t	10 sec	∞ sec	30 sec
fast_total_time	1 min	∞ min	15 min
fast_delay_time	0 min	**fast_total_time**	**int $^1/_3$ fast_total_time**
nor_delta_t	1 min	∞ min	6 min
nor_total_time	1 hr	∞ hr	100 hr
thermocouple	E, J, K, R, S, T		R
Comments	0 char	60 char	0 char

I. First the channel to startup is specified:

A. If the specified channel is in use, the user will be informed, not permitted to continue with the channel and returned to the MAIN MENU.

B. If the user enters a negative number they will be returned to the MAIN MENU.

C. If the channel number is valid the succeeding entries on the screen will prompt them for the acquisition startup data.

II. Next the fast acquisition parameters are specified:

A. *Interval time(sec):* (**fast_delta_t**). This is the length of time between fast acquisition samples. Of this time, the actual sampling is done for **fast_delta_t** - 2 seconds. The 2 seconds gives the program adequate time to calculate and write the collected data to disk.

B. *Total acquisition time(min):* (**fast_total_time**) is the total length of time for fast acquisition expressed in minutes.

C. *Fast acquisition delay(min):* (**fast_delay_time**) This is the time, after fast acquisition starts, that a digital signal is made available through the digital actuator card (in slot 3), for furnace raising or lowering. Startup fast acquisition will raise the furnace and termination fast acquisition will lower the furnace. Each apparatus must be equipped with an automatic relay system controlled by this signal for this parameter to be of any use. If no automatic furnace control is available any delay time may be used and furnaces must be manually raised.

III. Next the normal acquisition process parameters are requested.

A. *Interval time(min)*, **nor_delta_t**. The length of time between normal samplings.

B. *Total acquisition time(hr)*, **nor_total_time**. Any time of 999, or any **LARGE** value, may be used for an indefinite time. Also see option **E** above.

IV. **TC**. The type of thermocouple is specified next as upper or lower case letters. Any characters other than these are ignored. Temperature ranges for the various thermocouples are given below:

THERMOCOUPLE TEMPERATURE RANGES (∘C)

Thermocouple	Minimum temperature	Maximum temperature	Error ± ∘C
E	-100	1000	0.5
J	0	760	0.1
K	0	1370	0.7
R	0	1760	0.5
S	0	1750	1.0
T	-160	400	0.5

V. **comment**. May consist of up to 60 characters. A backspace may be used to erase incorrect characters on input.

After the comment is entered and the <ENTER> key is pressed the user is prompted with the following message to check the data.

If the above is correct enter a y/Y otherwise <ENTER>:

If the user enters a y/Y the data is accepted and the acquisition process starts. If any other key is entered the entire data entry process is restarted, beginning with the channel number. Therefore, if the user decided during the data entry process that the desired channel was not ready, a negative value for the channel number would be entered and the program would return to the MAIN MENU.

If the data is accepted, the program enters the fast acquisition mode for the specified channel. The fast acquisition screen will display the channel number, current data:time, and temperature after each fast acquisition sampling. During the fast acquisition process, if the automatic relays have been installed, at the specified delay time the furnace will be raised. This requires 50-120 seconds depending on the apparatus. After fast acquisition, the program checks all other active channels to determine if any of them missed a normal acquisition during the fast acquisition. If normal acquisition was missed, it is done for the required channels and then the user is returned to the MAIN MENU. Further, the next normal acquisition sampling time is based on this most recent sampling time, i.e. current time + **nor_delta_t**.

X eXit **program**: Provides for an orderly termination of **main**. If ANY channels are active or an incorrect channel number is specified, an error message is displayed and the user is returned to the MAIN MENU. ALL channels MUST be terminated, either by preprogrammed time expiration or via option E before **main** may be terminated.

If no channels are active the prompt:

ARE YOU SURE (Y/y):

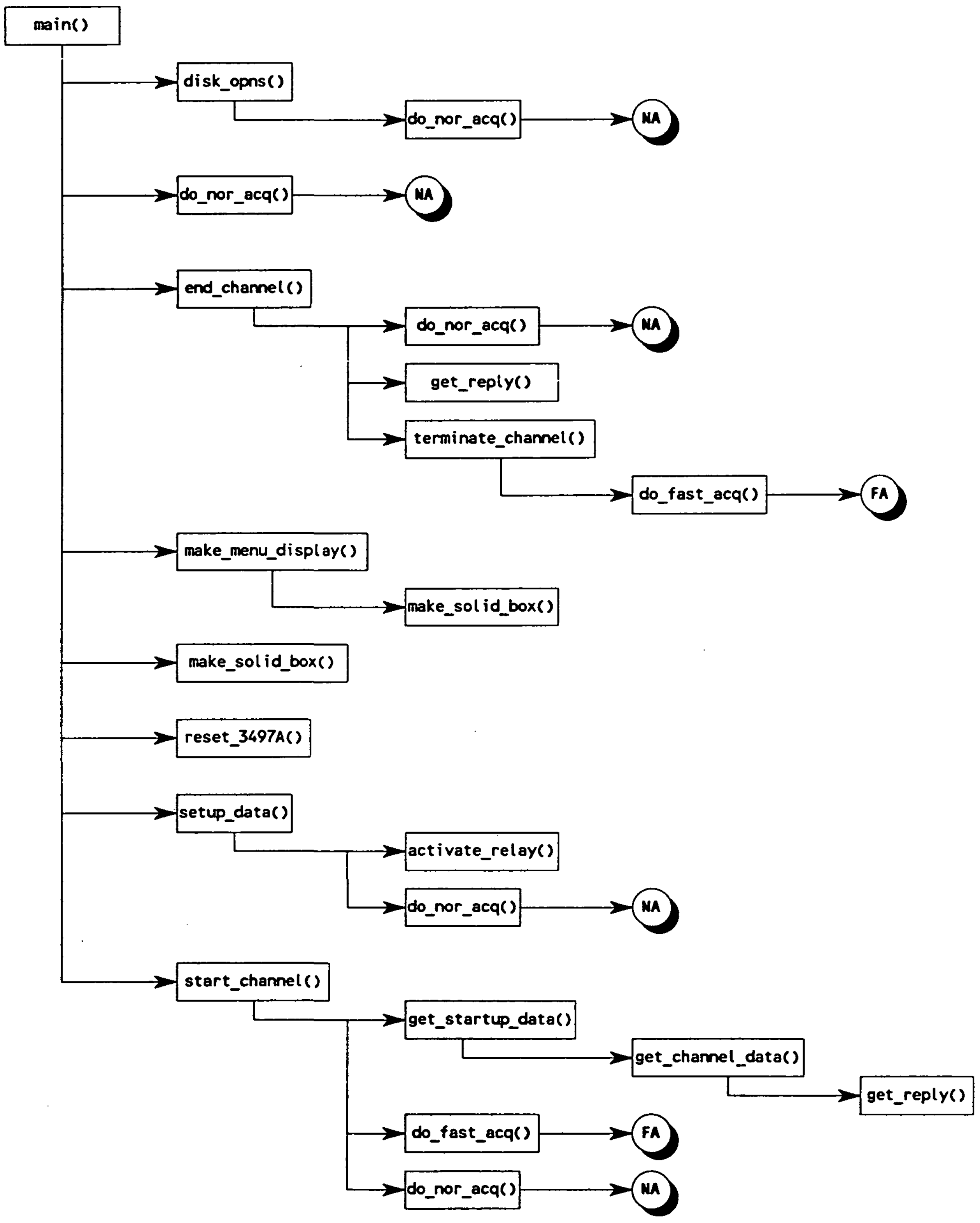
is displayed. If Y/y is entered **main** terminates and returns the user to the DOS prompt in the directory which contains **main**. Any other entry will return the user to the MAIN MENU.

PROGRAM MAIN.C

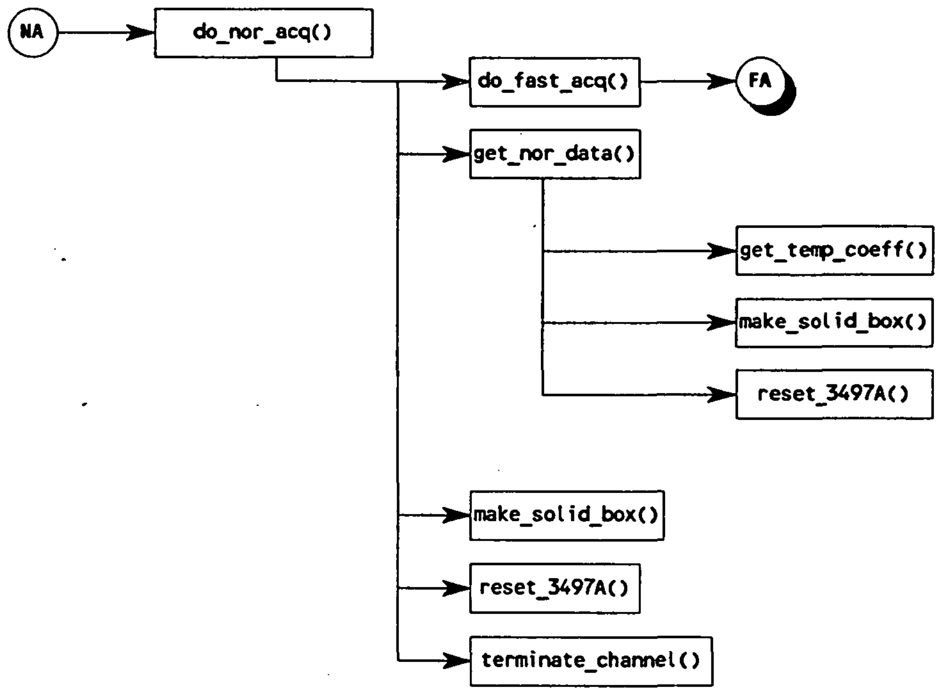
FUNCTION DESCRIPTIONS

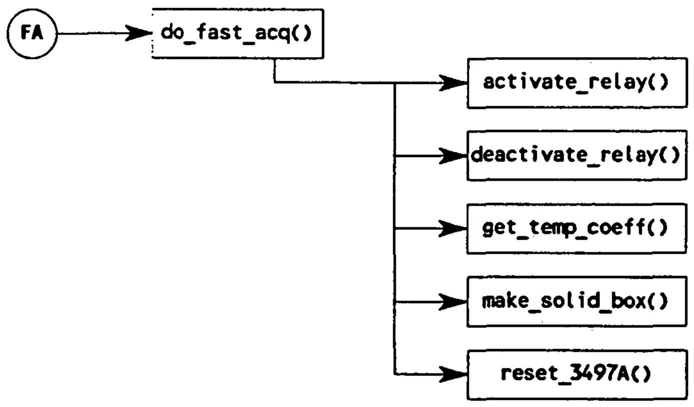
AND FLOWCHARTS

FUNCTION RELATIONSHIPS - 1/2



FUNCTION RELATIONSHIPS - 2/2





FUNCTION **activate_relay()**

CALLED BY: do_fast_acq()

CALLS:

PROTOTYPE: void activate_relay(int which_channel)

SEE: deactivate_relay()

This function simply creates a string of the form **DC*s*,*c*TD** which is used to close a digital relay. The *s* is the slot in the 3497A which contains the digital relay board and *c* is the channel number to be closed. Channels are numbered from 0 to 19 on a relay board, however, only 0 - 11 are used.

This signal is used to raise the furnace for the apparatus in accordance with **fast_delay_time** as discussed in **main()**.

No flowchart displayed.

FUNCTION **deactivate_relay()**

CALLED BY:	setup_data() do_fast_acq()
CALLS:	
PROTOTYPE:	void deactivate_relay(int which_channel)
SEE:	activate_relay()

This function simply creates a string of the form **DO*s*,*c*TD** which is used to open a digital relay. The *s* is the slot in the 3497A which contains the digital relay board and *c* is the channel number to be opened. Channels are numbered from 0 to 19 on a relay board, however, only 0 - 11 are used.

This signal is used to lower the furnace for the apparatus in accordance with **fast_delay_time** as discussed in **main()**.

No flowchart displayed.

FUNCTION disk_opns()

CALLED BY: main()

CALLS: void do_nor_acq(void)

PROTOTYPE: void disk_opns(void)

SEE:

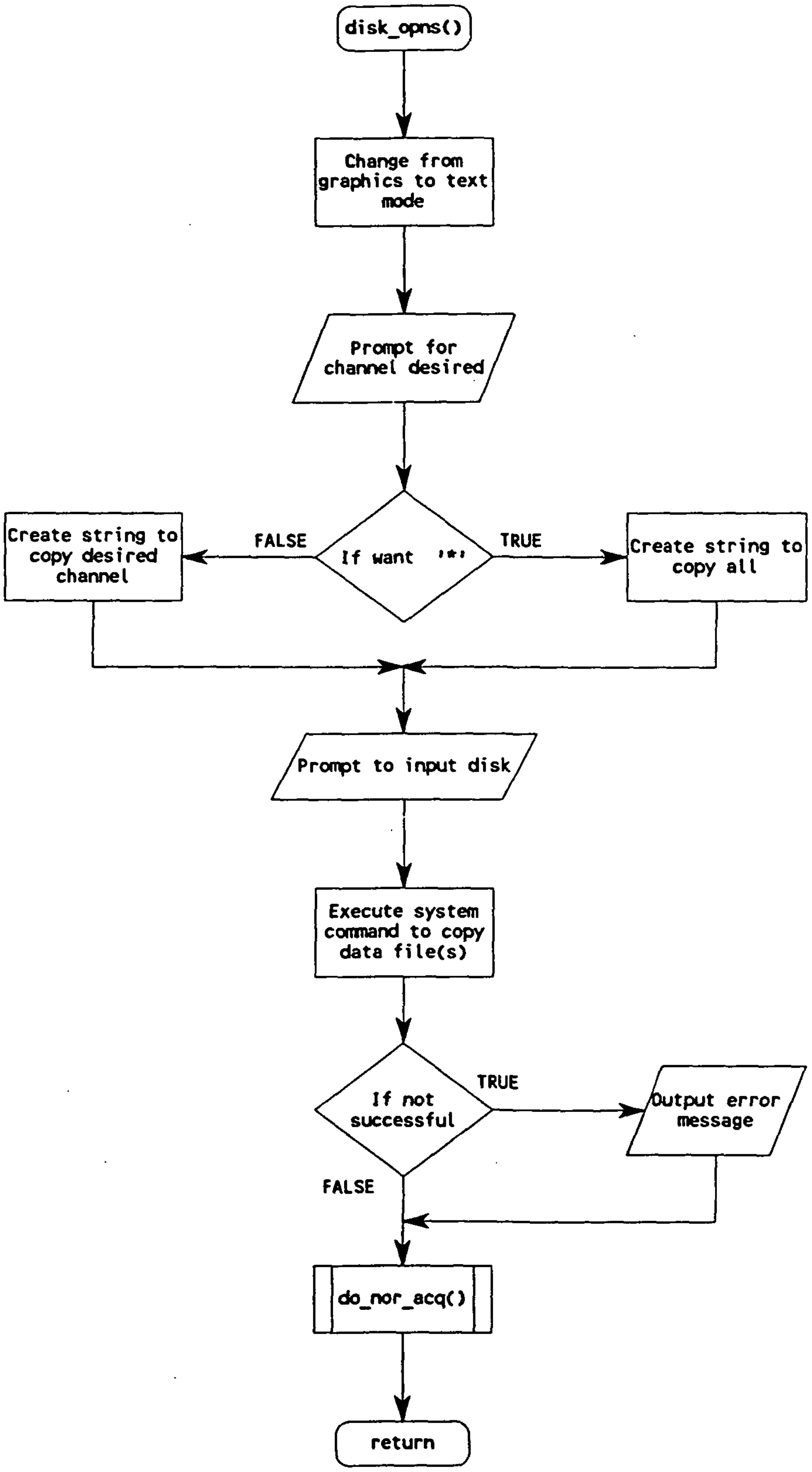
This function is used to copy, to the B: drive, the requested files. This may be changed to the A: drive by simply replacing **B:** with **A:** in this function and recompiling. At the present time the B: drive is a 1.44MB drive.

The user is prompted for the data to copy. If the user enters an '*' **ALL** data files are dumped to the diskette, i.e. *.DAT is copied. This will include files presently being acquired as well as all files previously collected. If the user enters a channel number, i.e. 5, then the data for that channel presently being acquired, i.e. CHAN5.DAT, is copied to the diskette.

Errors from the **system()** function may be generated by unformatted diskettes, disk full, etc. An appropriate error message is generated via **perror()**.

After the data is copied a normal acquisition is performed in the event a channel was to be acquired during the copy process. When the normal acquisition is finished the program returns to **main()** and redisplays the main menu.

disk_opns - 1/1



disk_opns()
Change from graphics to text mode
Prompt for channel desired
If want '*'
FALSE
TRUE
Create string to copy desired channel
Create string to copy all
Prompt to input disk
Execute system command to copy data file(s)
If not successful
TRUE
FALSE
Output error message
do_nor_acq()
return

FUNCTION do_fast_acq()

CALLED BY: do_nor_acq()
start_channel()
terminate_channel()

CALLS:
```
void activate_relay(int which_channel)
void deactivate_relay(int which_channel)
void make_solid_box(int lower_left_x,  int lower_left_y,
        int upper_right_x, int upper_right_y,
        int FILL_COLOR,    int BORDER_COLOR)
int reset_3497A(void);
```

PROTOTYPE: void do_fast_acq(channel channel_data[12], int channel)

SEE: main()
terminate_channel()

This function's purpose is to provide for a fast acquisition at the beginning and end of an experiment. In addition, the user may supply a 'delay time' which is used to determine when the furnace is raised or lowered, see below. For each acquisition point the data is written to disk immediately. Data is collected as specified in **main()** and the average of the datum is reported as a single point. The time of collection is specified as the end of the acquisition period for the point.

I. The function resynchonizes the 3497A clock to the CPU clock, see **main()**. Various initializations are performed, i.,e. the display, **end_sec**, **delay_sec**, etc. Two strings are constructed to perform an AI function. The first, **sendchan** is used to collect the weight while the second, **sendchan20**, is used to collect the temperature.

II. The following functions are performed while **end_sec** is greater than the current time.

  A. Then the TI function on the 3497A is activated and the function waits for the first interrupt from the 3497A in the form of a SRQ.

  B. When the SRQ is activated it is cleared, the fast acquisition screen is displayed and parameters are initialized. The variable **end_period_sec** is calculated as 2 seconds less than the **fast_delta_t** variable.

  C. The furnace is raised if the delay time has passed and the system is not INACTIVE. The furnace is lowered if the delay time has passed and the channel has been made INACTIVE.

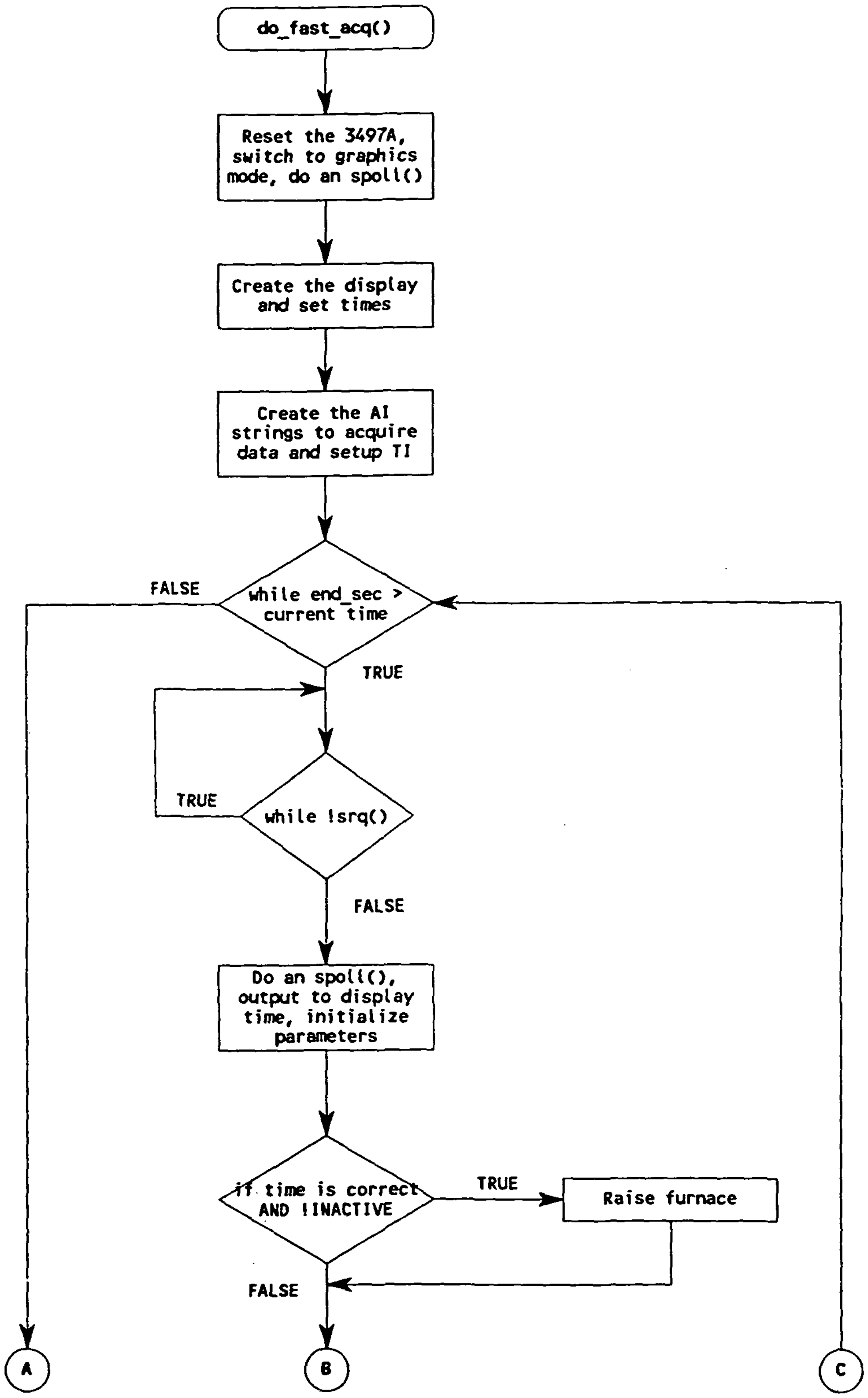
  D. Data is repetitively collected and summed as long as the end of the acquisition period exceeds the current time. After each **send** command the function delays 20ms to allow the 3497A voltmeter to stabilize prior to the **enter** command.

  E. After the data has been collected its average is determined and the temperature data is displayed. In addition the weight and temperature data is output to the requisite file.

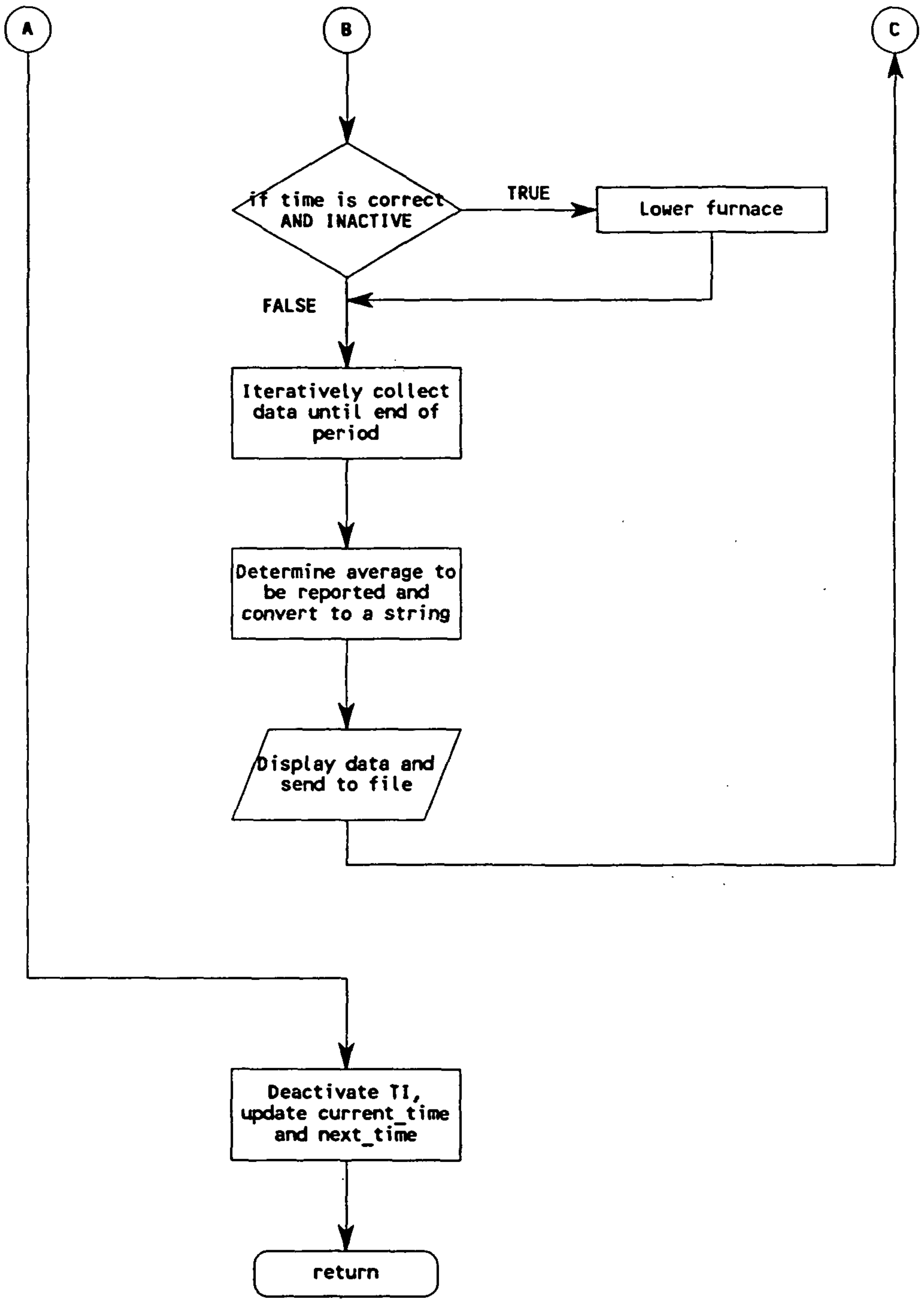
  F. This process is repeated from 2.

III. When the fast acquisition process is finished the TI is inactivated, the current time is updated and the next time for acquisition is determined and stored. The next time is the first normal acquisition time. In the case of a channel which is INACTIVE this is still calculated but ignored as the channel is terminated, see **terminate_channel()**.

do_fast_acq() - 1/2


do_fast_acq()
Reset the 3497A, switch to graphics mode, do an spoll()
Create the display and set times
Create the AI strings to acquire data and setup TI
while end_sec > current time
FALSE
TRUE
while !srq()
TRUE
FALSE
Do an spoll(), output to display time, initialize parameters
if time is correct AND !INACTIVE
TRUE
Raise furnace
FALSE
A
B
C

do_fast_acq() - 2/2


A
B
C
if time is correct AND INACTIVE
TRUE
Lower furnace
FALSE
Iteratively collect data until end of period
Determine average to be reported and convert to a string
Display data and send to file
Deactivate TI, update current_time and next_time
return

FUNCTION do_nor_acq()

CALLED BY: disk_opns()
end_channel()
main()
setup_data()
start_channel()

CALLS:
```
void do_fast_acq(struct *channel_data, int which_channel)
void get_nor_data(void)
void make_solid_box(int lower_left_x,  int lower_left_y,
        int upper_right_x, int upper_right_y,
        int FILL_COLOR,    int BORDER_COLOR)
int reset_3497A(void)
void terminate_channel(int channel)
```

PROTOTYPE: void do_nor_acq(void)

SEE: main()

This function's purpose is to provide for normal acquisition. In addition it checks for end of time for a channel and terminates the channel if necessary. It also determines the next channel to be sampled.

1. Any active TA are turned off and the SRQ is cleared, if it was active. Next the normal acquisition display is created and the 3497A clock is synchronized with the CPU.

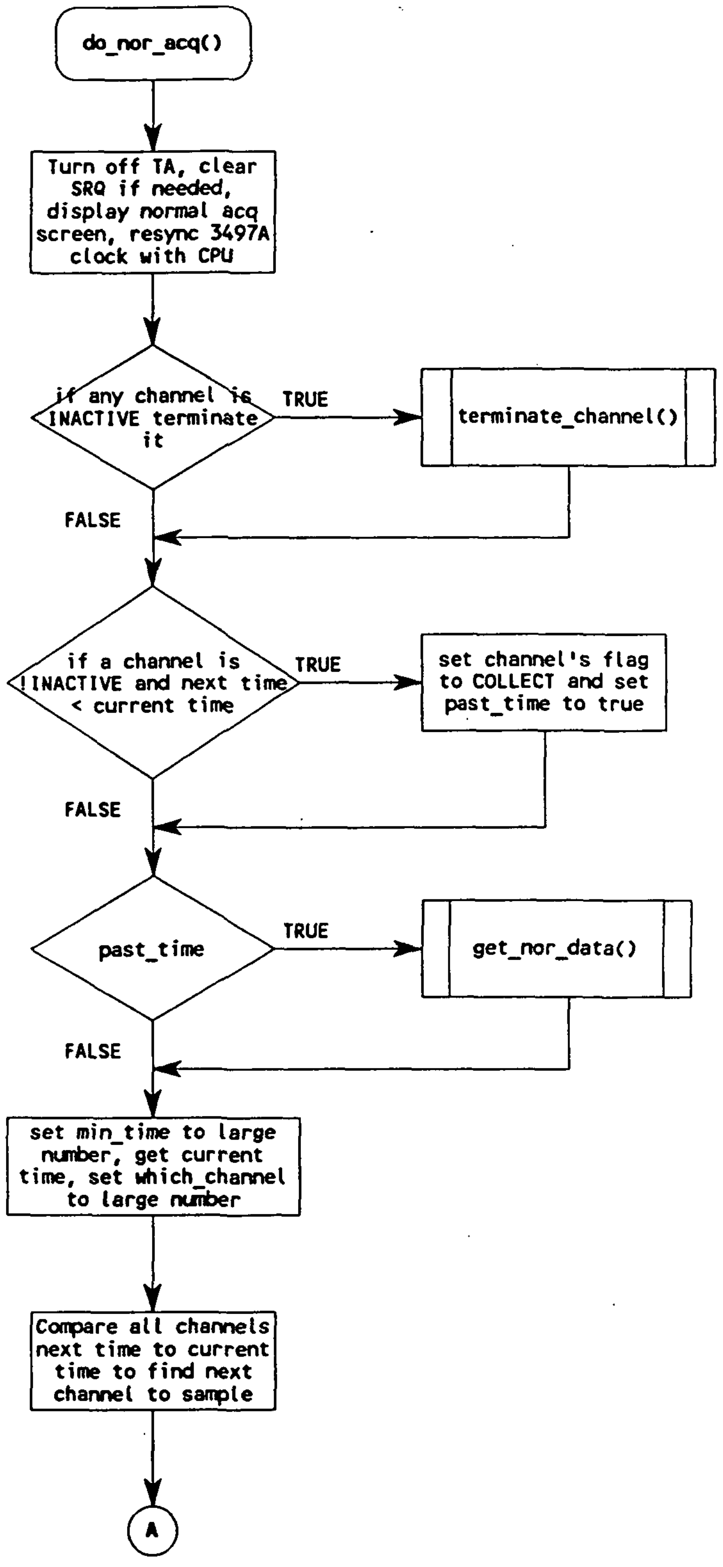
2. All channels are checked to determine if their termination time has expired. If it has these channels are terminated via **terminate_channel()**.

3. Next all channels are checked to see if their next_time is < or equal to the current time. If so **flag** is changed to COLLECT and the flag past_time is set to TRUE. If the flag past_time is TRUE then all channels whose **flag** has been set to COLLECT are sampled via **get_nor_data()**.

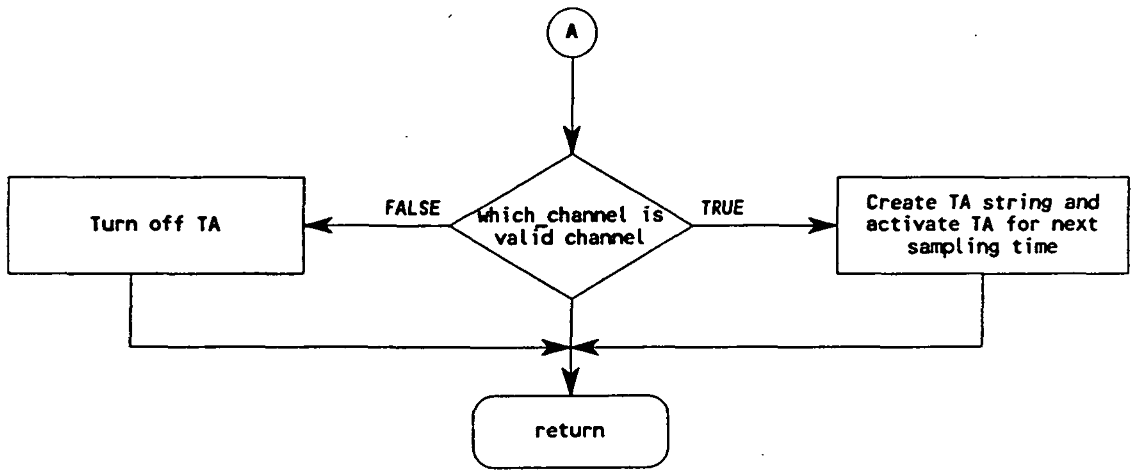
4. After sampling, the channel whose **next_time** is closest to the current time is selected. There may be more than one channel with the same **next_time**. This is irrelevant as the function is simply looking for the next time to sample, whether it is one or more channels.

5. If a channel is found, from step 4, Its **next_time** is used to determine the next TA for the 3497A. If no active channel was found TA is inactivated.

do_nor_acq() - 1/2


do_nor_acq()
Turn off TA, clear SRQ if needed, display normal acq screen, resync 3497A clock with CPU
if any channel is INACTIVE terminate it
TRUE
terminate_channel()
FALSE
if a channel is !INACTIVE and next time < current time
TRUE
set channel's flag to COLLECT and set past_time to true
FALSE
past_time
TRUE
get_nor_data()
FALSE
set min_time to large number, get current time, set which_channel to large number
Compare all channels next time to current time to find next channel to sample
A

do_nor_acq() 1 2/2


A
which_channel is valid channel
FALSE
TRUE
Turn off TA
Create TA string and activate TA for next sampling time
return

FUNCTION end_channel()

CALLED BY: main()

CALLS: void do_nor_acq(void)
int get_reply(void);
void terminate_channel(int which_channel)

PROTOTYPE: void end_channel(void)

SEE:

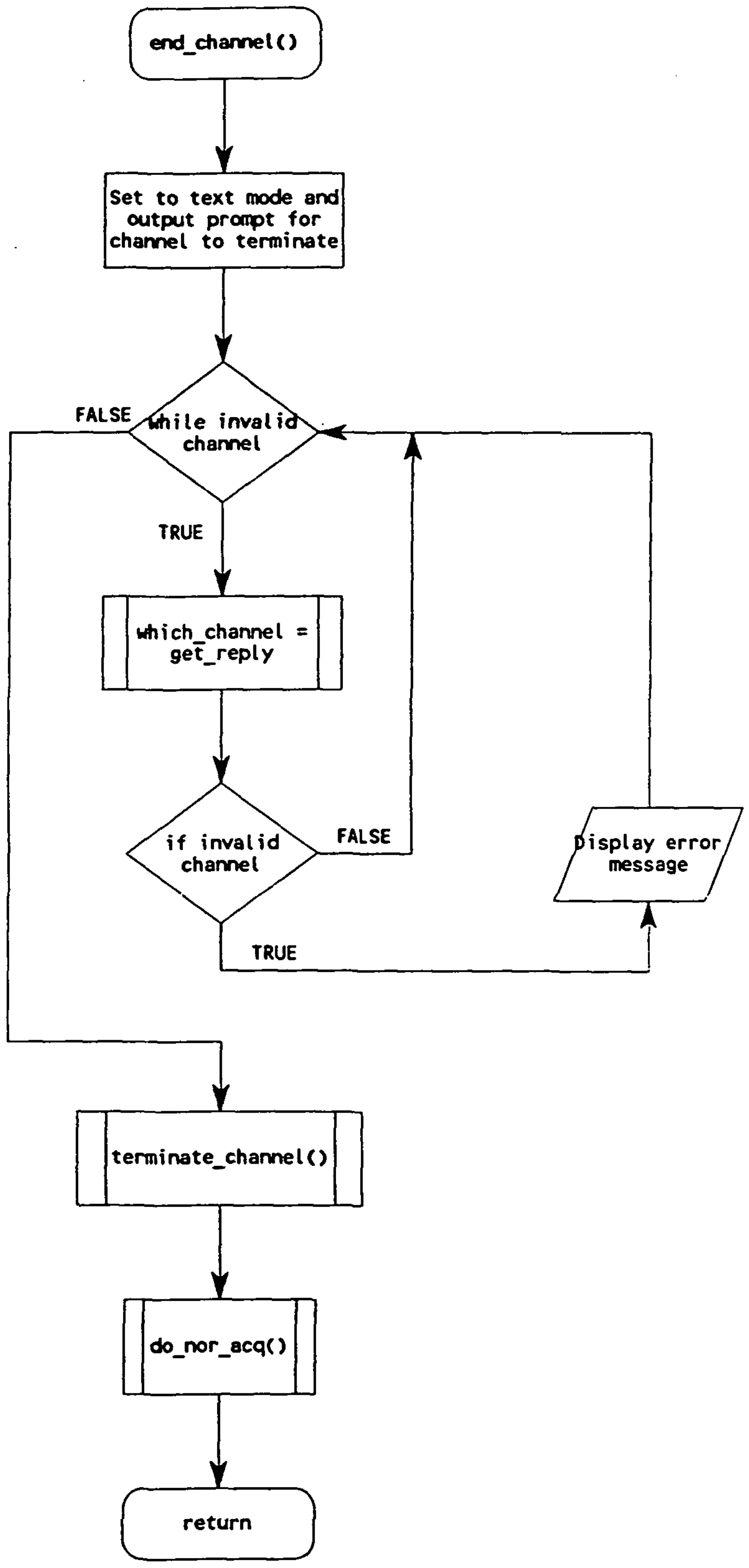
This function is called from the main menu of **main()** to terminate acquisition prior to the scheduled termination time for the channel.

1. The screen is set to text mode and the channel number is requested.

2. If the channel number is valid the channel is terminated via **terminate_channel()**.

3. A normal acquisition is requested via **do_nor_acq()**.

end_channel() - 1/1



FUNCTION get_channel_data()

CALLED BY:	get_startup_data()
CALLS:	int get_reply(void)
PROTOTYPE:	int get_channel_data(struct channel channel_data[12])
SEE:	main()

This function acquires user data values for the data structure channel_data for a given channel. The description of the individual fields are found in the description for **main()**.

1. The screen is set to text mode and the parameters screen is displayed.

2. First the desired channel number is requested. If it is an invalid number, outside the range of 0-11 inclusively, and error message is displayed and the user must reinput a valid number. If the channel has already been selected, again an error message is output and a valid channel must be input.

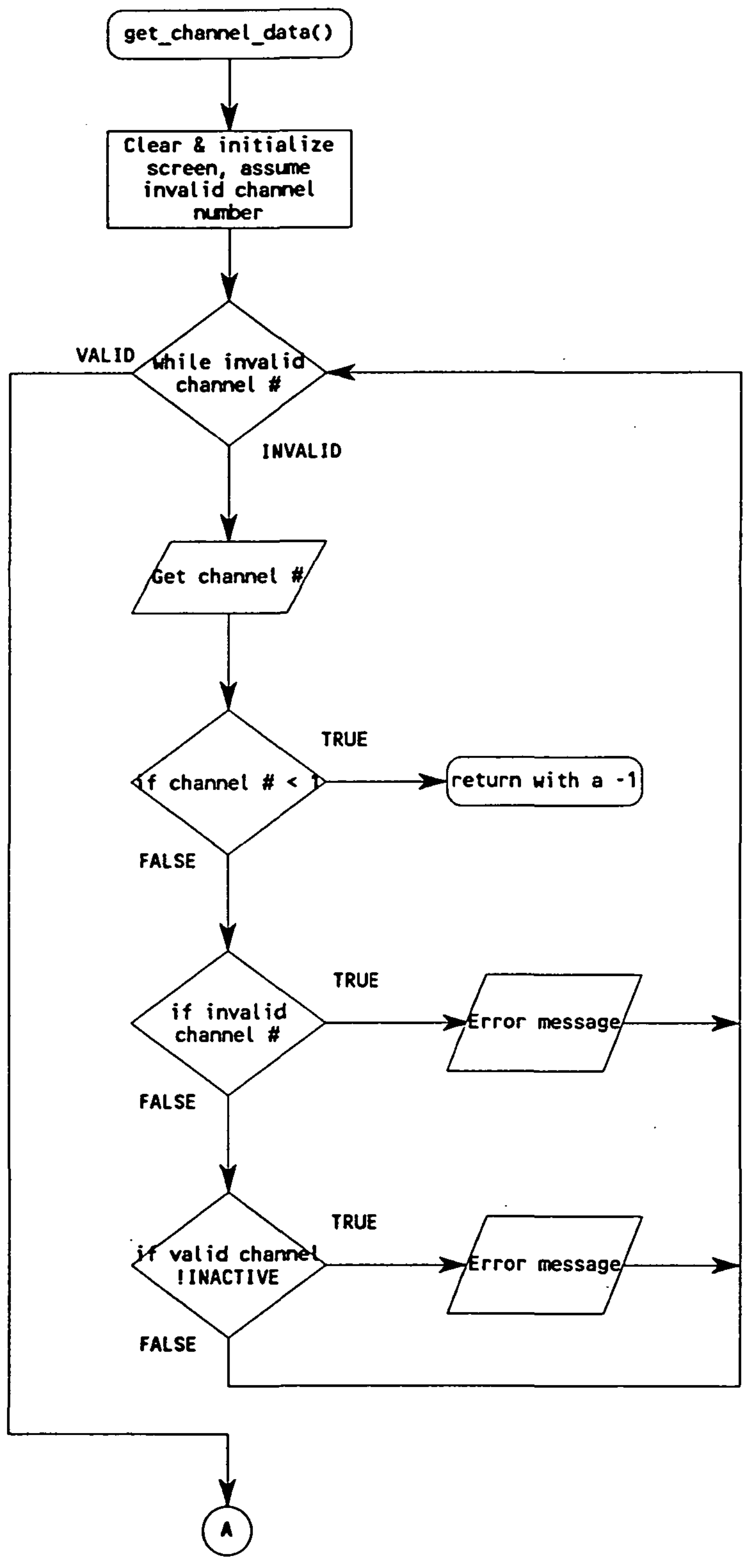
3. The next set of data requests deals with the fast acquisition process. The **fast_delta_t**, **fast_total_time**, and **fast_delay time** are requested. In each case the value is checked against its limits, see **main()**. If the input is outside its limits the default values are used. In the case of **fast_delay_time** a value less than **fast_total_time** <u>must</u> be entered.

4. Next the data values for the normal acquisition are requested. Values for **nor_delta_t** and **nor_total_time** are requested and handled as per 3 above.

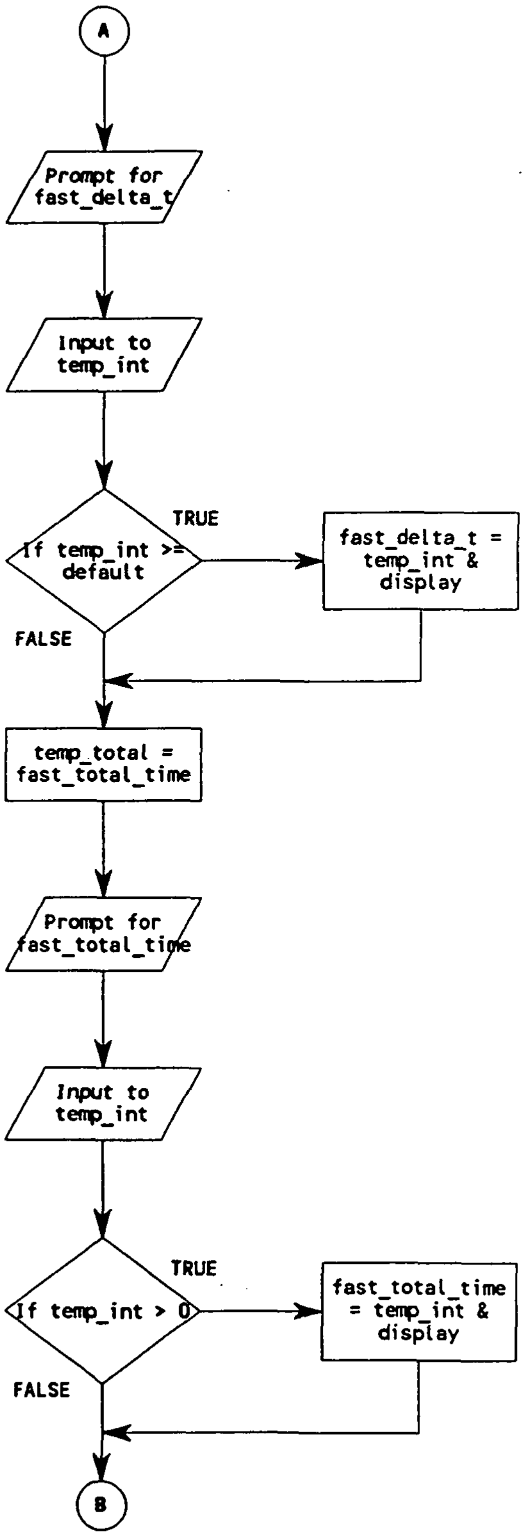
5. The type of thermocouple is requested. Only valid types are permitted with R as the default if no type is input by simply pressing 'ENTER'.

6. Finally up to 60 characters are requested for a comment. Typically, this would be used as identification for the run.

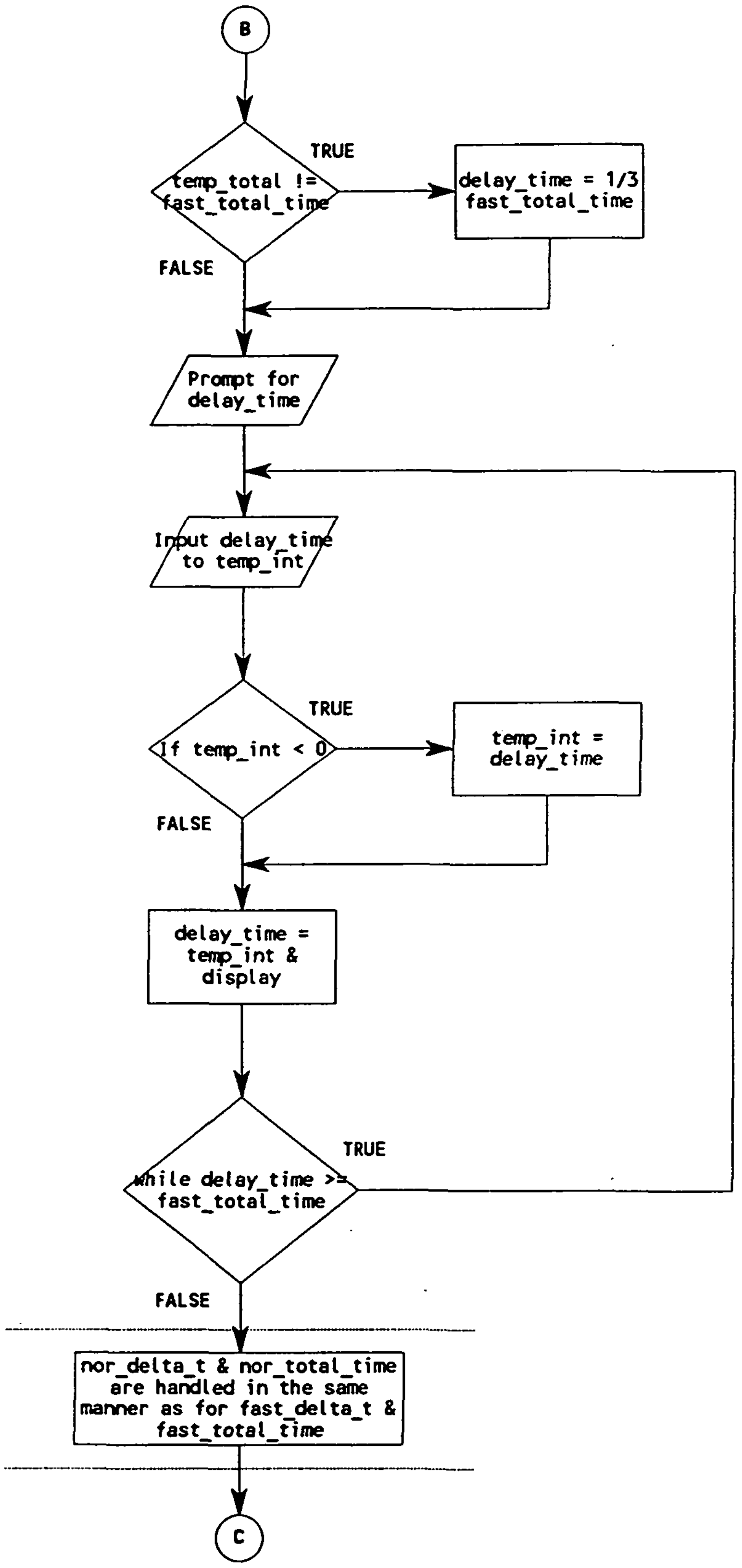
get_channel_data() - 1/5



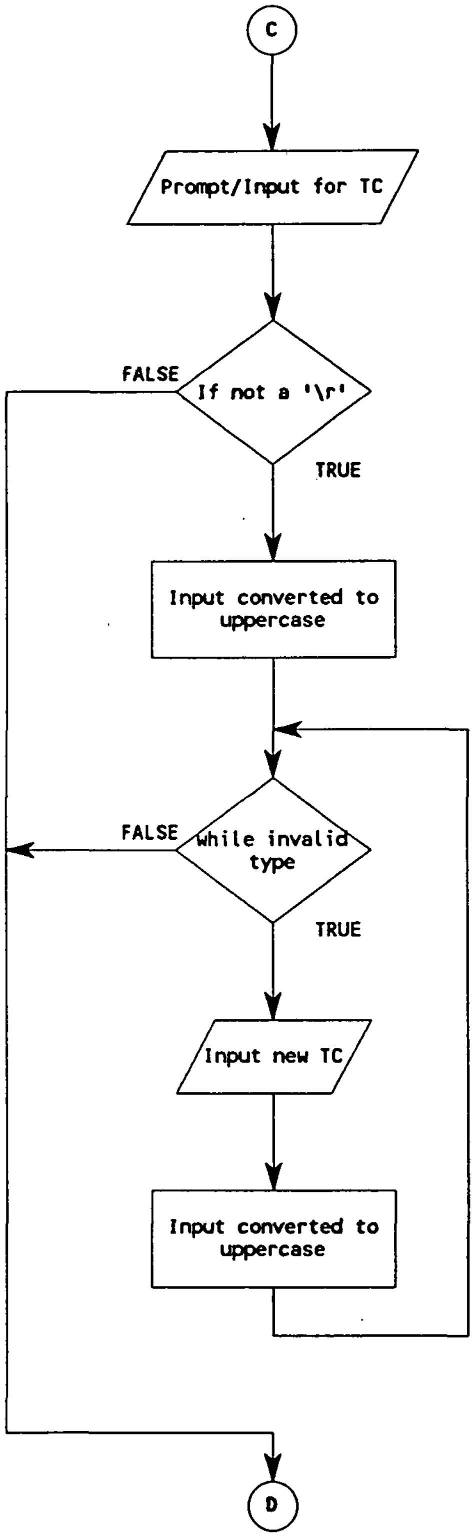
get_channel_data() - 2/5


A
Prompt for fast_delta_t
Input to temp_int
If temp_int >= default
TRUE
fast_delta_t = temp_int & display
FALSE
temp_total = fast_total_time
Prompt for fast_total_time
Input to temp_int
If temp_int > 0
TRUE
fast_total_time = temp_int & display
FALSE
B

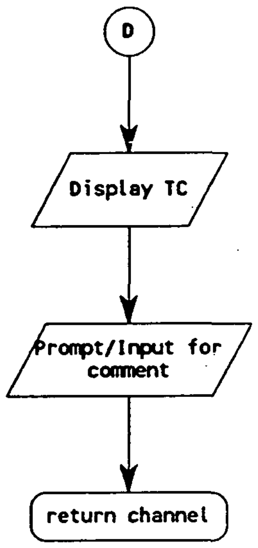
get_channel_data() - 3/5


B
temp_total != fast_total_time
TRUE
FALSE
delay_time = 1/3 fast_total_time
Prompt for delay_time
Input delay_time to temp_int
If temp_int < 0
TRUE
FALSE
temp_int = delay_time
delay_time = temp_int & display
while delay_time >= fast_total_time
TRUE
FALSE
nor_delta_t & nor_total_time are handled in the same manner as for fast_delta_t & fast_total_time
C

get_channel_data() - 4/5



get_channel_data() - 5/5



FUNCTION get_nor_data()

CALLED BY: do_nor_acq()

CALLS: get_temp_coeff()
make_solid_box()
reset_2497A

PROTOTYPE: void activate_relay(int which_channel)

SEE: do_nor_acq()
main()

This function performs the actual collection of data for the normal acquisition process. The determination of which ACTIVE channels will have their flag set to COLLECT is done in **do_nor_acq()**. The function displays the channels to be collected, creates the required strings to be sent to the 3497A, and reads the data from *cc* (weight) and *cc+20* (temperature). In the present implementation channel numbers, *cc*, are numbered from 0-11. The data are summed into the array **data_array[][]**. A single collection cycle lasts for 30 seconds. During this time the weight and temperature readings are stored in the above array. The average of these numbers is determined, the temperature determined and the raw weight and calculated temperature data are stored on the hard disk.

1. Various initializations are performed. The 3497A clock is synchronized with the CPU clock, the arrays to be used to sum the data are cleared and the normal acquisition screen showing the channels being collected from is displayed.

2. While **end_time** is greater then the current time the following functions are performed:

   A. All of the channels whose **flag** has been set to COLLECT have their weight and temperature data collected. The weight data comes from channel *cc* while the temperature data comes from channel *cc+20*.

   B. This resulting data is summed into **data_array[cc][0]** (weight) and **data_array[cc][1]** (temperature).

   C. The current time is redetermined and a counter, **iteration**, is incremented.

3. As the last pass through the while loop in step 2 results in the counter **iteration** being set to one greater than the number of times the channels are read, **iteration** is decremented.

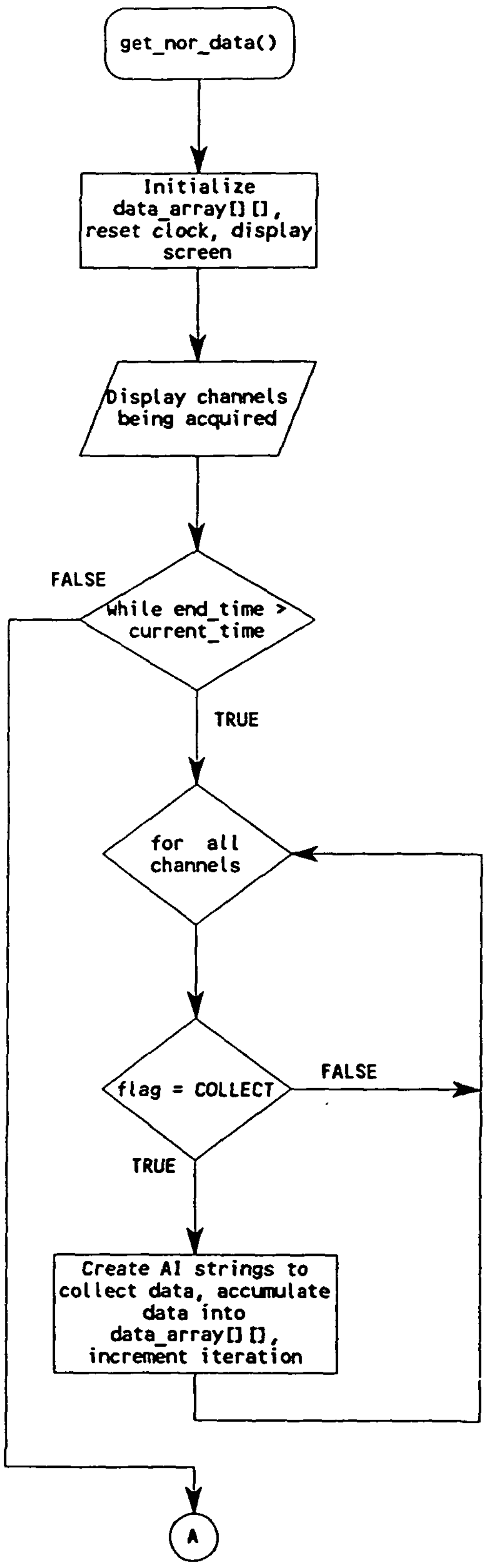
4. For all of the channels whose **flag** has been set to COLLECT the following steps are performed:

   A. The average of the datum are determined. The function **get_temp_coeff()** is called to determine the actual temperature.

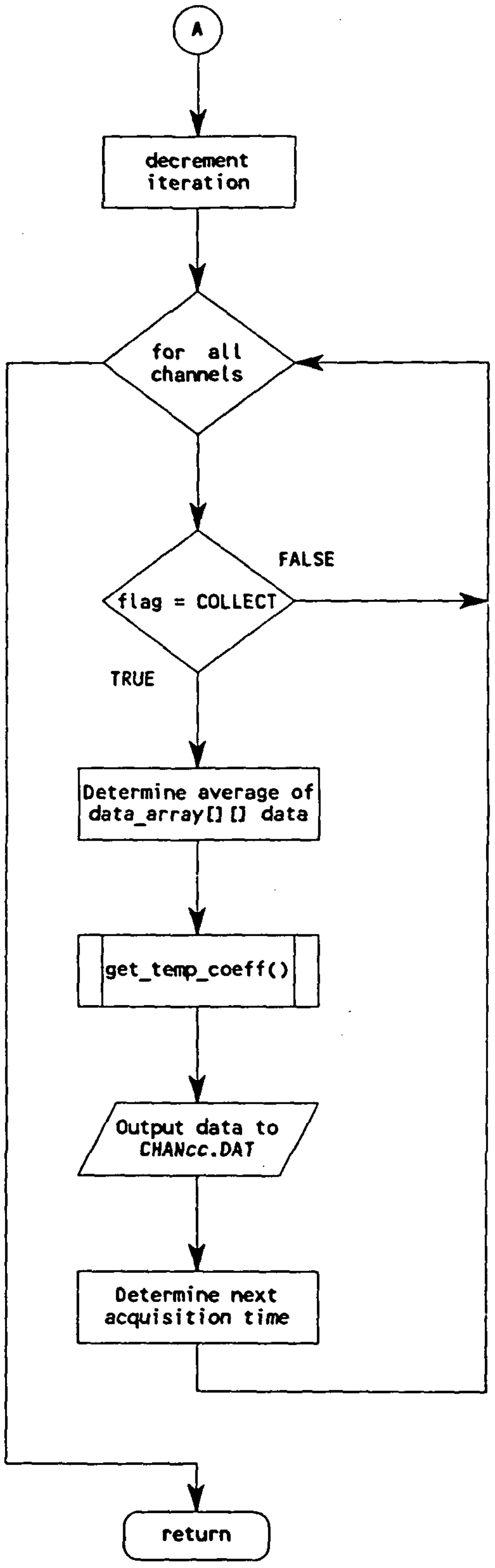
   B. The **flag** is reset to ACTIVE and the resulting relative time from the start of the run to the present time, the weight, temperature, and actual date:time are stored to the file whose name is in **filename**.

5. The function then returns.

get_nor_data() - 1/2


get_nor_data()
Initialize data_array[][], reset clock, display screen
Display channels being acquired
While end_time > current_time
FALSE
TRUE
for all channels
flag = COLLECT
FALSE
TRUE
Create AI strings to collect data, accumulate data into data_array[][], increment iteration
A

get_nor_data() - 2/2


A
decrement iteration
for all channels
flag = COLLECT
FALSE
TRUE
Determine average of data_array[][] data
get_temp_coeff()
Output data to *CHANcc.DAT*
Determine next acquisition time
return

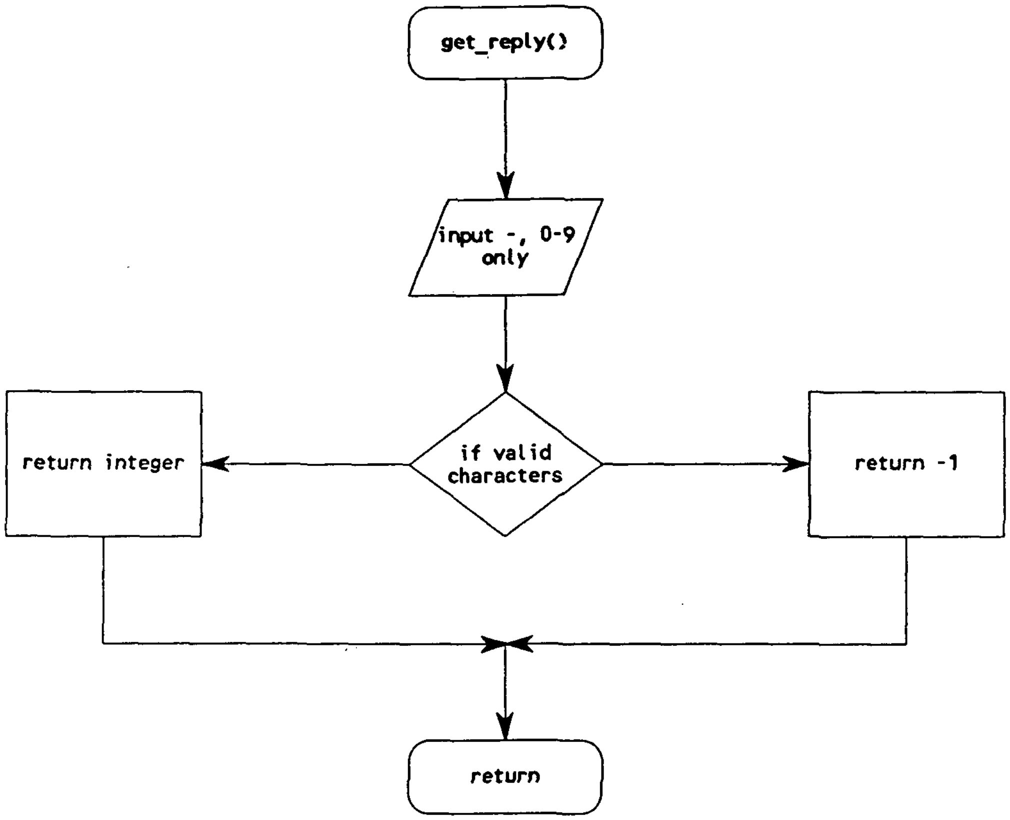
FUNCTION get_reply()

CALLED BY:	do_fast_acq()
	do_nor_acq()
	get_channel_data()
	main()

CALLS:	
PROTOTYPE:	int get_reply(void)
SEE:	

This function permits only numeric characters and the minus sign to be input via a **cscanf()**. If valid characters are input the function returns the resulting number otherwise a -1 is returned.

get_reply() - 1/1


get_reply()
input -, 0-9 only
if valid characters
return integer
return -1
return

FUNCTION get_startup_data()

CALLED BY:	start_channel()
CALLS:	int get_channel_data(struct channel *channel_data)
PROTOTYPE:	int get_startup_data(struct channel channel_data[12])

SEE:	get_channel_data()
	setup_data()

get_startup_data() calls **get_channel_data()** which prompts the user for the runs starting parameters. It also sets the channels **flag** to ACTIVE and writes the following starting parameters to CHAN*cc*.DAT:

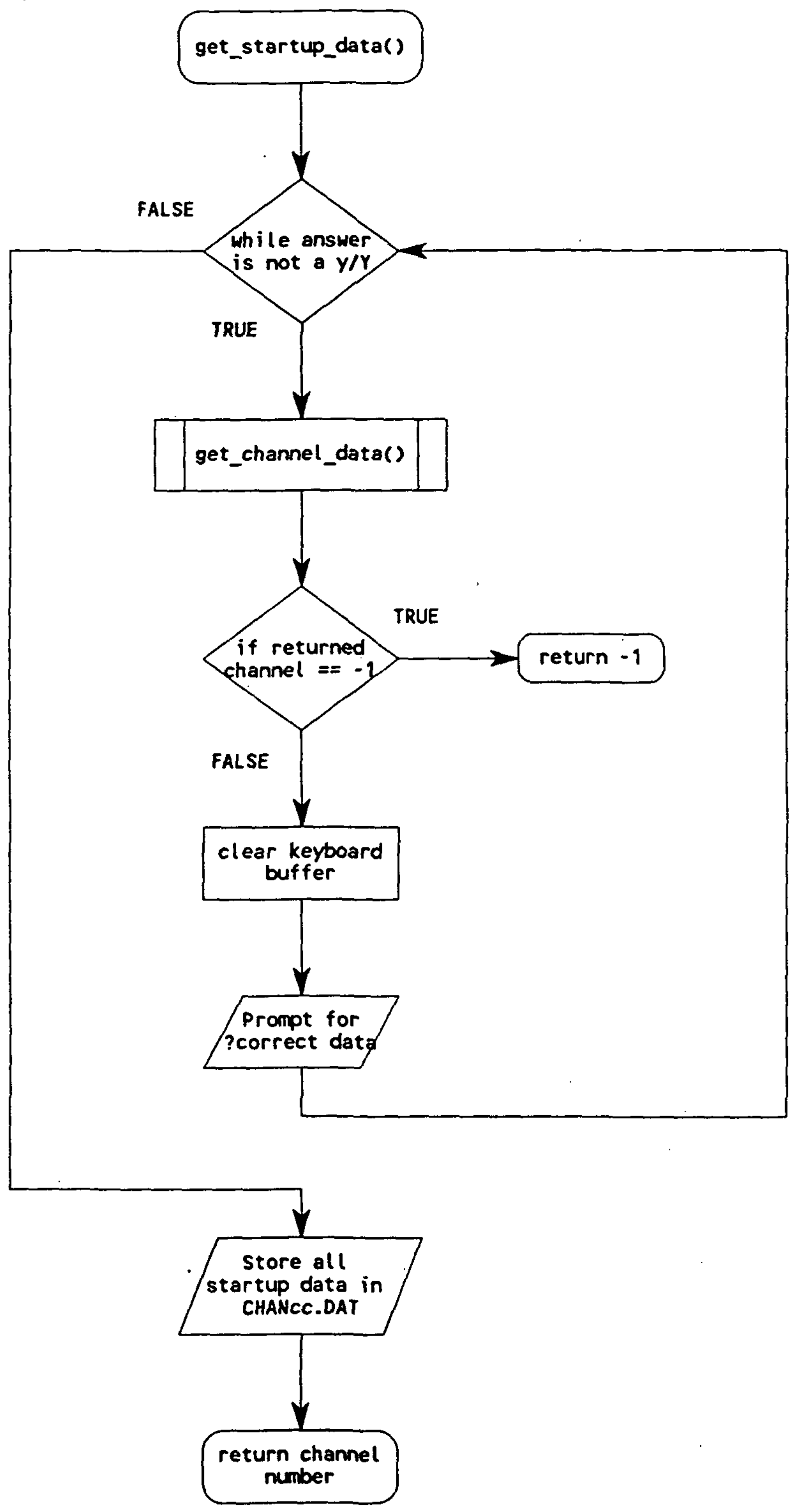
Filename
Channel data as numeric day, month, year, text day
Comment, up to 60 characters
Type of thermocouple
fast_delta_t
fast_total_time
fast_delay_time
nor_delta_t
nor_total_time
start_time
end_time

These 11 items are the FIRST records written to the data file. They may be used by **setup_data()** or for reference when analyzing the data.

The functions performed are as follows:

1. As long as the data is incorrect:
    - A. **get_channel_data()** is called. If it returns a -1, indicating the user did not want to proceed **get_startup_data()** returns to **start_channel()** with a -1.
    - B. Otherwise, the keyboard input buffer is cleared and the user is prompted to accept the data as correct.
2. If the data was correct from step 1 it is then written to the file specified in **filename**.
3. The function returns the channel number.

get_startup_data() 1/1



FUNCTION **get_temp_coeff()**

CALLED BY: do_fast_acq()
get_nor_data()

CALLS:

PROTOTYPE: float get_temp_coeff(float TC_volts, char TC_type)

SEE: get_temp_coeff()

The actual temperature is calculated by this function for a specified type of thermocouple. The permitted types are E, J, K, R, S, T. Other types may be easily added, probably even Don could do this. The function loads an array, **TC_data[][]**, with coefficients for a 9th order polynomial. This polynomial is used to determine the temperature from the voltage readings of the 3497A, channels *cc+20*. The data for all but one of the thermocouples is taken form the OMEGA "TEMPERATURE HANDBOOK", copyright 1988, see the program listing. The one exception is the type R thermocouple for the temperature range 1000-1760 degrees Celsius. These coefficients were determined by specifying the millivolt/temperature pairs over the range 900-1760 degrees Celsius in 10 degree increments starting with 900 degrees. Again the data were taken from the afore mentioned reference. These ordered pairs were input to "SIGMAPLOT" and the coefficients for a third order polynomial were determined. The determined regression value for these data was 1.00000.

Input data to these equations MUST be in VOLTS. Also note that all calculations and variables are specified as type double due to the required precision of the coefficients.

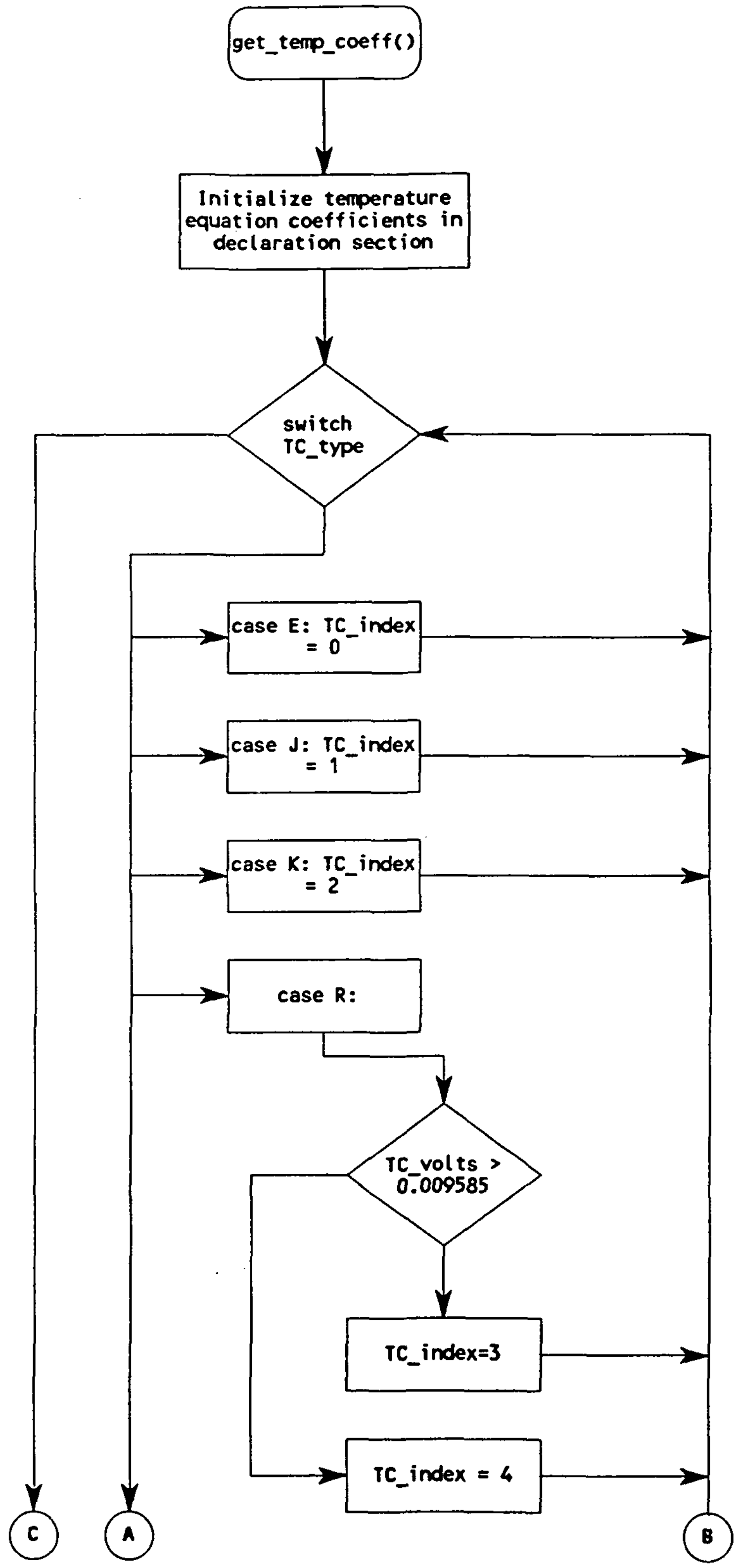
The function does the following:

1. An index into the array **TC_array[][]** is determined based on the type of thermocouple and the temperature range in the case of the R thermocouple.

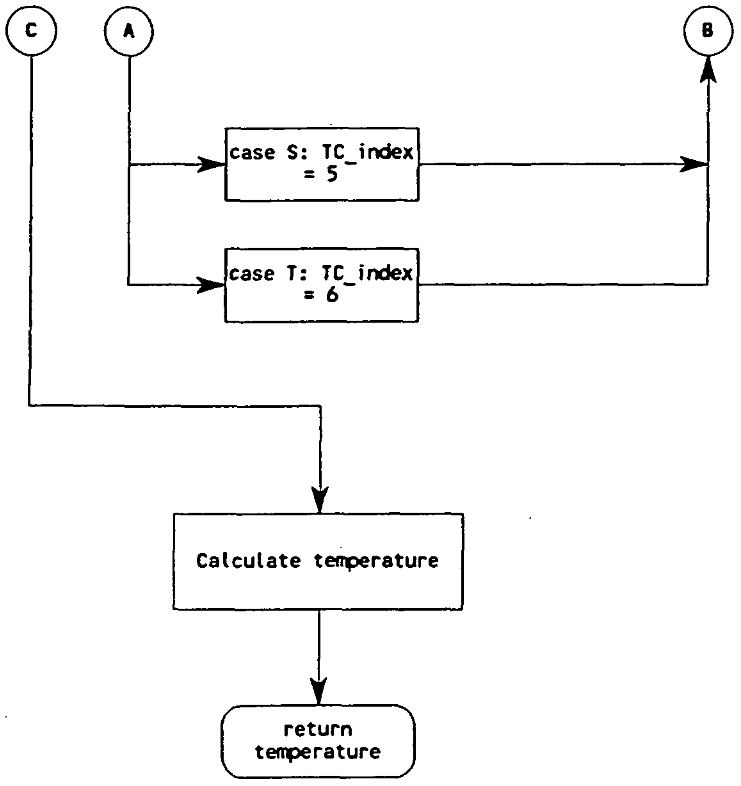
2. A 9th order polynomial uses the determined coefficients to calculate the temperature.

3. The temperature is returned as a type cast float variable.

get_temp_coeff() - 1/2



get_temp_coeff() - 2/2



FUNCTION main()

CALLED BY:

CALLS:
```
void disk_opns(void)
void do_nor_acq(void)
void end_channel(void)
void     make_menu_display(void)
void make_solid_box(int lower_left_x, int lower_left_y,
         int upper_right_x, int upper_right_y,
         int FILL_COLOR, int BORDER_COLOR)
int  reset_3497A(void)
void setup_data(void)
void start_channel(struct channel *channel_data);
```

PROTOTYPE: void main(void)

SEE:

General considerations are:

1. The program can handle 12 apparatuses or channels. The channels are numbered from 0 to 11.

2. The program synchronizes the 3497A clock to the CPU clock. The primary reason for this is that the 3497A clock will lose its time after a power outage of about 24 hours. Therefore, with the automatic startup feature incorporated in this system, the 3497A clock may be reset after any length of power shutdown. This enables the program to pick up right where it left off.

3. Time for the acquisition process is measured in seconds since 1 January 1970. The reason for this is that this greatly simplifies the determination of start, collection, and ending times as all times are in seconds. A consequence of this is that one does not have to be concerned with time changes at midnight, end of month, and end of year.

4. All acquisitions are started and stopped by a fast acquisition. The characteristics of each being the same except as noted for **fast_delay_time** below.

The **main()** function is the data acquisition controlling function which performs the following actions:

1. Initializes the 3497A by insuring its clock is in sync with the CPU operating the program. In addition, immediately before any data is acquired another clock sync is performed. The CEC IEEE-488 board is initialized, see the CEC manual.

2. The array of data structures, **channel_data[12]**, is set to default values. The meaning of the individual fields are:
flag - May have three values, INACTIVE, ACTIVE, and COLLECT. The first value indicates that no acquisition is to be performed on the specified channel. If the channel is ACTIVE or COLLECT then data *will* be collected for any channel whose collection time is due or past due.
fast_delta_t - Time between fast acquisition samples in seconds. Minimum time is 10 seconds, default is 30 seconds. The actual length of time data is collected is (**fast_delta_t - 2**).

fast_total_time - Total length of time allowed for fast acquisition, in minutes. Minimum time is 1 minute, default is 15 minutes.
fast_delay_time - The length of time, in minutes, from the start of the fast acquisition process before a digital output signal is set. This signal may be used to raise the furnace. Likewise this delay time is used to lower the furnace. In which case it is the time in minutes before the run termination when a digital output signal is reset. This may be used to lower the furnace. The default delay time is $^1/_3$ of the **fast_total_time.**
nor_delta_t - Time between normal acquisition samples in minutes. Minimum time is 1 minute, default is 6 minutes. The length of time data is collected is 30 seconds.
nor_total_time - Total length of time allowed for fast acquisition, in hours. Minimum time is 1 hour, default is 1 hours.
next_time - The time for collection of the next normal acquisition sample. This time is specified in seconds since 1 January 1970.
end_time - The time for termination of the acquisition for the channel. This time is specified in seconds since 1 January 1970.
start_time - The time the acquisition for the channel started. This time is specified in seconds since 1 January 1970.
filename - Filename for the collected data in the form **CHANx.DAT**. Where *x* is the channel number.
comment - Up to 60 characters of any comments may be entered. Default is a NULL string.
chan_date - The date and day, spelled out, the channel acquisition was started.
TC - The type of thermocouple used in the acquisition process. Type R is default. The only permitted types are E, J, K, R, S, and T.

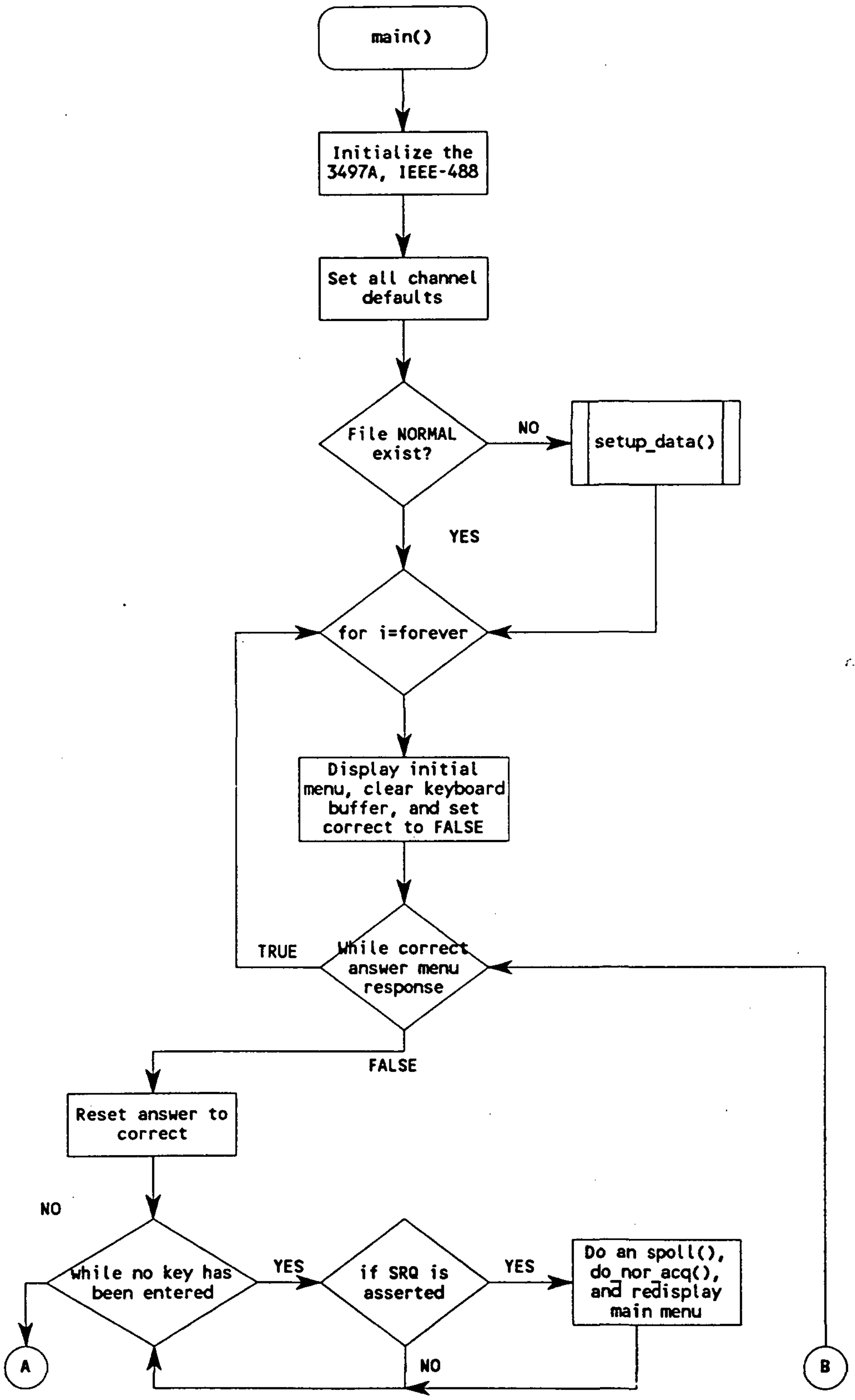
2. The file NORMAL is checked for. If it is present it signifies that the program was terminated in an orderly fashion, option E on the main menu. Further, NORMAL is deleted. If the NORMAL is not present at startup this indicates an abnormal shutdown and the function **setup_data()** is called to reset all channel parameters to their original values prior to shutdown. All of the requisite data is stored in the files **CHANx.DAT** by **get_startup_data()** upon channel startup.

3. An assembler routine is used to insure the keyboard buffer is clear.

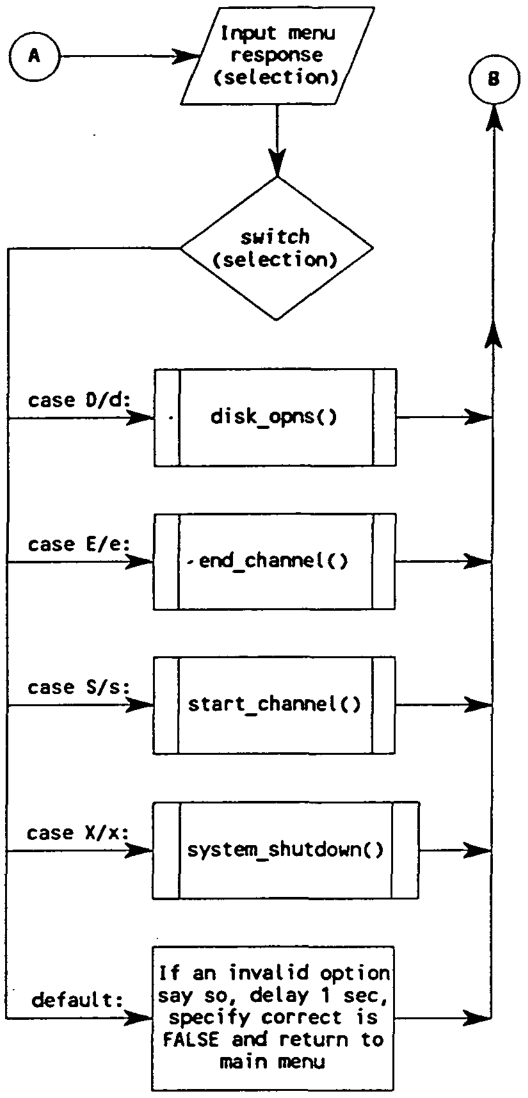
4. The program simply waits in a loop for a key to be struck or for SRQ to be asserted. If SRQ is asserted, it is cleared by an **spoll** then a normal acquisition is performed. If a key is pressed it is serviced depending on the key pressed. In any event, the function associated with the key is performed before another normal acquisition.

5. Correct responses to the master menu are either D/d, E/e, S/s, or X/x. The first case results in a disk operation by **disk_opns()**. The second case is used to start the acquisition process for a channel. The third case is used to terminate a channel. The last case provides an orderly shutdown of the system. This is critical with regard to 2 above. If none of the above keys are struck then the default option is performed indicating an error and requesting reinput of data.

main() - 1/2


main()
Initialize the 3497A, IEEE-488
Set all channel defaults
File NORMAL exist?
NO
setup_data()
YES
for i=forever
Display initial menu, clear keyboard buffer, and set correct to FALSE
TRUE
While correct answer menu response
FALSE
Reset answer to correct
NO
While no key has been entered
YES
if SRQ is asserted
YES
Do an spoll(), do_nor_acq(), and redisplay main menu
NO
A
B

main() - 2/2


A
Input menu response (selection)
B
switch (selection)
case D/d:
disk_opns()
case E/e:
end_channel()
case S/s:
start_channel()
case X/x:
system_shutdown()
default:
If an invalid option say so, delay 1 sec, specify correct is FALSE and return to main menu

FUNCTION make_menu_display()

CALLED BY: main()

CALLS: make_solid_box()

PROTOTYPE: void make_menu_display(void)

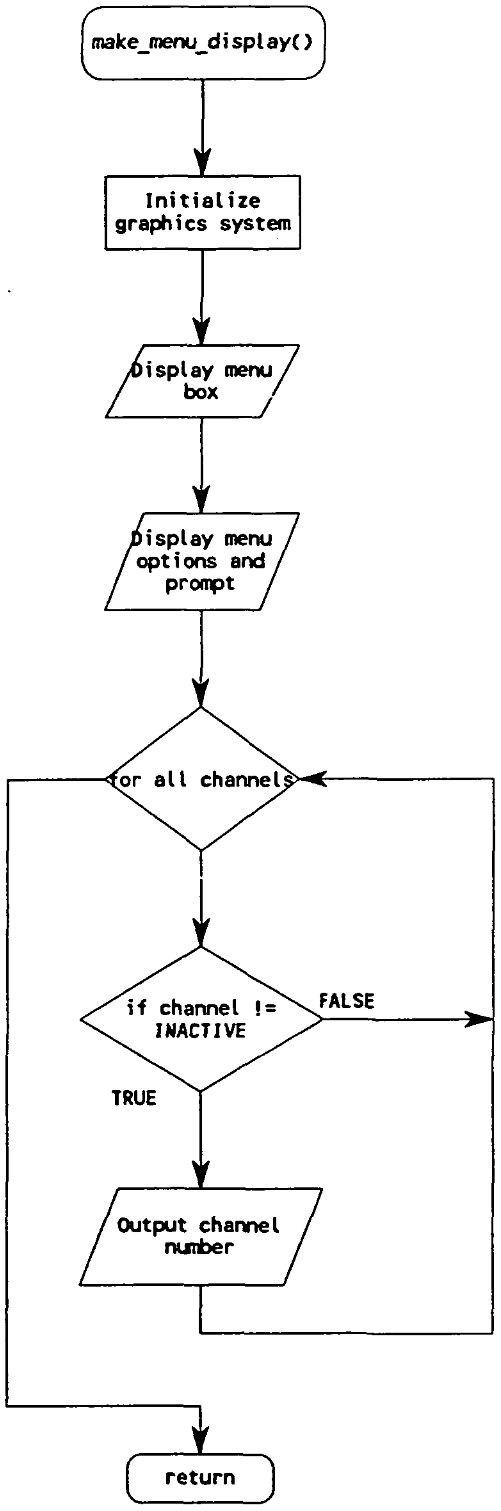
SEE:

The main menu display is prepared by this function. Don's favorite text font, GOTHIC, is used to title the screen. The menu options are presented and finally the active channels are displayed. Additional options may be added by expanding the box created in step 2 below.

The function:

1. Initializes the graphics system.
2. Outputs the solid box for the options, displays the screen title, then enters the options in the box.
3. The user prompt for an option selection is displayed.
4. Finally, all channel's flags are scanned to determine which channels have been activated, flag != 0, and these channel numbers are displayed.

make_menu_display() - 1/1



FUNCTION make_solid_box()

CALLED BY: do_fast_acq()
do_nor_acq()
get_nor_data()
main()
make_menu_display()

CALLS:

PROTOTYPE:
```
void make_solid_box(int lower_left_x, int lower_left_y,
             int upper_right_x, int upper_right_y,
             int FILL_COLOR, int BORDER_COLOR)
```

SEE:

Uses the ANSI C function **fillpoly()** to create a polygon, rectangle filled with FILL_COLOR and having a border of color BORDER_COLOR. The polygon is initialized form the parameter list.

No flowchart displayed.

FUNCTION reset_3497A()

CALLED BY: do_fast_acq()
do_nor_acq()
get_nor_data()
main()

CALLS:

PROTOTYPE: int reset_3497A(void)

SEE:

Reads the current time from the CPU using the ANSI C function **localtime()**. The ANSI function **strftime()** is used to create a time string which is sent to the 3497A with a TD command to reset the 3497A clock. The primary reason for using the CPU clock is that it will, typically, be more stable in terms of battery life than the 3497A batteries. Past experience has shown that if the 3497A is powered off for over 24 hours that it will lose its time. This requires the operator to manually reset the clock.

No flowchart displayed.

FUNCTION setup_data()

CALLED BY:	main()
CALLS:	void activate_relay(int which_channel) void do_nor_acq(void)
PROTOTYPE:	void setup_data(void)

SEE:	do_nor_acq()
	get_startup_data()
	startup() - *separate program*

In the event of an abnormal termination the startup data written to the file(s) CHAN*cc*.DAT is read back into **channel_data[]**. A normal acquisition is started and the acquisition continues. Channels are restarted with a fast acquisition if the current time is less than Regardless of the duration of the shutdown, upon a restart via this function all channels are restarted with a normal acquisition via **do_nor_acq()**. This means that if a channel had just completed a sampling and the power failed for a brief period of time then the next normal acquisition time would be relatively close to the previous acquisition time.

1. The function checks for the existence of all channels files of the type CHAN*cc*.DAT. For each of these files the initialization data, see **get_startup_data()** is read into **channel_data[]**.

2. The furnace is reactivated. It should still be in the up position unless the failure occurred before the **delay_time** on startup or after the **delay_time** on termination. In these latter two cases the furnace is raised. and in the last case it will be lowered during the termination of the channel when **do_nor_acq()** is called.

3. **do_nor_acq()** is called to restart the channels.

No flowchart displayed.

FUNCTION **start_channel()**

CALLED BY: main()

CALLS: int get_startup_data(struct channel channel_data[12])
void do_fast_acq(struct channel channel_data[12], int channel)
void do_nor_acq(void)

PROTOTYPE: void start_channel(struct channel *channel_data)

SEE:

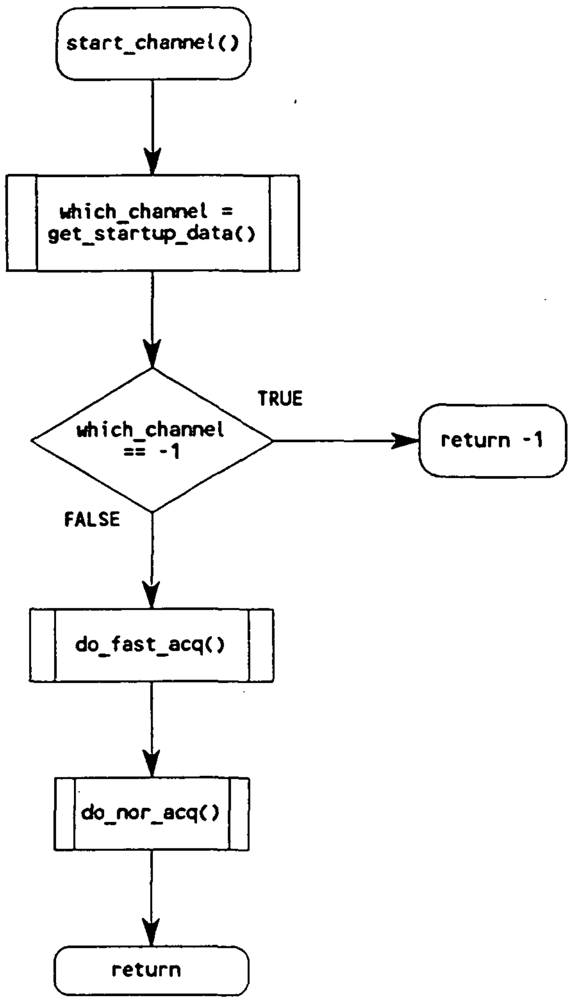
1. It first calls **get_startup_data()** to get the channel to be started.

2. If the returned channel is -1 the function returns to **main()** which redisplays the main menu.

3. If a valid channel is returned first **do_fast_acq()** is called then **do_nor_acq()** is called.

4. After step three the function also returns to **main()**.

start_channel() - 1/1



FUNCTION system_shutdown()

CALLED BY: main()

CALLS:

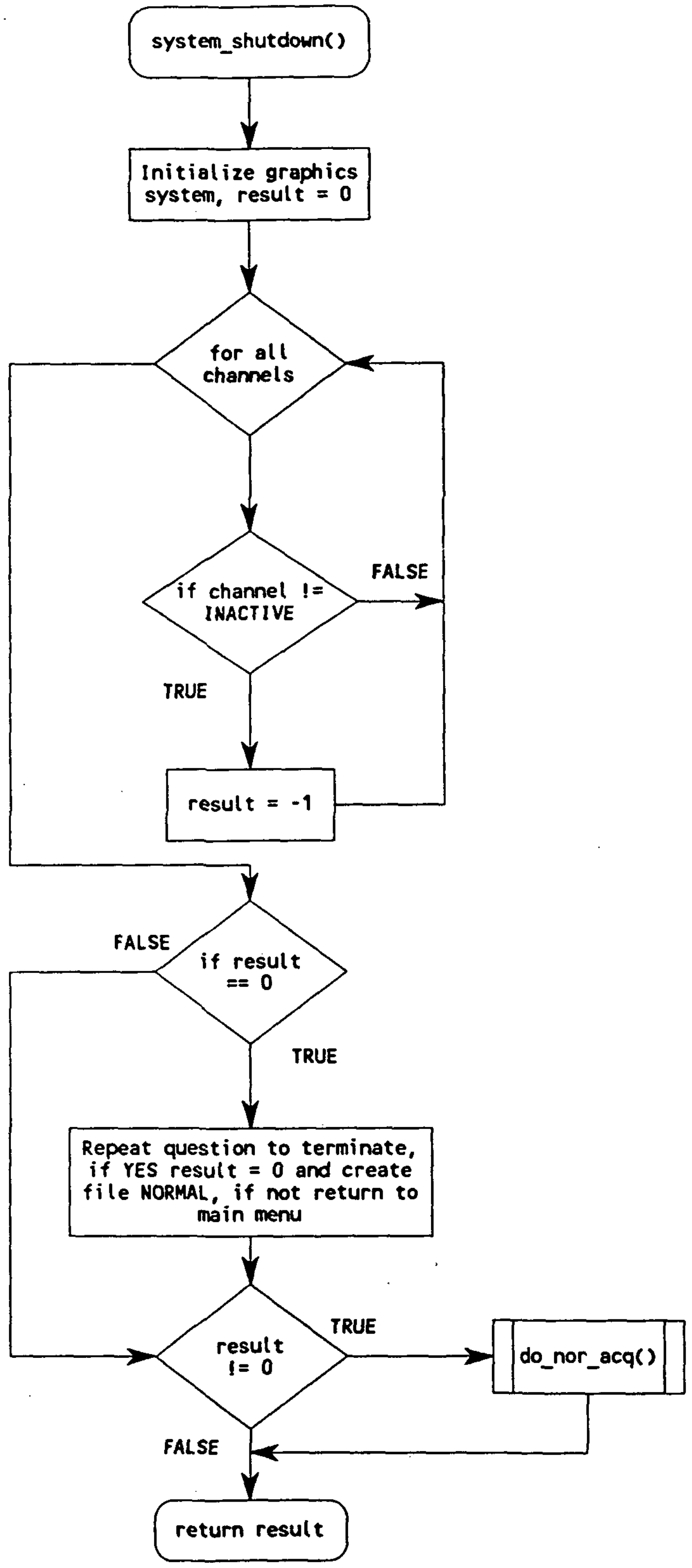
PROTOTYPE: int system_shutdown(void)

SEE: main()

system_shutdown() is used in response to option X/x of the main menu. It provides an orderly method of terminating the acquisition program. If this option is not used the file NORMAL will not be created leading to possible problems the next time the system is started. It also insures that the system cannot be stopped if ANY channels are active. All channels must be terminated via option E/e of the main menu before this function will terminate the system.

1. The graphics system is first initialized and the variable **result** is set to 0, indicating a possible shutdown.
2. All channel's **flag** is checked to determine if any channels are active. If any channel is active **result** is set to 1.
3. If result, from 2 above, is still 0 and additional prompt is output to insure that the operator wants to terminate the system. If so the function returns 0.
4. If result was set to 1 the function **do_nor_acq()** is called then the system returns to the main menu with a 1.

system_shutdown() - 1/1



FUNCTION terminate channel()

CALLED BY:	end_channel()
	do_nor_acq()

CALLS:	
PROTOTYPE:	void terminate_channel(int channel)
SEE:	do_fast_acq()

terminate_channel() is used to stop data acquisition either manually, via **end_channel()** or when its time is expired, via **do_nor_acq()**.

channel_data[].flag is first set to INACTIVE and a fast acquisition is performed. A fast acquisition terminates all channels in the same manner as start up. After this final data is stored in the file as specified in **channel_data[].filename**. This file is renamed according to the following:

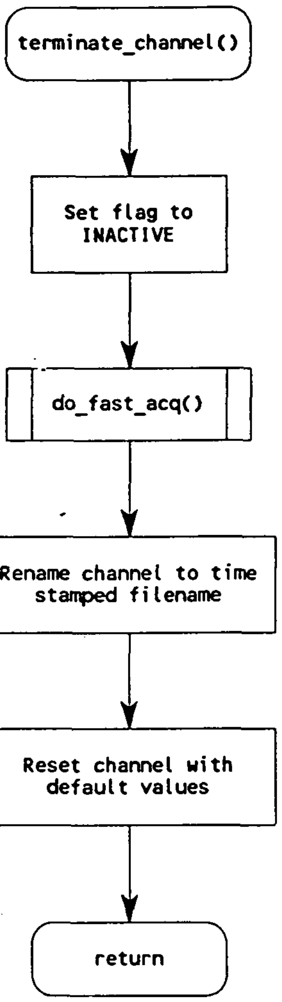
CHcchhmm.DAT	cc - The channel number, a one or two digit number
	hh - Hour of termination, 24 hour clock.
	mm - Minute of termination

This scheme insures unique channel names UNLESS, a channel is stopped at precisely the same time as a previous acquisition was stopped. It was assumed this was a remote possibility. Also as the channels are going to be dumped to a diskette on a periodic basis any previous channel with the same name will be retained on another diskette. It is incumbent for the apparatus operator be aware of this potential *disaster* and store data on a periodic basis on floppy diskettes. In addition, on a periodic basis all data files, files ending in a DAT, should be purged from the hard disk. This would be done by returning to the DOS system and manually deleting these files.

The final action of this function is to reestablish the channel default values so the channel may be used for another acquisition.

This signal is used to raise the furnace for the apparatus in accordance with **fast_delay_time** as discussed in **main()**.

terminate_channel() - 1/1


terminate_channel()
Set flag to INACTIVE
do_fast_acq()
Rename channel to time stamped filename
Reset channel with default values
return

PROGRAM STARTUP.C

FUNCTION startup()

CALLED BY:

CALLS: void main(void) - *via execv()*

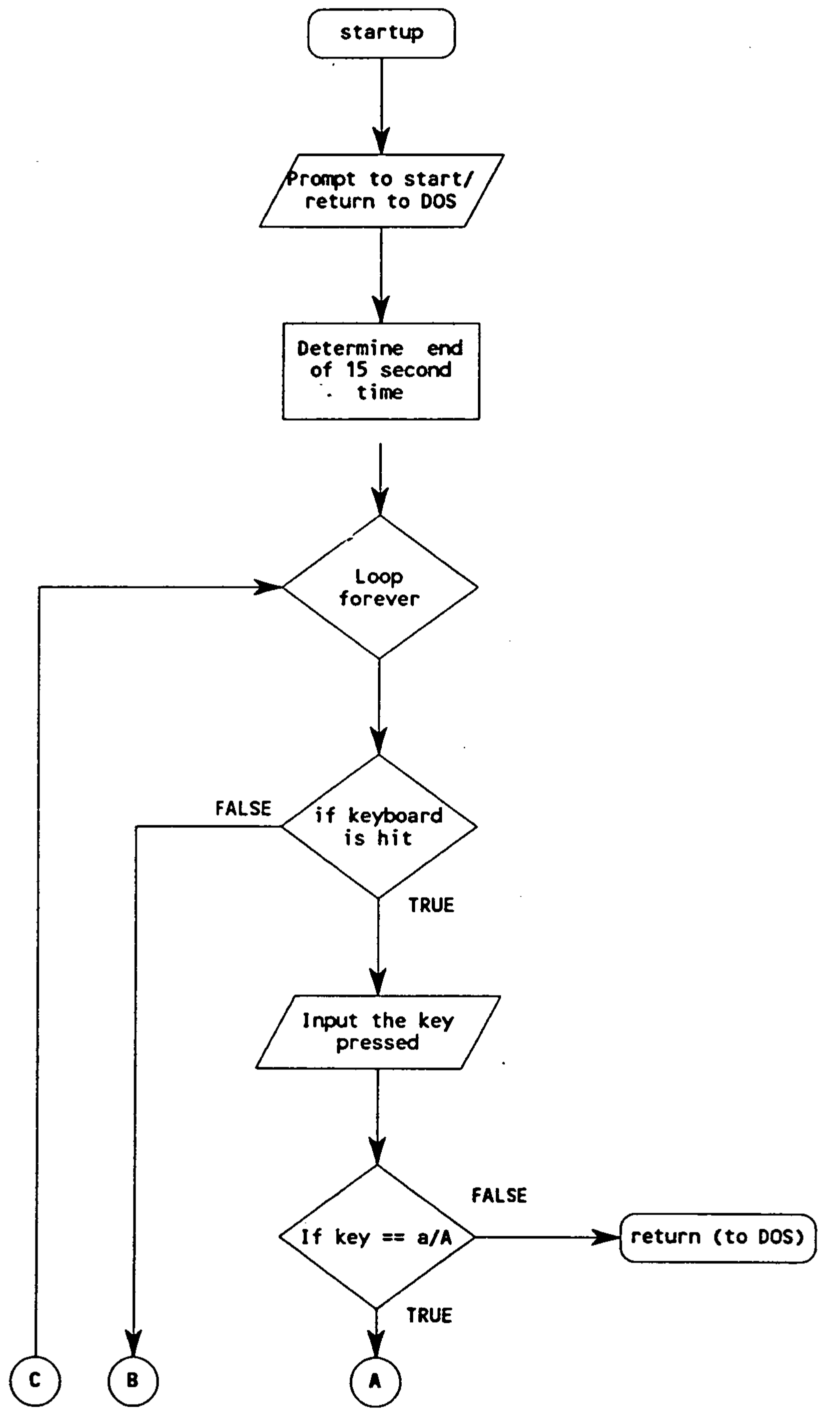
PROTOTYPE: void startup(void)

SEE: **SYSTEM OPERATION** section

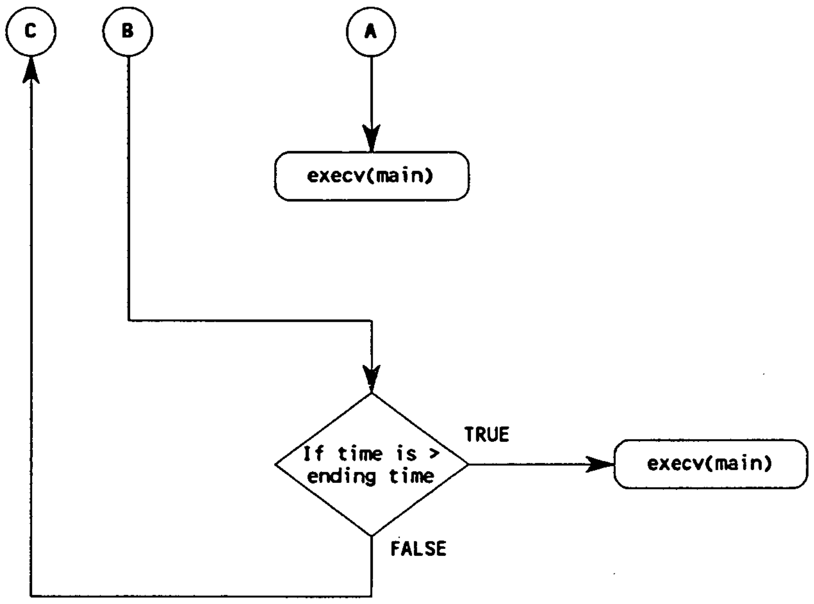
startup() is used to restart execution of **main()** in the event of an abnormal termination. Its execution is started by placing the file name in the AUTOEXEC.BAT file.

1. It displays a prompt to input an a/A.
2. If no response occurs within 15 seconds **main.exe** is started via an **execv()**.
3. If a/A is entered within 15 seconds, again **main.exe** is started via an **execv()**.
4. If any other key is entered the system returns the user to the DOS prompt in the working directory.

startup() - 1/2


startup
Prompt to start/ return to DOS
Determine end of 15 second time
Loop forever
FALSE
if keyboard is hit
TRUE
Input the key pressed
FALSE
If key == a/A
return (to DOS)
TRUE
C
B
A

startup() - 2/2



REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1991	Final Contractor Report

4. TITLE AND SUBTITLE Isothermal Thermogravimetric Data Acquisition Analysis System	5. FUNDING NUMBERS
6. AUTHOR(S) Kenneth Cooper, Jr.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Transylvania University Brown 315 Lexington, Kentucky 40508	8. PERFORMING ORGANIZATION REPORT NUMBER E-6474
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES) National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR-187171

11. SUPPLEMENTARY NOTES
Project Manager, James L. Smialek, Materials Division, NASA Lewis Research Center, (216) 433-5500.

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Unclassified - Unlimited Subject Category 61	

13. ABSTRACT *(Maximum 200 words)*

The description of an Isothermal Thermogravimetric Analysis (TGA) Data Acquisition System is presented. The system consists of software and hardware to perform a wide variety of TGA experiments. The software is written in ANSI C using Borland's Turbo C^{++}. The hardware consists of a 486/25 MHz machine with a Capital Equipment Corporation IEEE488 interface card. The interface is to a Hewlett Packard 3497A data acquisition system using two analog input cards and a digital actuator card. The system provides for 16 TGA rigs with weight and temperature measurements from each rig. Data collection is conducted in three phases. Acquisition is done at a rapid rate during initial startup, at a slower rate during extended data collection periods, and finally at a fast rate during shutdown. Parameters controlling the rate and duration of each phase are user programmable. Furnace control (raising and lowering) is also programmable. Provision is made for automatic restart in the event of power failure or other abnormal terminations. Initial trial runs were conducted to demonstrate system stability. Extensive parameter variation between runs, many simultaneous runs, simulation of power outages have demonstrated system stability and reliability under a variety of operating conditions. This system has improved on the prior one in these main areas:
A. Recover from abnormal termination conditions with no loss of data.
B. Preprogramming all phases, allowing unattended startup and shutdown of a run.
C. Ease of operation - utilizing disk based systems as opposed to a tape based system.
D. Ready availability of data during a run.

14. SUBJECT TERMS Thermogravimetry; Metal oxides; Oxidation; Data acquisition	15. NUMBER OF PAGES 64
	16. PRICE CODE A04

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	