A Computer-Based Respiratory Measurement System and a Temperature Transducer for Monitoring Respiratory Flow Temperature

by

MICHAEL HARRY MASTERS

B.S., Kansas State University, 1982

A MASTER'S THESIS

submitted in partial fulfillment of the requirements
for the degree
MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by

Richard D. Gallagher

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I. INTRODUCTION

A primary interest of the respiratory physiologist is the ability to obtain a quantitative measnre of ventilation, the exchange of oxygen and carbon dioxide between the alveolar spaces in the lung and the venons blood. Snch measurements may be approximated by measnring the gas exchange between the snbject and the atmosphere. Classical ventilation measnrements (e.g. bag techniqnes) rely on the collection of expired gases for several respiratory cycles [14]. Snch measnrements yield average ventilation valnes so that only steady-state responses may be considered. By making the ventilatory measnrements on a per breath basis the ventilatory response to transient or short dnration stimnli is possible while, by totaling the per breath valnes, stndy of the response to a steady-state stimnlns is maintained. These breath-by-breath measnrements may provide both the clinician and the researcher new insights into the health or conditioning of a snbject [6,14].

Recordings at the month of the bi-directional (inspiratory and expiratory) respiratory gas flow and flow temperatnre signals and the concentration signals of the component gases yield a complete history of the gas exchanged between the atmosphere and the snbject. Integration of the flow, which is converted to constant and standard conditions nsing the flow temperatnre and ambient conditions, over an inspiratory cycle and over an expiratory cycle yields the respective inspiratory and expiratory tidal volumes. Also, the flow signal can be mnltiplied by the individnal gas concentrations to obtain flows of the individnal gases. These gas flows can be integrated to obtain their respective tidal volumes. The difference between the inspiratory volume and the expiratory volume of a gas yields the volume of gas exchanged for the respiratory cycle. These per breath tidal volumes and exchanged volumes can be converted to minnte valnes by dividing by the respiratory cycle time. Also, several breath-by-breath valnes may be totaled and averaged to obtain an average response over a specified period of time.

With cnrrent technology, transdncers are available to monitor the necessary signals to make the respiratory measnrements. Pneumotachometer and low inertia tnrbine flow transdncers provide the ability to monitor bi-directional respiratory flow. Rapidly responding gas analyzers have the ability to monitor the rapidly varying respiratory gas concentration signals. A rapidly responding temperatnre transdncer can monitor the flow temperatnre signal. These signals may be digitally recorded with a compnter-based data

acqnisition system. The compnter provides the ability to numerically integrate the gas flows. The compnter also provides the flexibility to provide a variety of techniqnes for data display.

A Compnter-Based Respiratory Measnrement System (CBRMS) has been developed by Creel [4] and Riblett [13] to monitor fonr respiratory signals; respiratory flow, fractional concentration of 0_2 and CO_2 in the dry gas, and respiratory flow temperatnre. A Godart PneumoTachoGraph (PTG), nsing a Fleisch #2 PneumoTachoMeter (PTM), is nsed to transdnce the respiratory flow signal. A Perkin-Elmer 1100 Medical Gas Analyzer (the acronym GMS, Gas Mass Spectrometer, is nsed to refer to the gas analyzer) is nsed to transdnce the dry gas concentration signals of 0_2, F_{0_2}, and CO_2, F_{CO_2}. The design of a rapidly responding temperatnre transdncer based on a 0.001 inch chromel-constantan thermoconple is presented in this stndy. This transdncer is naed to monitor the respiratory flow temperature signal.

Using the fore-mentioned signals the CBRMS can make a variety of ventilation measnrementa. These breath-by-breath ventilation measnrements are a measnre of gas exchange between the lnngs and the atmosphere on a per breath basis and approximate, assuming no change in pnlmonary gas stores, alveolar ventilation; the exchange of respiratory gases between the alveolar spaces and the venons blood. Other stndies have extended aimilar systems, nsing the same measnred signals, to calcnlate the trne alveolar gas exchange on a breath-by-breath basis [2] yielding additional insight into transient changes in ventilation. An effort to provide snch an extension to the CBRMS of this stndy is cnrrently under way with promising preliminary resnlts [12].

Chapter II. GENERAL SYSTEM DESCRIPTION

To appreciate the types of respiratory measnrements made with the Compnter-Based Respiratory Measnrement System (CBRMS) it is necessary to have a general nnderstanding of the respiratory signals and how the system monitors, records, and manipnlates these signals. The measurements necessary for assessing respiratory ventilation are the varions gas volumes (air, oxygen, carbon dioxide), both on a tidal and time rate basis and respiratory freqnency. Obtaining these measnrements for each breath allows investigation of the respiratory system's transient and, by averaging these breath-by-breath valnes, steady-state responses.

An overview of the CBRMS is presented in this chapter while the details of the system are explained later. This overview provides the respiratory physiologist the necessary understanding to make confident respiratory fnnction measnrements. Prior to the overview a brief discnssion of the respiratory signals correlated with the varions stages of the respiratory cycle is presented. Reference to several introdnctory physiology texts provides the necessary description of the respiratory system and defines the necessary terminology.

2.1 The Respiratory Signals

Investigation of respiratory signals appearing at the month is essential in understanding how a system may be developed which can take these signals and compnte valnes for the respiratory variables. Consider the respiratory model shown in Fignre 2.1.1. This model is a simple representation of the pnlmonary system.

The model contains the necessary detail for nnderstanding the compntation of the breath-by-breath gas volumes based on measnrements made at the month. The two volumes inclnded in the model are the constant dead space volume, inclnding that added by the PTM, and the lung volume which varies thronghont the respiratory cycle. The lung volume inclndes the varions pnlmonary volumes (tidal volume, inspiratory reserve volume, expiratory reserve volume, and residnal volume) and the pnlmonary capacities (functional residnal capacity, inspiratory capacity, vital capacity, total lung capacity) [6]. The alveolar oxygen consumption and carbon dioxide prodnction are also inclnded in the model. These valnes may be derived from the month valnes

usiug this model [2,12].

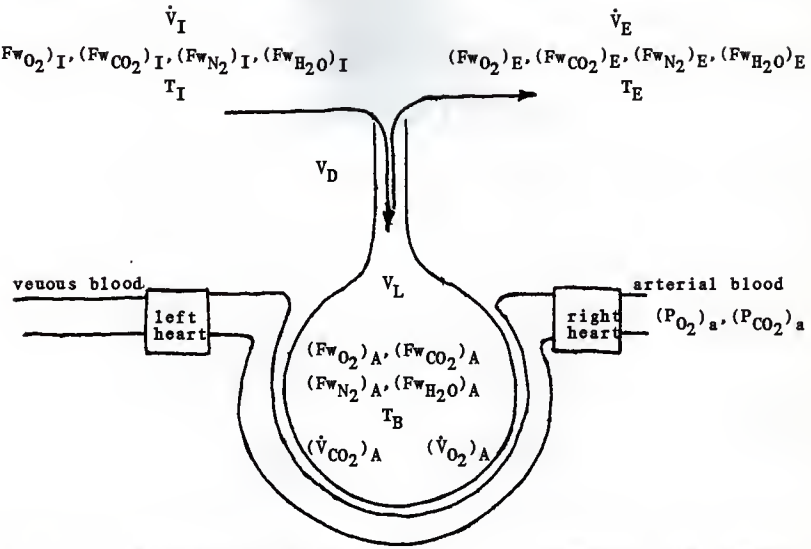


Figure 2.1.1 A respiratory model for understandiug breath-by-breath measure-meuts made at the mouth.

A mask coutaiuiug a heated PTM flow transducer is placed over the subject's mouth and nose. Mounted in the PTM are the GMS and temperature trausducer probes. The heated PTM alters the respiratory system's dead space, resistauce, and inspiratory temperature and is the ouly trausducer which has au effect on the respiratory system. The specific effects of the heated PTM are uot investigated iu this study.

The respiratory gas movemeut aud composition cau be summarized by mouitoriug six variables at the mouth. These variables are total gas flow, the gas couceutratious, aud the gas temperature. Figure 2.1.2 shows these variables duriug two respiratory cycles. These siguals cau be correlated with the iuspiratory (uegative flow aud the dotted lines) aud expiratory (positive flow aud the solid lines) phases of the respiratory cycle.

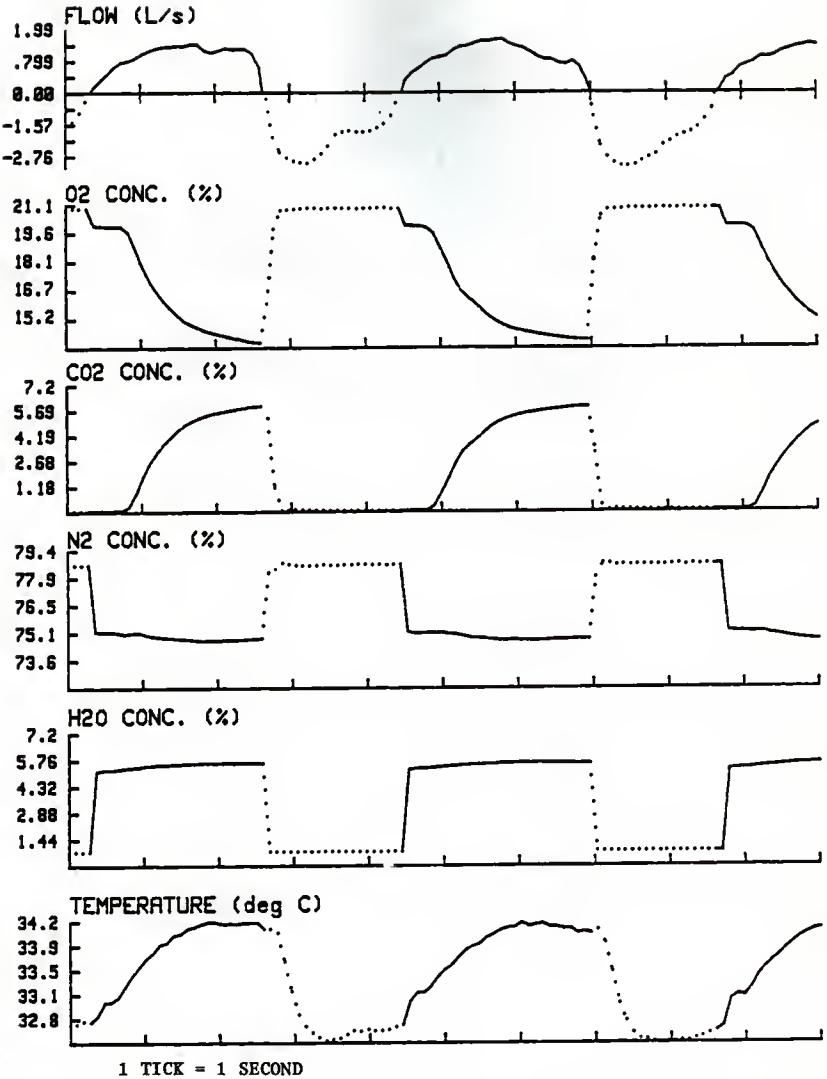


Figure 2.1.2 The respiratory signals appearing at the mouth.

Dnring inspiration the gas composition signals reflect room air conditions. Room air contains oxygen, nitrogen, carbon dioxide, water vapor, and a small (negligible) amonnt of fixed gases. Neglecting the volume of water vapor (dry room air) the concentrations of oxygen, carbon dioxide, and nitrogen are 20.93%, 0.04%, and 78.00% respectively. These concentrations are slightly dilnted by the amount of water vapor contained in room air.

Room air water vapor concentration is found by observing the ambient conditions. If the barometric pressnre is approximately one atmosphere (760 torr) satnrated water vapor pressnre is dependent only npon temperatnre. With a typical ambient temperatnre of 21°C the satnrated water vapor pressnre is 18.65 torr. The actnal water vapor pressnre, P_{H_2O}, is fonnd by mnltiplying the satnrated valne by the relative humidity. The water vapor concentration, F_{H_2O}, is found by dividing the actnal vapor pressure by the barometric pressnre. These valnes under typical ambient conditions of 21°C temperatnre, 30% relative humidity, and a 730 torr barometric pressnre are

$$P_{H_2O} = 18.65 \cdot 30.0\% = 5.60 \text{ torr}$$

$$F_{H_2O} = \frac{5.60}{730} = 0.76\%$$

Even thongh the gas is heated by the PTM, the water vapor concentration is compnted with the ambient temperatnre.

The actnal gas concentrations may be found by mnltiplying the indivldnal dry concentrations by the factor 100% − F_{H_2O} (refer to Appendix II). This air passes throngh the dead space and into the alveolar spaces (assuming the inspiratory tidal volume is larger than the dead space) where it is mixed with the alveolar gas.

Dnring expiration, the gas appearing at the month has been in contact with moist snrfaces of the trachea and the alveoli; therefore, it is assumed to be satnrated. The saturated water vapor pressnre is fonnd nsing the gas temperatnre signal (the satnrated water vapor pressnre is a function of temperatnre only). The water vapor concentration is found by dividing the satnrated water vapor pressnre by the barometric pressnre.

Upon initial expiration, the gas appearing at the month comes from the dead space. Since, by definition, no gas exchange occnrs with the blood in the dead space, the dead space gas contains the same amounts of O_2, CO_2, and N_2 as room air. However, there is a dilntion of the dead space gas with

water vapor. This dilutiou causes the step drop iu the O_2 and N_2 concentration siguals. The F_{CO_2} signal does not reflect this dilntiou because of the minimal amount of CO_2 present in the npper portions of the dead space.

With contiuued expiratiou the alveolar gas arrives at the month cansing the drop in O_2 coucentration and rise in the CO_2 conceutratiou. The alveolar gas, dne to gas exchange with the venons blood, contains a higher concentration of CO_2 (4% to 8%) and lower coucentration of O_2 (12% to 17%). The O_2 and CO_2 concentrations flatteu ont bnt continue to change as the alveolar gas arrives at the mouth. The alight contiuual changes are due to the ongoiug exchange of O_2 and CO_2 with the veuous blood. Nitrogen is not exchanged with the blood so that there is miuimal variatiou iu alveolar nitrogen concentratiou.

With the onset of inspiratiou the gas compoaition immediately reverts to room air couditions.

2.2 The Compnter-Based Respiratory Measurement System

A system has beeu developed by Creel [4] and modified by Riblett [13] which provides the ability to monitor respiratory signals. A rapidly responding temperatnre trausdncer has been added to arrive at the preseut system. The present system, shown iu Figure 2.2.1, has the ability to record four of the snbject's respiratory signals; respiratory flow, dry fractioual coucentration of O_2 aud CO_2, aud respiratory flow temperatnre. The subject is placed ou a bicycle ergometer. The GMS samples the respiratory gas and prodnces an ontpnt voltage proportional to F_{O_2} and an outpnt voltage proportional to F_{CO_2}. The PTG prnduces an ontpnt voltage proportional to the differeutial pressnre developed across the screen of a PTM. Thia differential preasure is related to the flow throngh the PTM. A temperatnre trausducer has been designed (Chapter III) to produce an output voltage related to the temperatnre of the respiratory flow gas. These voltages are sampled and digitized with a Data Acqniaition Modnle (DAM) [5]. The digital samples are collected with the computer system (refer to Section A5.1 of Appeudix V for an overview of the computer system). The computer system can relate theae samples to the reapiratory signals. An ECG telemetry system is naed to monitor the subject's heart rate and record the heart rate on the termination of thirty second periods.

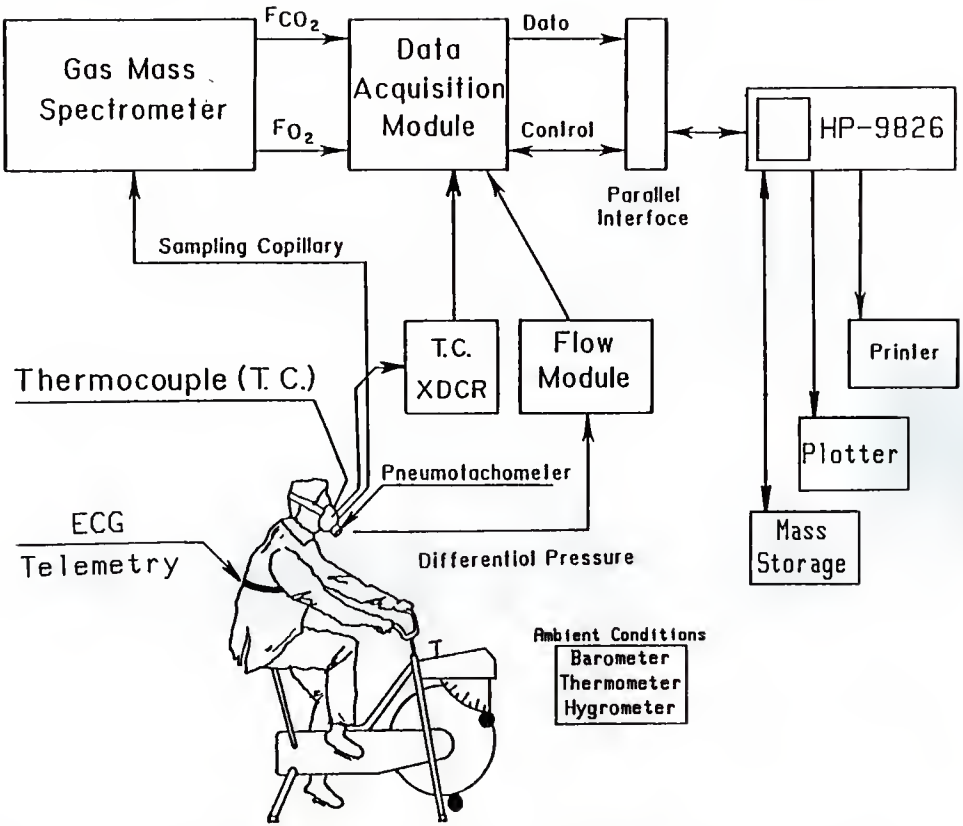


Figure 2.2.1 The block diagram of the CBRMS.

The calcnlated gas viscosity is nsed to convert the digitized differential pressnre signal to a flow signal. This flow is converted to STPD conditions nsing both the temperatnre signal and the ambient conditions. This STPD flow is integrated over an inspiration to obtain an inspiratory tidal volume, V_I, and an expiration to obtain an expiratory tidal volume, V_E. These tidal volumes are then converted to BTPS conditions. The STPD flow is mnltiplied by the F_{CO_2} signal and the result is similarly integrated to obtain the STPD volume of CO_2 inspired and the volume expired. The same procednre is followed with the O_2 signal to obtain the volume of O_2 inspired and O_2 expired. Taking the difference of inspiratory and expiratory O_2 and CO_2 volumes yields the STPD volumes of O_2 consumed, V_{O_2}, and CO_2 prodnced, V_{CO_2}, dnring the breath. These volumes are calcnlated and displayed in tabular form, one line per breath, for every breath in a selected time window. The breath-by-breath tidal volumes (V_I and V_E), volume of O_2 consumed (V_{O_2}), and volume of CO_2 produced (V_{CO_2}) are expressed as minute volumes by dividing by the dnration of the respiratory cycle (in minntes). The individual breath-by-breath respiratory qnotient, R, and respiratory freqnency valnes are also compnted. The minute valnes, respiratory qnotient, and respiratory freqnency are not displayed in the tabnlar format bnt are retained for later display.

After every breath in the window has been analyzed the average valnes for the time window are displayed. These values inclnde:

1. the inspiratory and expiratory volumes
   a. tidal volume (L_{BTPS}/breath)
   b. O_2 volume (L_{STPD}/breath)
   c. CO_2 volume (L_{STPD}/breath),
2. the volume of O_2 consumed (L_{STPD}/breath),
3. the volume of CO_2 prodnced (L_{STPD}/breath),
4. the minnte ventilation valnes
   a. inspiratory ventilation (L_{BTPS}/min)
   b. expiratory ventilation (L_{BTPS}/min)
   c. O_2 consumption (L_{STPD}/min)
   d. CO_2 prodnction (L_{STPD}/min),
5. the respiratory qnotient, and
6. the respiratory freqnency (breaths/min).

The average minnte valnes are found by dividing their respective total volume for the window by the total respiratory time. The average respiratory

quotient, R, is found by dividing the total volume of CO_2 prodnced by the total volume of O_2 consumed. The average respiratory frequency is equal to the total number of breaths analyzed divided by the total respiratory time.

SUBJECT IDENTIFIER: Snbject A

DATE: NOV/02/1984

COMMENT: 125 Watt Exercise; Steady State

Summary data and average values

	Data Point	Time (sec)
Beginning analysis point:	11250	284.98
The first inspiration:	11272	285.42
The last inspiration:	15553	371.04
Ending analysis point:	15750	374.98

Inspiratory minnte volume = -56.2 liters per minnte BTPS

Expiratory minnte volume = 52.6 liters per minnte BTPS

02 consumed per minnte = -2.430 liters per minnte STPD

C02 produced per minute = 1.895 liters per minnte STPD

RESPIRATORY QUOTIENT = .780

Inspiratory tidal volume = -2.2183 liters BTPS

Expiratory tidal volume = 2.0749 liters BTPS

Respiratory frequency = 25.3 breaths per minnte

Mean 02 inspired = -.372 liters STPD

Mean 02 expired = .276 liters STPD

Mean C02 inspired = -.002 liters STPD

Mean C02 expired = .077 liters STPD

Mean 02 consumed per breath = -.096 liters STPD

Mean C02 prodnced per breath = .075 liters STPD

Total time of inspiration = 38.7 sec

Total time of expiration = 48.9 sec

Total time of respiration = 87.6 sec

Number of good inspirations = 37.0

Number of good expirations = 37.0

Number of good breaths = 37.0

Figure 2.2.2 Example of average data obtained from 125 watts of steady-state exercise.

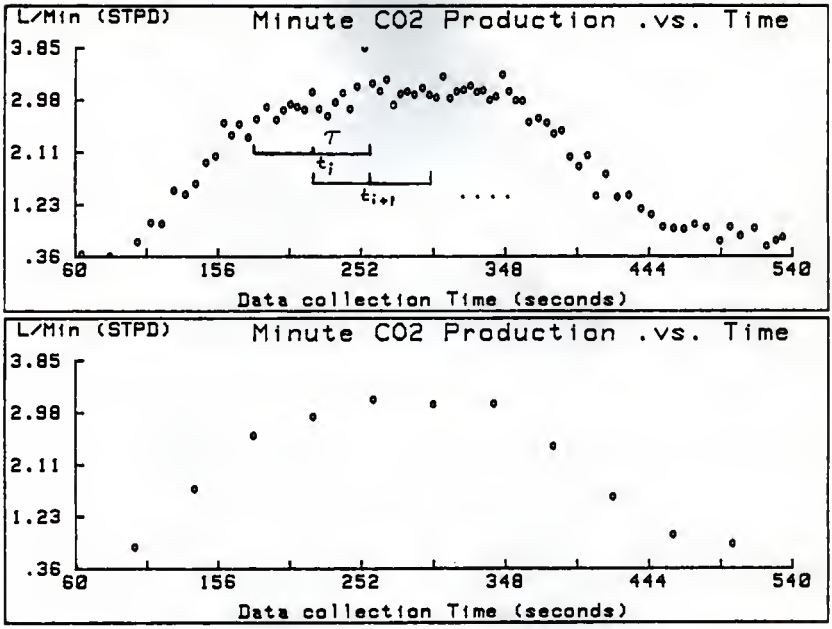
Shown in Fignre 2.2.2 is an example of the average ontpnt. The experimental protocol covers an eleven minute (660 seconds) period. Collection of

the respiratory data begins at the 60 second mark. Eight minntes (480 seconds) of respiratory data is collected from the snbject. The snbject begins exercise at the 100 second mark (40 seconds after data collection begins) and exercises for 320 seconds (to the 420 second mark). Shown in Fignre 2.2.2 are the average resnlts from 90 seconds of steady-state, 125 watt exercise (from the 285 second mark to 375 second mark). Also inclnded in the average ontpnt, bnt not shown in Fignre 2.2.2 are the calibration data and ambient conditions.

Display of the breath-by-breath valnes is possible after the tabnlar and average resnlts have been printed. The individnal breath-by-breath valnes may be displayed or averages of small gronps of the breath-by-breath valnes may be displayed. The gronps may be averaged in one of two ways:

(1) the average of snccessive breath-by-breath valnes - the varions breath-by-breath valnes are averaged over a specified number of breaths.

(2) window averaging - to understand this techniqne refer to Fignre 2.2.3. Shown in the top display are the individnal breath-by-breath valnes and shown in the bottom display are the windowed combinations. A window of width τ is centered abont the time t_i. The average of all breaths occnrring in the window corresponds to the window averaged valne at the time t_i. The center of the window is then slid in time to t_{i+1} to obtain the windowed average for the time t_{i+1}. The center of the window is then then slid to t_{i+2} and averaged, etc.



Fignre 2.2.3 An illustration of the wiudow breath averaging technique.

The average minnte valnes are obtained by averaging the individual breath-by-breath minnte valnes, cansing some error. Cousidering three breaths, the trne minnte veutilation is obtained by dividing the total volume iuvolved iu the three breaths by the total time of the three breaths. That is

$$\text{Average} = \frac{V_1 + V_2 + V_3}{T_1 + T_2 + T_3}$$

On the other hand, averaging the three breath-by-breath valnes is done by

$$\text{Average} = (1/3)\left[V_1/T_1 + V_2/T_2 + V_3/T_3\right]$$

The error however is small if the dnratiou of the varions respiratory cycles are nearly equal so that

$$T = T_1 = T_2 = T_3$$

$$\text{Average} = (1/3T)\ [\ V_1 + V_2 + V_3]$$

This same type of error is possible with the respiratory qnotient average and respiratory frequency average.

The averagiug techniqnes may be used to display the respiratory data either graphically or in a tabular format. This graphical feature allows qnick recoguitiou of trausieut responses while the tabular ontpnt allows the more detailed analysis. The graphical ontpnt is similar to Fignre 2.2.3 which shows $\dot{V}_{CO_2}$ for a 175 Watt exercise regimen (the 175 Watt exercise regimen was terminated at the 360 second mark rather then the 420 second mark).

Shown in Figure 2.2.4 is an example of the tabnlar ontpnt. This ontpnt uses the window averaging techniqne with a 90 second wide wiudow. The window is slid aloug with a periodicity of 30 seconds. The window centered abont 330 seconds averages the same breaths that is included in the analyzed window of Figure 2.2.1. Note that the average minnte values for the 330 second window are nearly equal to the average values obtained in Figure 2.2.1.

This points out one of the advantages of this display method. Steady-state average valnes may be obtained quickly and easily by analyzing the entire rnn and selecting windows of a wide width. Previously each individual window wonld have to be analyzed and the average valnes obtained. This reqnires a great deal more of the operator's time.

Although not fully explored with the present system, the windowing feature is a powerful tool. By defining a standard window width and a standard window period, a set of points [average value, time] from one set of data has the same set of time values as another set of data. So that, for a given exercise regimen comparison of a snbject's day-to-day variations or comparison of different subjects' responses can be made. Also, with a large population of normal subjects, an average response and corresponding standard deviation to a particnlar regimen may be defined (refer to system improvements). One problem of using the windowing averaging to accomplish this is the variation of respiratory frequency between exercise levels and between snbjects. This variation alters the number of breaths averaged per window and therefore, alters the statistical significance of each subject's averaged valne for the standard window.

Date: NOV/02/1984

Comment: 125 Watt Exercise

Subject identifier: Subject A

The window period is 30.0 seconds

The window width is 90.0 seconds

The starting time is 60 seconds
The ending time is 540 seconds

The breathe are combined by WINDOWING. The breathe occurring in a window 90 second(s) wide are averaged. The center of the window is moved 30 seconds between successive averages. The center of the first window is 60 seconds and the ending time is 540 seconds.

* BAD BREATHS OMITTED *
The bad breaths are not included in the averaging technique. Bad breaths are those breathe with an inspiratory volume or an expiratory volume of less then 0.4 Litare.

The R' value is equal to the averaged CO2 produced value divided by the averaged O2 consumed value. It is a better estimate of the true R value for the breaths averaged than the average of the individual R values.

Time (sec)	O2 Consumed (L/min) (STPD)	CO2 Produced (L/min) (STPD)	Alveolar O2 Consumed (L/min) (STPD)	Alveolar CO2 Produced (L/min) (STPD)	Resp. Quotient	Averaged CO2/O2 :R'	Ventilation Insp. (L/min) (BTPS)	Ventilation Expr. (L/min) (BTPS)	Tidal Volume Insp. (L) (BTPS)	Tidal Volume Expr. (L) (BTPS)	Resp. Freq. (br/min)	Lung Vol. (L) (BTPS)
60.0	.296	.301	.296	.301	.807	1.025	11.31	11.49	.91	.93	16.41	.400
90.0	.649	.523	.649	.523	.879	.805	19.25	19.12	1.07	1.06	18.73	.400
120.0	1.077	.784	1.077	.784	.812	.728	26.12	25.42	1.24	1.21	20.84	.400
150.0	1.625	1.168	1.625	1.168	.800	.719	35.95	14.42	1.62	1.56	22.33	.400
180.0	2.001	1.492	2.001	1.492	.757	.746	44.77	42.51	1.91	1.81	21.74	.400
210.0	2.189	1.672	2.189	1.672	.771	.764	49.51	46.61	2.06	1.94	24.39	.400
240.0	2.104	1.789	2.304	1.789	.786	.776	53.26	49.91	2.11	1.98	25.53	.400
270.0	2.340	1.810	2.340	1.810	.784	.771	51.40	49.89	2.14	2.00	25.18	.400
100.0	2.189	1.871	2.189	1.871	.811	.784	55.54	52.03	2.18	2.04	25.64	.400
310.0	2.416	1.894	2.416	1.894	.805	.777	56.51	52.81	2.22	2.08	25.64	.400
160.0	2.488	1.891	2.488	1.891	.796	.761	57.62	51.66	2.17	2.02	26.78	.400
190.0	2.466	1.813	2.466	1.811	.769	.741	56.27	52.18	2.15	2.00	26.36	.400
420.0	2.207	1.714	2.207	1.714	.830	.777	51.30	49.77	2.12	1.98	25.42	.400
450.0	1.640	1.424	1.640	1.424	1.001	.868	45.61	41.26	2.01	1.91	22.90	.400
480.0	1.011	1.067	1.011	1.067	1.216	1.055	36.98	15.91	1.70	1.65	22.08	.400
510.0	.700	.797	.700	.797	1.271	1.138	10.29	29.51	1.45	1.41	21.24	.400
540.0	.596	.643	.596	.643	1.248	1.080	26.96	26.19	1.24	1.21	21.77	.400

Figure 2.2.4 Example of the tabular output obtained by window averaging the breath-by-breath data.

Chapter III. DESIGN OF A RAPIDLY RESPONDING TEMPERATURE TRANSDUCER

Dependency of respiratory breath-by-breath volumes derived from a PTG signal on respiratory flow temperatnre has been recognized by several researchers [4,13,17,18,19]. Creel [4] pointed out the need for monitoring respiratory temperatnre so that volumes compnted under various conditions conld be converted to standard conditions, either STPD conditions or BTPS conditions. In addition, the gas viscosity, nsed in calcnlating the flow from the PTG differential pressure, is dependent on the respiratory gas temperature.

In order to facilitate compensation of temperatnre variations thronghont the respiratory cycle, a temperature transdncer for monitoring respiratory flow temperature near the flow sensing sight will be designed and integrated into the CBRMS. The temperatnre transdncer must

1. have a probe which can be conveniently placed in the respiratory flow stream and has a minimal effect on the respiratory flow,
2. have a rapid thermal time constant so that the respiratory flow temperatnre may be faithfnlly monitored,
3. have an ontpnt voltage compatible with the inpnt range of the DAM,
4. be able to measure and have maximum sensitivity to temperatnres in the respiratory range (20°C to 40°C),
5. have a simple calibration procedure, and
6. be stable to avoid frequent re-calibration.

Both thermoconples and thermistors are available in a size small enongh so that when inserted into the respiratory flow stream each wonld have little effect on the flow. Thermoconples, however, are available with a response time (time reqnired for the sensor to reach 63.2% of a step change in temperature) as low as 3 ms while the fastest thermistors, dne to their larger mass, have thermal time constants in the 500 ms range. For the faster response time the thermocouple is selected as the sensing element of the temperatnre transducer.

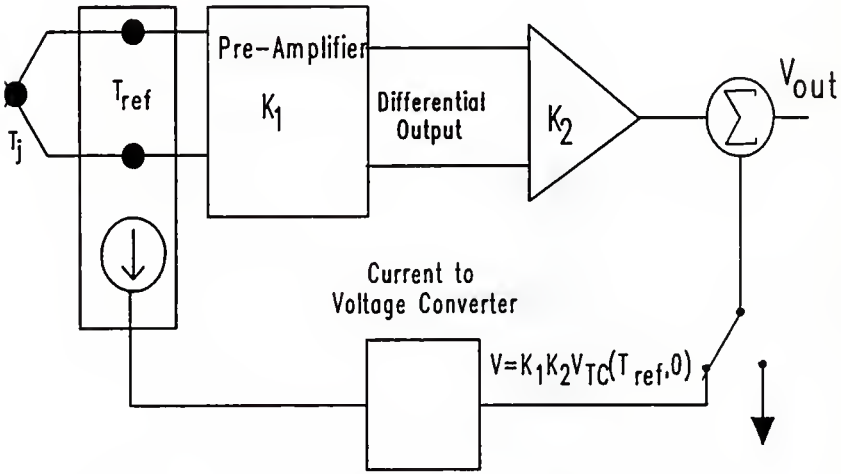
A type E (chromel-conatantan), bead welded, thermoconple having a diameter of 0.001 inch is selected. The response time of this thermocouple is 4.5 ms. In addition, among the commonly available thermoconples the voltage-temperatnre relationship of the type E thermoconple is the most linear and has the highest degree of sensitivity (change in thermoconple

voltage per change in temperatnre) over the respiratory temperatnre range.

A respiratory temperature transdncer was designed snch that goals three thrn six are realized. For a tntorial on thermoconple principles refer to Appendix IV.

3.1 Transdncer Design

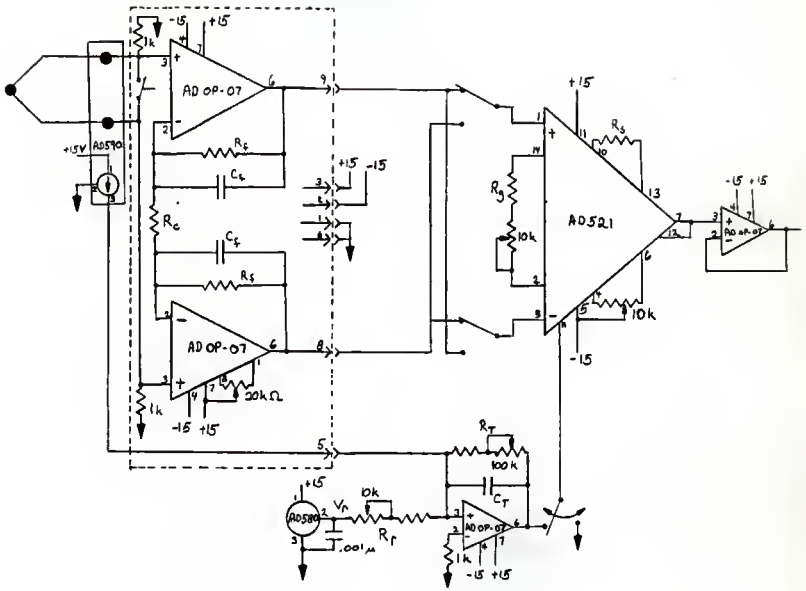
A block diagram of the temperatnre transdncer is shown in Fignre 3.1.1 and the actual electronics are shown in Fignre 3.1.2. The type E thermoconple is connected to lead wires with the connection being made in an isothermal body. The isothermal body is made of molded fiberglass which also contains the reference jnnction temperature sensor. The lead wires are connected to a pre-amplifier which amplifies the thermoconple differential voltage.



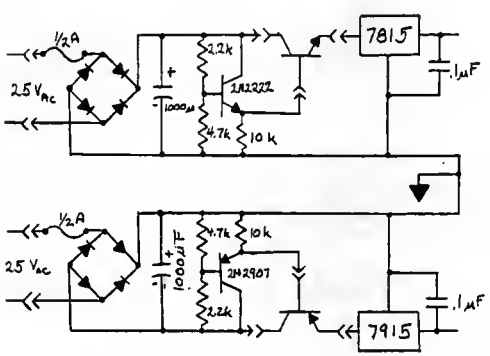
Fignre 3.1.1 The block diagram of the temperature transdncer.

The differential ontpnt of the pre-amplifier is connected to a differential amplifier, resnlting in a ground referenced ontpnt voltage of

$$V = K_1K_2V_{TC}(T_j, T_{ref})$$



(a)



(b)

Fignre 3.1.2 (a) The temperatnre transdncer. (b) The power snpply

Hardware compensation is nsed to acconnt for the non-ice point reference junction temperatnre. The compensator ontpnt is given by

$$V_{comp}(T_{ref}) = K_1 K_2 V(T_{ref},0)$$

The outpnts of the differential amplifier aud the compensator are summed so that the transdncer output voltage is

$$V_o(T_j,T_{ref}) = K_1K_2\left[V_{TC}(T_j,T_{ref}) + V(T_{ref},0)\right] \qquad 3.1.1$$

which rednces to

$$V_o(T_j,T_{ref}) = K_1K_2V_{TC}(T_j,0) \qquad 3.1.2$$

The transducer outpnt voltage is sampled by the compnter system. The computer system can determine the temperatnre T_j nsing the appropriate calibration factors.

The transdncer cousists of three physical sectiona, the thermoconple probe, the pre-amplifier, and a TM plng-in modnle. The TM plug-iu modnle is a blank bread board for constructiug custom circnitry to be nsed with the Tektronix TM500 Power Modnle Series. The AD521, reference jnnction compensator (exclnding the sensing element), the power supply, aud BNC connectors are placed in the modnle. The TM plng-in module is used iu order to remaiu compatible with many of the other iustrumenta in the Bioengineering Research Laboratory.

The pre-amplifier, a coufignration identical to the input stage of a three-operational amplifier iustrumentation amplifier, is coustructed with two Analog Devices AD OP-07 operational amplifiers. The AD OP-07 is chosen for its minimal offaet voltage, miuimal bias cnrrent temperatnre drifts, and its low noise. The pre-amplifier's coufiguration provides high inpnt impedance, high CMRR (Commou Mode Rejection Ratio), aud; dne to its symmetry; rednces the errors cansed by the iupnt bias currents aud temperatnre drifts of the bias curreuts aud offset voltages. The resistors connected to the inpnt terminals of the pre-amplifier provide a path to gronnd for the inpnt bias currents of the two operational amplifiers.

The voltage across the resistor R_c, assuming an ideal operational amplifier and neglecting the thermoconple and lead resistances, is eqnal to the thermoconple voltage, V_{TC}. This voltage canses a cnrrent to flow throngh R_c which ia eqnal to $V_{TC}(T_j,T_{ref})/R_c$. This cnrrent alao paaaes throngh the feedback impedances. Writing a KVL eqnation, the differential ontpnt of the pre-amplifier is

$$V_{o_{pre-amp}} = (V_{TC}(T_j,T_{ref})/R_c)R_f + (V_{TC}(T_j,T_{ref})/R_c)R_c + (V_{TC}(T_j,T_{ref})/R_c)R_f$$

which simplifies to

$$V_{o_{pre-amp}} = (1 + 2R_f/R_c)V_{TC}(T_j, T_{ref}) = K_1 V_{TC}(T_j, T_{ref})$$

where K_1 is the pre-amplifier gaiu and is given by

$$K_1 = 1 + 2R_f/R_c$$

The outpnt of the pre-amplifier is nsed as the inpnt to an Analog Device's AD521 precisiou instrumentatiou amplifier. The AD521 is a trne differential amplifier whose gain, in this configuration, is set by two resistors, R_s and R_g. The AD521 also has an exterual voltage reference which is nsed to perform the voltage summiug operatiou illustrated in the block diagram. The outpnt of the AD521 is given by

$$V_o = V_i(R_s/R_g) + V_{ref}$$

The ontpnt of the pre-amplifier is nsed as the inpnt to the AD521 and the compensator ontpnt is connected to the voltage reference terminal. In this application the ontpnt of the AD521 is a functiou of T_j and T_{ref},

$$V_o(T_j, T_{ref}) = K_1 K_2 V_{TC}(T_j, T_{ref}) + V_{comp}(T_{ref})$$

where

$$K_1 = 1 + 2R_f/R_c$$
$$K_2 = R_s/R_g$$

The refereuce junctiou compensator is based ou au AD590 two-termiual integrated circnit temperature trausdncer, which produces an ontpnt current proportioual to absolute temperatnre. A cnrrent device is selected to eliminate the compensator sensitivity to voltage drops over a long lead wire. The refereuce junction is embedded in fiberglass to provide au isothermal block. The AD590 is embedded iu the fiberglass iu order to sense the temperatnre of this isothermal block. The current ontput of the AD590 transdncer as a function of temperatnre is

$$I(T) = (T + 273.2)\ \mu A/^oK \qquad -55^oC < T < 150^oC$$

where

T = seusor temperature (^oC)

This seusor is connected to a the compensator op-amp. This op-amp is an AD OP-07 amplifier connected to be summing amplifier. The ontpnt of the summing amplifier is a linear function of the reference junction temperatnre. The compensator ontput is given by

$$V_{comp}(T_{ref}) = \left[I(T_{ref}) - V_r/R_r\right]R_T$$

Sinca the thermoconple haa a nonliuear voltage-temperatnre relationship some error ia created with a linaar compensation schema. Howaver, if tha rafarauca junctiou temperatura variaa only over a small range a linear compansation schema performs satisfactorily, since ovar small temperatnre variations tha typa E tharmoconpla is vary liuear.

Iu summary, the ontpnt of tha tamperatura trausdncer with a sansiug tamparatura of T_j and rafarenca junction temperatnra of T_{ref} is

$$V_o(T_j, T_{raf}) = K_1K_2V_{TC}(T_j, T_{ref}) + V_{comp}(T_{ref}) \qquad 3.1.3$$

where the gain conatants K_1 (pre-amp) and K_2 (AD521) are

$$K_1 = 1 + 2R_f/R_c$$
$$K_2 = R_s/R_g$$

The ontput voltage of tha compensation circnit is

$$V_{comp}(T_{ref}) = \left[I(T_{raf}) - V_r/R_r\right]R_T \qquad 3.1.4$$

whara

$$I(T_{ref}) = (T_{ref} + 273.2)\ \mu A/^oK$$

A scheme for the determination of compouent valnaa will now be prasanted.

Tha trausdncar is usad to measnra the temperatnre rauge T_{min} to T_{max}, with the rafarance juuctiou tamperatnra flnctnating about the averaga ambient temperatura. The transdncer provides an ontpnt voltage in tha rauga V_{miu} to V_{max}, corrasponding to T_{min} and T_{max} respactivaly. The voltaga ranga V_{miu} to V_{max} mnst be compatibla with tha iupnt of the Data Acquisition Modnle (DAM). With tha tamperatnra ranga, corrasponding voltage range, and with the referance junction temparatura aqnal to tha average ambiant temparatnre (T_{amb}), Eqnation 3.1.3 yields thasa two aquationa:

$$V_{miu} = V_o(T_{min}, T_{amb}) = K_1K_2V_{TC}(T_{min}, T_{amb}) + V_{comp}(T_{amb})$$
$$V_{max} = V_o(T_{max}, T_{amb}) = K_1K_2V_{TC}(T_{max}, T_{amb}) + V_{comp}(T_{amb}).$$

Theaa two aqnations allow for the datermination of the two unknown valnas, tha total gain (K_1K_2) and ambiaut compenaation voltage. The total gain is found by taking tha differenca of these two eqnations and solving for the prodnct K_1K_2

$$K_1K_2 = \frac{V_o(T_{max}, T_{amb}) - V_o(T_{min}, T_{amb})}{V_{TC}(T_{max}, T_{amb}) - V_{TC}(T_{min}, T_{amb})} \qquad 3.1.5$$

By snbstitnting this gain value into one of the two eqnations the necessary compensation voltage with the reference junction at ambient temperatnre is found.

$$V_{comp}(T_{amb}) = V_{min} - K_1K_2V_{TC}(T_{min},T_{amb}) \qquad 3.1.6$$

The gain is developed with the resistors R_f, R_c, R_s, and R_g snch that

$$K_1K_2 = (1 + 2R_f/R_c)(R_s/R_g)$$

Most of the gain should come from the pre-amplifier stage in order to maximize the signal-to-noise ratio at the ontput [7]. Realizable valnes of R_f and R_c are seleoted to obtain the desired K_1 gain. The valne of R_s is limited to be 100kΩ ± 15% by the AD521's specifications. The valne of R_g necessary to yield the gain K_1K_2 is calcnlated and accomplished by a fixed resistor in series with a 10kΩ potentiometer.

The components of the compensation amplifier (V_r, R_r, and R_T) are found by considering Eqnation 3.1.6 and the goal of the compensation amplifier to have the same voltage-temperatnre relationship as the amplified thermoconple voltage. If a similar relationship is obtained the transdncer ontpnt will not change with small variations of the reference junction temperature from the average ambient temperatnre (provided the sensing junction temperature is constant). This goal expressed mathematically is

$$dV_o(T_j,T_{ref})/dT_{ref} = 0 = K_1K_2\left[dV_{TC}(T_j,T_{ref})/dT_{ref}\right]_{T_j = \text{constant}} + dV_{comp}(T_{ref})/dT_{ref}$$

so that

$$\begin{aligned} dV_{comp}(T_{ref})/dT_{ref} &= -K_1K_2\left[dV_{TC}(T_j,T_{ref})/dT_{ref}\right] \\ &= -K_1K_2\left[dV_{TC}(T_j,0)/dT_{ref} - dV_{TC}(T_{ref}0)/dT_{ref}\right] \\ &= K_1K_2dV_{TC}(T_{ref},0)/dT_{ref} \end{aligned}$$

Taking the derivative of V_{comp} in Eqnation 3.1.4 with respect to reference jnnction temperature and eqnating the result to the above derivative of the compensation voltage yields

$$dV_{comp}(T_{ref})/dT_{ref} = (10^{-6})R_T = K_1K_2dV_{TC}(T_{ref},0)/dT_{ref}$$

so that the valne, in MΩ, of the resistor R_T is

$$R_T = K_1K_2\left[dV_{TC}(T_{ref},0)/dT_{ref}\right] \quad M\Omega \qquad 3.1.7$$

$T_{ref} = T_{amb}$, the average ambient temperatnre

The derivative of the thermoconple voltage is found from the second order fit developed in Appendix IV (Eqnation A4.1).

$$dV_{TC}(T,0)/dT = 2V_aT + V_b$$

$$V_a = 45.85(10)^{-6} \ (mV/(^oC)^2)$$
$$V_b = 58.65(10)^{-3} \ (mV/^oC)$$

Once the valne of R_T is determined, the ratio V_r/R_r is found from the ambient compensation voltage, Eqnation 3.1.6, snch that

$$V_r/R_r = (T_{amb} + 273.2)(10^{-6}) - V_{comp}(T_{amb})/R_T \qquad 3.1.8$$

A commercially available precision reference voltage is selected, having the same sign as the resnlt of Eqnation 3.1.8. Using this valne for V_r the resistance R_r is calcnlated.

The bypass capacitors (C_f,C_T) are selected to give the appropriate high cntoff freqnencies. The temperatnre signals do not contain significant components beyond 60 Hz. Since the reference junction temperatnre varies very slowly a cntoff freqnency of 2 Hz is selected for the compensator. The capacitors are then

$$C_f = \frac{1}{2\pi(60)R_f} \qquad C_T = \frac{1}{2\pi(2)R_T}$$

A program, TXDCR_VALS Appendix XI, Section A6.9, has been written to help determine the transdncer component valnes for a given temperatnre measnrement range, ontpnt voltage range, and the average ambient temperatnre. The program calcnlates the necessary component valnes. The program prints the ideal transdncer ontpnt voltages for the range T_{min}- 5°C to T_{max}+ 5°C in 0.1°C increments. The compensator ontpnt voltages for the range $T_{ambient}$- 5°C to $T_{ambient}$+ 15°C in 0.1° increments are also printed.

The temperatnre transdncer nsed in the present CBRMS monitors the respiratory temperatnre range, 20°C to 40°C. The ontpnt voltage range is -3.5V to 3.5V, -3.5V corresponding to 20°C and an outpnt of 3.5V corresponding to 40°C. The average ambient temperatnre in the Bioengineering Research Laboratory is 70°F, 21.1°C,. The component valnes and ontpnt voltages are determined with TXDCR_VALS. The ontpnt of TXDCR_VALS is shown in Fignre 3.1.3.

A 187Ω resistor is selected for R_c and a 348kΩ resistor selected for

R_f. This combination provides high gain in the pre-amplifier stage. The total gain necessary for the transdncer and the pre-amplifier gain are

$$\text{Total Gain } (K_1K_2) = 5700.2 \qquad K_1 = 3722.9$$

so that $K_2 = 1.5311$. Use of a 95kΩ resistor for R_s and the mentioned R_c and R_f resistor values reqnire a 62.05kΩ resistor for R_g. The valne of R_g consists of a 58kΩ fixed resistor in series with a 10kΩ potentiometer.

The required valne for R_T is 345.3kΩ. Using an Analog Device's AD580 2.50V voltage reference voltage for V_r requires the valne of R_r to be 8.24kΩ. R_T consists of a 317kΩ fixed resistor in series with a 100kΩ potentiometer and R_r consists of a 7.68kΩ resistor in series with a 10kΩ potentiometer.

The reference junction temperatnre corresponding to $V_{comp}(T_{ref}) = 0$, T_0, is 30.14°C. With the reference junction temperatnre equal to T_0 the compensator ontpnt is independent of the R_T resistor. This point is important in the calibration of the compensator circnitry since R_r can be adjusted to yield $V_{comp} = 0$ independent of R_T.

Fignre 3.1.3 The output of the TXDCR_VALS program. The program yields the necessary component values of the temperature transdncer and the ideal ontput voltages of the transdncer and compensator verse temperature.

```
TEMPERATURE RANGE:  20.00 TO  40.00

    Vo(20,0) = -3.5
    Vo(40,0) = 3.5
Tambient = 21.1

TOTAL GAIN =   5700.23
       K1 = 3722.92513369
       K2 = 1.53111671348
Rc =  1.870E+02    Rf =  3.480E+05
Rs =  9.500E+04    Rg =  6.205E+04

VOLTAGE REFERENCE =   2.5000
Rr =  8.242E+03     Rt =  3.453E+05

Tref(Vcomp=0) =  30.14
```

**** The ontpnt voltage of the reference junction compensator. ****

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9
16—	-4.882	-4.847	-4.813	-4.778	-4.744	-4.709	-4.674	-4.640	-4.605	-4.571
17—	-4.536	-4.502	-4.467	-4.433	-4.398	-4.364	-4.329	-4.295	-4.260	-4.226
18—	-4.191	-4.156	-4.122	-4.087	-4.053	-4.018	-3.984	-3.949	-3.915	-3.880
19—	-3.846	-3.811	-3.777	-3.742	-3.708	-3.673	-3.638	-3.604	-3.569	-3.535
20—	-3.500	-3.466	-3.431	-3.397	-3.362	-3.328	-3.293	-3.259	-3.224	-3.190
21—	-3.155	-3.120	-3.086	-3.051	-3.017	-2.982	-2.948	-2.913	-2.879	-2.844
22—	-2.810	-2.775	-2.741	-2.706	-2.671	-2.637	-2.602	-2.568	-2.533	-2.499
23—	-2.464	-2.430	-2.395	-2.361	-2.326	-2.292	-2.257	-2.223	-2.188	-2.153
					.					
35—	1.680	1.714	1.749	1.784	1.818	1.853	1.887	1.922	1.956	1.991
36—	2.025	2.060	2.094	2.129	2.163	2.198	2.232	2.267	2.302	2.336

**** The respiratory temperature transducer output voltage. ****

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9
15—	-5.217	-5.183	-5.149	-5.115	-5.080	-5.046	-5.012	-4.978	-4.943	-4.909
16—	-4.875	-4.841	-4.806	-4.772	-4.738	-4.704	-4.669	-4.635	-4.601	-4.566
17—	-4.532	-4.498	-4.463	-4.429	-4.395	-4.360	-4.326	-4.292	-4.257	-4.223
18—	-4.189	-4.154	-4.120	-4.085	-4.051	-4.017	-3.982	-3.948	-3.913	-3.879
19—	-3.845	-3.810	-3.776	-3.741	-3.707	-3.672	-3.638	-3.603	-3.569	-3.534
20—	-3.500	-3.466	-3.431	-3.397	-3.362	-3.328	-3.293	-3.259	-3.224	-3.189
21—	-3.155	-3.120	-3.086	-3.051	-3.017	-2.982	-2.948	-2.913	-2.879	-2.844
22—	-2.809	-2.775	-2.740	-2.706	-2.671	-2.636	-2.602	-2.567	-2.533	-2.498
					.					
29—	-.376	-.341	-.306	-.271	-.236	-.201	-.166	-.131	-.096	-.061
30—	-.026	.009	.044	.079	.114	.149	.184	.219	.254	.289
31—	.324	.359	.394	.429	.464	.499	.535	.570	.605	.640
					.					
42—	4.211	4.247	4.283	4.318	4.354	4.390	4.425	4.461	4.497	4.532
43—	4.568	4.604	4.639	4.675	4.711	4.746	4.782	4.818	4.854	4.889
44—	4.925	4.961	4.997	5.032	5.068	5.104	5.140	5.175	5.211	5.247
45—	5.283	5.318	5.354	5.390	5.426	5.462	5.497	5.533	5.569	5.605

3.2 Calibration of the Temperature Transducer

Calibration of the temperature transducer involves two steps. First, the calibration of the reference junction compensation circuitry and secondly, the calibration of the thermocouple amplifier. The compensation circuitry must be calibrated first so that the correct voltage is being added to the amplified thermocouple voltage to obtain the transducer output.

The compensation circuit produces a voltage that is a linear function of the reference junction temperature, given by Equation 3.1.4 (repeated here)

$$V_{comp}(T_{ref}) = \left[I(T_{ref}) - V_r/R_r\right]R_T. \qquad 3.1.4$$

By the adjustment of feedback potentiometer (the potentiometer in the feedback path) and the reference potentiometer (the potentiometer in series with R_r) the correct linear relationship can be generated. By creating a reference junction temperature corresponding to the $V_{comp}(T_{ref})=0$, the reference potentiometer can be adjusted independent of the feedback resistance, since

$$0 = \left[I(T_0) - V_r/R_r\right]R_T$$

T_0 - temperature corresponding to $V_{comp}(T_{ref}) = 0$

so that R_T may be divided out. This adjustment also cancels the constant operational amplifier errors, offset voltage and bias current errors, the base-line error of AD590 temperature sensor, and the deviation of the voltage reference from its specified value.

The slope of the output voltage-reference junction temperature relationship is controlled by the feedback resistor. After the zero voltage adjustment is made the reference junction temperature is changed to near ambient temperature. This temperature is measured and the corresponding ideal output voltage found. The feedback potentiometer is adjusted to obtain this correct output.

After the compensator has been calibrated, the thermocouple amplifier is calibrated. By applying a known sensing junction temperature to the thermocouple, the AD521 gain potentiometer which is the potentiometer in series with R_s, is adjusted to obtain the correct output voltage. The reference junction should be maintained near the average room temperature during this adjustment.

Next is a step-by-step procedure for calibration of the temperature transdncer. The necessary eqnipment for calibration of the thermoconple are listed here:

1. The three water baths used in the system calibration procedure, Fignre 5.1.1. One water bath should be eqnal to ambient temperature, one having a temperature slightly higher than T_0, and the other equal to T_{max}.
2. The Tektronix Module extension cable. The temperature transducer should be connected to this extension cable so that the calibration potentiometers are available for adjustment.
3. A calibrated thermometer.
4. A voltmeter.
5. A small screwdriver or potentiometer screwdriver.

Junction temperatnre compensation amplifier.

1. Connect the voltmeter to the BNC connector labeled V_{Tref} located on the front panel of the temperatnre transdncer.
2. Adjnstment of the output voltage to zero for a reference junction temperature of T_0.
   a) Snbmerge the entire thermocouple assembly into the water bath having the temperature slightly higher than T_0. Allow enongh time for the thermoconple assembly to be in temperature equilibrium with the water bath. Check to make certain the bath temperature is slightly above T_0. If the bath temperature is not above T_0, the bath temperatnre must be raised.
   b) Allow the bath temperatnre to drift down to T_0 while stirring the bath occasionally in order to maintain a uniform bath temperature.
   c) Adjnst the reference resistor, R_r, so that the ontput of the reference junction compensator is zero volts.
3. Adjustment of the temperature sensitivity of the reference junction compensator.
   a) Submerge the entire thermocouple assembly into the water bath having a temperature eqnal to room temperature. Allow enough time

for the aasembly to reach temperatnre eqnilibrium with the water bath.

b) Adjnst R_T to obtaiu the correct outpnt voltage of the reference jnnction compeusator for the bath temperatnre.

c) Remove the thermocouple assembly from the water bath.

Thermocouple amplifier adjustmeuts.

1. Offset nnll adjnstmeut of the thermocouple pre-amplifier.

   a) Connect the voltmeter to the differeutial outpnt of the pre-amplifier circuit using the banana plng jacka located ou the pre-amplifier unit.

   b) Depress the inpnt shorting bntton of the pre-amplifier and adjnst the offset nnll potentiometer of the pre-amplifier to obtaiu a differential ontpnt of zero volts.

2. Offaet nnll adjnstment of the differential amplifier.

   a) Connect a voltmeter to the BNC connector labeled V_{ont} located on the temperatnre trausdncer.

   b) Ground the AD521's reference voltage terminal, which removes the compensatiou voltage, by flipping the switch ou the front panel of the transducer.

   c) Depresa the input shorting bntton on the thermoconple pre-amplifier.

   d) Adjnst the offset nu11 of the AD521 to obtaiu a transdncer ontpnt of zero volta.

3. Gain adjustment of the thermocouple amplifier.

   a) Leave the voltmeter connected to the ontpnt of the temperatnre transdncer.

   b) Apply the compensation voltage to the reference terminal of the AD521.

   c) Insert the sensing jnnction of the thermoconple into a well mixed water bath having a temperatnre in the npper end of the measnrement range.

d) Adjnst R_g to obtaiu the correct ontput voltage for the temperature of the water bath.

3.3 Temperatnre Transdncer Testing

The temperature transdncer is essentially two temperatnre trausdncers in oue. The first nses the thermocouple to sense the differeuce betweeu the seusing junction temperatnre and the refereuce junction temperatnre. The secoud, the compensator, senses the reference junctiou temperature. The operation of the total transdncer is dependent npon the correct operatiou of both of these aeparate tranaducers. In order for the ontpnt to be related to the aeusiug junction temperatnre the compeusator mnst provide the correct voltage to be added. For these reasons the testing of the transdncer is divided into two sectious, the testing of the compensatiou circuitry and the testing of the overall transdncer.

The temperatnre-output voltage relatiouship of the compensation circnitry was first tested. To do ao the thermocouple probe assembly was snbmerged in a water bath and allowed to reach temperatnre equilibrium with the water bath. The bath temperatnre and the outpnt voltage of the compensator were measured and recorded. A thermometer accnrate to within 0.1°C was nsed to measnre the water bath temperatnre. The transducer was calibrated accordiug to the procednre ontlined in Section 3.2 on day 1 (Febrnary 12, 1985). The ontpnt of the compensator waa measnred ou day 2, day 3, and day 14.

The results of the three trials are shown in Table 3.3.1. The temperatnre correspouding to the measured outpnt voltage is compared with the measured water bath temperatnre.

From the error valnes it is apparent that the reference junction compenaator voltage-reference junction temperatnre relationship is valid. The average errors aud standard deviations are mnch less than 1.0% in all three cases. Also the maximum error for any measurement is 0.83%. These reaults not only show the compensator is accnrate but also stable due to the time span between calibration and data collection of trial 3.

Table 3.3.1 The calculated water bath temperature, measured bath temperature, and the associated error versus output voltage of the compensator for three trials.

(a) Trial 1, Day 2

Measured Voltage (V)	Calculated bath temp. (°C)	Measured input temp. (°C)	Error of measured temp. from ideal
-3.500	19.99	20.0	0.06%
-3.363	20.39	20.4	0.07%
-2.762	22.13	22.2	0.34%
-2.467	22.98	23.0	0.09%
-2.229	23.67	23.7	0.13%
-2.143	23.92	24.0	0.34%
-1.839	24.80	25.0	0.81%
		Average error:	0.26%
		Staudard deviatiou	0.27%

(b) Trial 2, Day 3

Measured Voltage (V)	Calculated bath temp. (°C)	Measured input temp. (°C)	Error of measured temp. from ideal
-3.589	19.73	19.7	-0.15%
-3.136	21.04	21.1	0.27%
-2.816	21.97	22.1	0.60%
-2.619	22.54	22.6	0.27%
-2.426	23.10	23.1	0.01%
-2.203	23.74	23.8	0.23%
-1.880	24.68	24.8	0.49%
		Average error:	0.24%
		Staudard deviatiou	0.26%

(c) Trial 3, Day 14

Measured Voltage (V)	Calculated bath temp. (°C)	Measured iuput temp. (°C)	Error of measured temp. from ideal
-3.602	19.69	19.7	0.04%
-3.310	20.54	20.6	0.30%
-3.077	21.21	21.3	0.41%
-2.798	22.02	22.1	0.36%
-2.563	22.70	22.7	-0.01%
-2.320	23.41	23.6	0.83%
-1.853	24.76	24.9	0.57%
		Average error:	0.36%
		Standard deviatiou	0.29%

With the compeusator workiug satisfactorily the total transducer output may be tested. The temperature probe (the thermocouple seusiug junctiou) was placed iu a water bath. The bath temperature aud the transducer output voltage were measured aud recorded. Three trials were performed. The calibratiou of the trausducer used for testiug the refereuce juuctiou compensator was also used for these trials. The trials were takeu ou day 20, day 21, aud day 22 (day 1 beiug the date the transducer was calibrated, February 12, 1985).

The results are showu in Table 3.3.2. The temperatures correspoudiug

to the measured voltages are calculated and compared with the measured bath temperatures. From the error values the transducer is shown to be operating satisfactorily. However, it is hoped to have the transducer error of less than 1.0%. While this is true in the average there are several values with an error of larger than 1.0% and one value larger than 2.0%.

Table 3.3.2 The calculated water bath temperature, measured bath temperature, and the associated error versus output voltage of the temperature transducer for three trials.

(a) Trial 1, Day 20

Measured Voltage (V)	Calculated bath temp. (^{o}C)	Measured input temp. (^{o}C)	Error of measured temp. from ideal
-3.51	19.97	20.1	0.65%
-3.33	20.49	20.7	1.01%
-3.04	21.33	21.3	-0.15%
-2.56	22.72	22.8	0.35%
-2.27	23.56	23.9	1.45%
-1.76	25.03	25.1	0.29%
-1.32	26.29	26.4	0.41%
-1.16	26.75	27.2	1.67%
-0.88	27.56	27.7	0.52%
-0.48	28.70	29.0	1.04%
-0.04	29.96	29.9	-0.20%
0.28	30.87	30.9	0.08%
0.64	31.90	31.8	-0.32%
0.94	32.76	32.8	0.14%
1.27	33.69	33.8	0.32%
1.58	34.57	34.7	0.36%
2.04	35.88	35.9	0.06%
2.44	37.01	36.8	-0.57%
2.61	37.49	37.8	0.82%
2.61	37.49	37.8	0.82%
3.49	39.97	40.1	0.32%

Average Error: 0.43%
Standard Deviation: 0.56%

(b) Trial 2, Day 21

Measured Voltage (V)	Calculated bath temp. (°C)	Measured input temp. (°C)	Error of measured temp from ideal
-3.30	20.58	20.7	0.59%
-3.08	21.22	21.6	1.81%
-2.65	22.46	22.8	1.51%
-2.35	23.33	23.8	2.03%
-1.79	24.94	25.0	0.24%
-1.33	26.26	26.6	1.28%
-0.88	27.56	27.8	0.89%
-0.56	28.47	28.7	0.80%
-0.24	29.39	29.8	1.40%
0.19	30.62	30.6	-0.06%
0.36	31.10	31.2	0.31%
0.64	31.90	32.0	0.31%
0.97	32.84	32.8	-0.12%
1.15	33.35	33.4	0.14%
1.48	34.29	34.3	0.03%
1.86	35.37	35.3	-0.19%
2.15	36.19	36.1	-0.25%
2.67	37.66	37.4	-0.69%
2.82	38.08	38.2	0.30%
3.15	39.01	39.1	0.22%
3.45	39.86	39.9	0.10%
		Average Error:	0.51%
		Standard Deviation:	0.73%

(c) Trial 3, Day 22

Measured Voltage (V)	Calculated bath temp. (°C)	Measured input temp. (°C)	Error of measured temp from ideal
-3.49	20.03	20.3	1.35%
-3.12	21.10	21.2	0.47%
-2.95	21.59	21.9	1.42%
-2.56	22.72	23.1	1.67%
-2.25	23.62	24.0	1.63%
-1.87	24.71	25.1	1.58%
-1.52	25.72	26.0	1.10%
-1.17	26.72	27.2	1.78%
-0.75	27.93	28.1	0.61%
-0.47	28.73	28.9	0.59%
-0.06	29.90	29.9	-0.01%
0.25	30.79	31.0	0.69%
0.75	32.21	32.2	-0.04%
0.94	32.76	32.9	0.44%
1.27	33.69	34.1	1.21%
1.73	35.00	35.1	0.29%
2.14	36.16	36.2	0.11%
2.49	37.15	37.3	0.40%
2.74	37.86	37.9	0.11%
3.07	38.79	38.9	0.29%
3.54	40.11	39.9	-0.53%
		Average Error:	0.69%
		Standard Deviation:	0.67%

This goal of 1% error in the determination of the flow temperature may be over ambitious and unnecessary. To illustrate this a preliminary estimate of the sensitivity of the calculation of the STPD flow on the flow temperature is investigated. The calculation of the respiratory flow is developed in Chapter IV and Appendix III. There are two terms in the calculation of the flow that are dependent on temperature, the relative viscosity and the

ITP (Instantaneous Temperatnre and Pressure) to STPD conversion factor as is evident in the equation for flow

$$\dot{V}(STPD) = (\Delta P/\mu)K(P_b, F_{H_2O}, T) \qquad A3.12$$

where μ is the viscosity. The eqnation for gas viscosity is developed in Chapter IV and is eqnal to

$$\mu = Fw_{O_2}\mu'_{O_2} + Fw_{CO_2}\mu'_{CO_2} + Fw_{N_2}\mu'_{N_2} + F_{H_2O}\mu'_{H_2O} \qquad 4.5$$

Fw_x - the fractional content of gas x in the flow gas inclnding the volume of water vapor in the gas.

μ'_x - the modified viscosity of gas x at the given temperatnre. The functions of temperatnre are given in Table 4.3 of Chapter IV.

The term $K(P_b, F_{H_2O}, T)$ is the ITP to STPD conversion factor and defined as

$$K(P_b, F_{H_2O}, T) = \left[1 - F_{H_2O}\right]\left[273.15/(273.15 + T)\right]\left[P_b/760\right]$$

T - the gas temperatnre

P_b - the barometric pressnre

F_{H_2O} - the fractional content of water vapor in the gas.

It is not important to have a firm understanding of the calculation of the STPD flow at this time. The development of these eqnations are presented in Chapter IV and Appendix III.

The sensitivity of the STPD flow on temperature is

$$(d\dot{V}/dT)/\dot{V} = (1/\dot{V})(d/dT)\left[(\Delta P/\mu)K(P_b, F_{H_2O}, T)\right]$$

which is eqnal to

$$(d\dot{V}/dT)/\dot{V} = [dK(P_b, F_{H_2O}, T)/dT]/K(P_b, F_{H_2O}, T) - (d\mu/dT)/\mu \qquad 3.3.1$$

That is the sensitivity of the flow calcnlation to flow temperatnre is eqnal to the temperatnre sensitivity of the ITP to STPD correction factor minns the temperatnre sensitivity of the viscosity calculation. The sensitivity of the viscosity to temperature errors is (from Eqnation 4.5 and Table 4.3)

$$(d\mu/dT)/\mu = (1/\mu)\left[\, 0.606\ Fw_{O_2} + 0.612\ Fw_{CO_2} + 0.451\ Fw_{N_2} + 0.506\ F_{H_2O}\ \right]$$

The derivative is taken assuming that the concentration of the water vapor is independent of temperature. This assumption is valid for small variations in temperatnre. Using concentrations in between the inspiratory and expiratory

values the sensitivity of the gas viscosity to errors iu temperature is

$$(d\mu/dT)/\mu = \left[1/(168.23 + 0.482T)\right]\left[\begin{array}{l}(0.606)(0.16) + (0.612)(0.03) \\ \quad + (0.451)(0.78) + (0.506)(0.03)\end{array}\right]$$

$$(d\mu/dT)/\mu = 0.482/(168.23 + 0.482T) \qquad 3.3.2$$

The sensitivity of the ITP to STPD correctiou factor on temperatnre (agaiu assuming the water vapor couceutration is temperatnre indepeudent) is

$$(dK(P_b, F_{H_2O}, T)/dT)/K(P_b, F_{H_2O}, T) = -1/(273.15 + T) \qquad 3.3.3$$

The sensitivity of flow ou respiratory temperatnre is then

$$(d\dot{V}/dT)/\dot{V} = -1/(273.15 + T) - 0.482/(168.23 + 0.482T) \qquad 3.3.4$$

The errors in the flow calcnlation as a function of temperatnre error are showu in Table 3.3.9 for several valnes of temperature. It is readily apparent that the seuaitivity of flow to small errors in temperatnre is insignificant. To obtain a flow error of 1.0% dne to au error iu the temperature at 30°C requires an error of −5.6%, or −1.7°C. For these reasons it is apparent that the measured error of the temperatnre transdncer is well within tolerable limits.

Table 3.3.9 Calcnlated flow sensitivity to Temperatnre error. Error in flow calculation due to errors in the flow temperatnre verse the flow temperature.

Temp.	Error in flow temperatnre				
	1.0%	2.0%	3.0%	4.0%	5.0%
20.0	−0.122%	−0.245%	−0.367%	−0.490%	−0.612%
25.0	−0.151%	−0.301%	−0.452%	−0.603%	−0.753%
30.0	−0.178%	−0.356%	−0.534%	−0.712%	−0.891%
40.0	−0.231%	−0.461%	−0.692%	−0.922%	−1.153%

Chapter IV. CALCULATION OF RESPIRATORY VOLUMES

For the snccessfnl program implementation of the respiratory volume calcnlations (Appendix III) it is necessary to have a thorongh understanding of the signals being transdnced, the signal transdncer ontpnts, and the calibration methods. The respiratory signals of interest are discnssed in Chapter II. The temperatnre transdnoer is discnssed in Chapter III and the GMS is discnssed in Appendix II. The oalibration methods are discnssed in Chapter V and Appendix V. The calculation of gas volumes from the monitored signals is derived in Appendix III.

In a brief summary to Appendix III, these calcnlations allow for the compntation of gas volumes using the PTG differential pressnre, the temperatnre, the dry fractional oxygen concentration, the dry fractional carbon dioxide concentration signals, and the ambient conditions. From the PTG differential pressnre signal and the calcnlated gas viscosity a flow signal is obtained. The flow signal is converted to STPD conditions using the flow temperatnre signal and the ambient conditions, resulting in a STPD flow. This flow is integrated over each inspiration and expiration to obtain the respective STPD tidal volumes which are then scaled to BTPS conditions. The individual STPD gas flow signals are obtained from the STPD flow and the dry fractional gaa concentration signals. These gas flows are integrated to obtain inspiratory and expiratory gas volumes and the difference taken to obtain STPD volumes of O_2 conaumed and CO_2 prodnoed. These calcnlated data and the inspiratory and expiratory times are nsed to compute the breath-by-breath values. Simplifications and the program implementation of the volume calcnlationa are presented in this chapter.

Conaider the volume calcnlations developed in Appendix III

$$V(STPD) = \int \dot{V} \cdot K(P_b, F_{H_2O}, T) \cdot dt$$

where V(STPD) is the STPD (Standard Temperature and Presanre; Dry) volume in liters, $\dot{V}$ is the flow in liters per second. The unitless factor $K(P_b, F_{H_2O}, T)$ is nsed to convert the ITP (Instantaneons Temperatnre and Pressure) flow to STPD conditions. This factor is defined as:

$$K(P_b, F_{H_2O}, T) = [1 - F_{H_2O}] \cdot \frac{273.15}{273.15 + T_i} \cdot \frac{P_b}{760}$$

where

P_b - the ambient barometric pressaure (torr)

F_{H_2O} - the fractional content of water vapor in the flow gas

T - the temperatnre of the gas (^oC).

The volume calcnlation in terms of the differential pressure is eqnal to

$$V(STPD) = \int \frac{k\pi r^4}{81\mu} \cdot \Delta P \cdot K(P_b, F_{H_2O}, T) \cdot dt$$

where

ΔP - differential pressnre obtained from the PTM (torr)

μ - viscosity of the gas (μP)

1 - effective length of the PTM (cm)

r - effective radiua of the PTM (cm)

k - 7.5062 (10^{-6}) ((dyne/cm^2)/torr)

The viscosity, length, radins, differential pressnre, and the ITP to STPD conversion factor vary throughout the respiratory cycle. From the work of Turney et. al. [19] the radins and length of a no. 1 Fleisch PTM vary less than 0.1% over the respiratory temperatnre range, 20^{o}C - 40^{o}C. Assuming this is also the case with the larger no. 2 Fleisch PTM nsed in this stndy, these variables can be considered constant and pulled outside the integral to obtain

$$V(STPD) = \frac{k\pi r^4}{81} \int \frac{\Delta P}{\mu} \cdot K(P_b, F_{H_2O}, T) \cdot dt \qquad 4.1$$

Wilke's eqnation can be used to calcnlate the overall viscosity of mixtnres of these gasea [8,9,20]. The general form of Wilke's eqnation is given by:

$$\mu_g = \sum_{i=1}^{4} \frac{Fw_i \cdot \mu_i(T)}{\sum_{j=1}^{4} Fw_j \cdot \beta_{ij}} \qquad 4.2$$

where

μ_g - the overall gas viscosity (μP)

μ_i - the viscosity of gas i (μP) at the temperatnre T

Fw_i - the fractional concentration of gas i in the wet gas

M_i - the molecular weight of gas i

β_{ij} - a dimensionlesa constant developed by Wilke [20] as

$$\beta_{ij} = \frac{[1 + (\mu_i/\mu_j)^{(1/2)}(M_j/M_i)^{(1/4)}]2}{[8\cdot(1 + M_i/M_j)]^{(1/2)}} \qquad 4.3$$

Over the respiratory temperature range the individual gas viscosities of O_2, CO_2, N_2, and H_2O are linearly related to gas temperature. These linear approximations are shown in Table 4.1

Table 4.1 Linear approximation of pure gas viscosities between 20°C - 40°C

$$\mu_{O_2} = 191.0 + 0.616\ T \quad [10]$$
$$\mu_{CO_2} = 137.6 + 0.450\ T \quad [10]$$
$$\mu_{N_2} = 166.6 + 0.454\ T \quad [10]$$
$$\mu_{H_2O} = 132.6 + 0.570\ T \quad [19]$$

where

μ_x - the viscosity of gas x (μP)

T - the gas temperature (°C)

These viscosity calculations are rather complex and, considering they are executed for each sample require a great deal of computer time. Wilke's Equations, 4.2 and 4.3, are the most complex and time consuming of these calculations. The iterative calculation of the denominator term in Equation 4.2 would require the largest amount of computer time. If this equation could be reduced to the linear combination of the individual viscosities a large amount of computation time would be saved. Turney et. al. [19] assumed mixtures of the four gases to have an overall viscosity equal to the linear combination of the viscosities of the components multiplied by their respective concentrations. However, this does not account for the complex interaction between the various gas components. This interaction is quantitated with the β_{ij} term. In order to account for this interaction and maintain the simplicity of the linear combination, Equation 4.4 is considered

$$\mu_g = \frac{Fw_{O_2}\mu_{O_2}}{\lambda_{O_2}} + \frac{Fw_{CO_2}\mu_{CO_2}}{\lambda_{CO_2}} + \frac{Fw_{N_2}\mu_{N_2}}{\lambda_{N_2}} + \frac{Fw_{H_2O}\mu_{H_2O}}{\lambda_{H_2O}} \qquad 4.4$$

where the λs are constants which account for the gas interactions. To determine if this is a valid approximation of Equation 4.2 for the normal respiratory concentrations the amount of variation of the denominator terms are investigated.

Shown in Figure 4.1 are plots of the four denominator terms of Wilke's equation plotted against the flow of a single respiratory cycle. This cycle has respiratory signals similar to those of Figure 2.1.2. The curve is

traced as the gas composition and gas temperatnre varies throughout the respiratory cycle. The average valne and standard deviation of these denominator terms over several normal respiratory cycles are listed in Table 4.2.

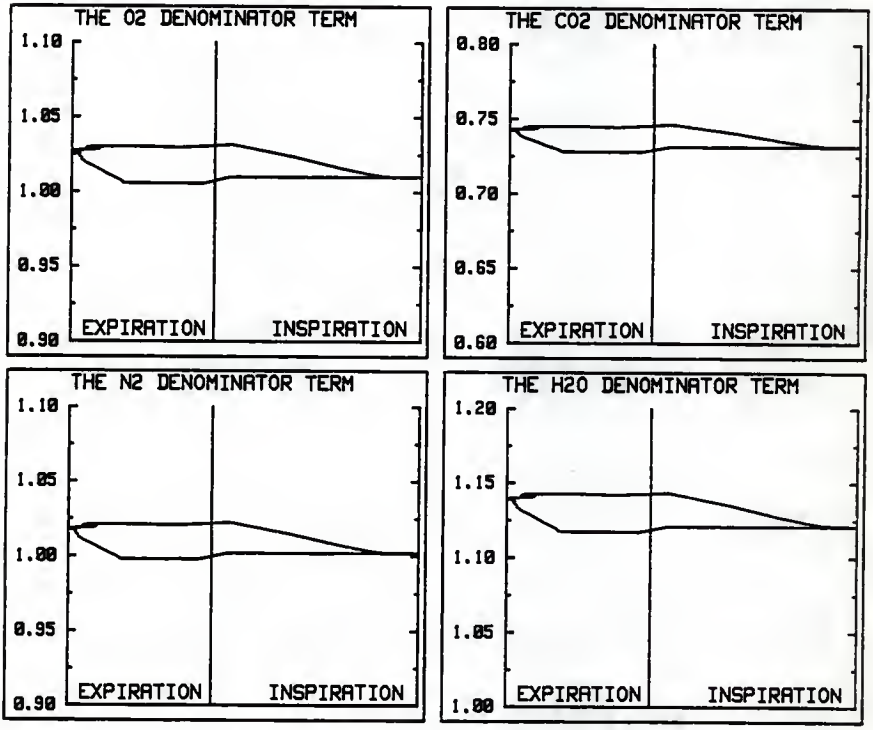


Figure 4.1 The denominator terms of Wilke's equation versus respiratory flow.

Table 4.2 Average values of the denominator terms of Wilke's eqnation

Gas	Average ± 1 Standard Deviation
Oxygen	1.0157 ± 0.0089
Carbon Dioxide	0.7353 ± 0.0060
Nitrogen	1.0074 ± 0.0084
Water Vapor	1.1274 ± 0.0094

From Fignre 4.1 and the standard deviation (less than 1%) it is apparent that these denominator terms are nearly constant. Therefore, a near linear relationship exists making the approximation of Equation 4.4 valid.

The coustant λ valnes are equated to the averages of the deuominator values. By iucorporatiug these coustauts iuto the viscosity calculatious of Table 4.1 the number of calculatioua may be reduced eveu fnrther. These modified viscosity eqnatioua are shown in Table 4.3.

Table 4.3
Modified linear approximatious for pure gas viscosities (20°C-40°C)

$$\mu'_{O_2} = 188.0 + 0.606\ T$$
$$\mu'_{CO_2} = 187.1 + 0.612\ T$$
$$\mu'_{N_2} = 165.4 + 0.451\ T$$
$$\mu'_{H_2O} = 117.6 + 0.506\ T$$

μ'_x - the modified viscoaity of gas x (μP)

The overall viscosity cau be found as the liuear combination of the iudividual modified viscosities mnltiplied by their respective fractioual coucentrations.

$$\mu_g = Fw_{O_2}\mu'_{O_2} + Fw_{CO_2}\mu'_{CO_2} + Fw_{N_2}\mu'_{N_2} + Fw_{H_2O}\mu'_{H_2O} \qquad 4.5$$

Using this linear approximation to calculate gas viscosity over several normal reapiratory cycles results in au average error of 0.5% from the viscosity calculated with Wilke's equation.

A relative viscosity is defined as the viscosity of the gas relative to room air under STPD couditious. The value of relative viscosity is approximately eqnal to oue. This allows the calibratiou factors calculated nsing viscosity and temperatnre correctious to be nearly equal to thoae calculated assumiug the flow ia directly proportional to the differential pressnre.

$$\mu_r = \frac{\mu_g}{\mu_{STPD}} \qquad 4.6$$

Applying these last few developments to Eqnatiou 4.1 resnlts in the volume calculation of

$$V(STPD) = \frac{k\pi r^4}{8l\mu_{STPD}} \int \frac{\Delta P}{\mu_r} \cdot K(P_b, F_{H_2O}, T) \cdot dt \qquad 4.7$$

The trapezoidal uumerical integratiou techuiqne is nsed to perform all integration operations.

Chapter V. SYSTEM CALIBRATION

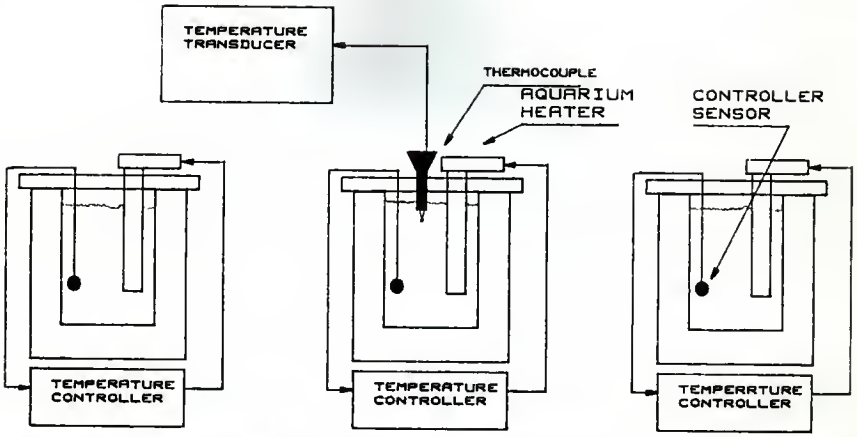
Calibration factors must be determined in order to determine the respiratory signals in terms of their correct units. These factors relate the unitless binary sampled data, BCD data, to the physical phenomena being measured. Calibration of the gas mass spectrometer as presented by Creel [4] and Riblett [13] has not been altered and is not discnssed here. The temperature transducer calibration procedure and an alternate scheme for calibration of the pneumotachograph (PTG) are presented. The new PTG calibration scheme nses viscosity and temperature corrections.

5.1 Temperature Calibration

The thermocouple based temperatnre transducer produces an ontput voltage related to the temperatnre at the thermocouple tip. Material presented in Appendix IV indicates that this temperature-voltage relationship is nonlinear. For the chromel-constantan thermoconple the temperature-voltage and, since the BCD sample is linearly related to the transducer voltage, the temperature-sampled voltage relationships may be modeled by second order polynomials over the respiratory temperatnre range (20°C - 40°C). To determine the relationship between the temperature and the sampled voltage, three temperature-BCD data points are taken. From these three points the second order relationship is generated.

The temperatnre calibration setup is shown in Fignre 5.1.1. Three water baths are nsed to provide three inpnt temperatnres to the temperatnre transdncer. Each water bath is heated with a separate aqnarium heater. Each heater ia controlled with a temperature controller in order to maintain the bath temperature at a constant level. The temperatnres of the three water baths, 22°C, 30°C, and 39°C, are referred to as T_{low}, T_{mid}, and T_{high} respectively. The normal respiratory temperature range is spanned by these temperatnres.

The tip of the thermocouple is placed into a water bath and the bath temperatnre is measnred with a calibrated thermometer. The thermometer reading is entered into the computer. The computer program contains the thermometer's calibration factors so that the true temperatnre may be determined. The computer collects and averages 500 samples of the transdncer ontpnt voltage. This step is performed for all three baths.



Fignre 5.1.1 The equipment setup for the calibration of the temperatnre transducer.

The three calibration points $[T_{low}, B_{T_1}]$, $[T_{mid}, B_{T_2}]$, and $[T_{high}, B_{T_3}]$ are nsed to generate the second order polynomial:

$$T = T_c + T_b \cdot B_T + T_a \cdot (B_T)^2 \qquad 5.1.1$$

From the three calibration points, the equation:

$$\begin{bmatrix} T_{low} \\ T_{mid} \\ T_{high} \end{bmatrix} = \begin{bmatrix} 1 & B_{T_1} & (B_{T_1})^2 \\ 1 & B_{T_2} & (B_{T_2})^2 \\ 1 & B_{T_3} & (B_{T_3})^2 \end{bmatrix} \begin{bmatrix} T_c \\ T_b \\ T_a \end{bmatrix} \qquad 5.1.2$$

is generated, where

T – the temperatnre corresponding to the binary valne B_T (°C)

B_T – the binary value corresponding to the TC transdncer outpnt

T_c – the zero order calibration coefficient (°C)

T_b – the first order calibration coefficient (°C)

T_a – the second order calibration coefficient (°C)

B_{T_1} – the binary valne corresponding to the TC transducer ontpnt for a temperatnre of T_{low}

B_{T_2} – the binary valne corresponding to the TC transdncer outpnt for a temperature of T_{mid}

B_{T_3} – the binary valne corresponding to the TC transdncer ontpnt for a temperature of T_{high}

Equation 5.1.2 is then solved for the calibration coefficients T_a, T_b, and T_c. These temperature calibration coefficients are used during flow cali-

bration snd are stored for later nse by the analysis rontine.

5.2 Flow Calibration

The flow calibration of Creel [4] and Riblett [13] assumes thst calibration and exercise data collection occnr under the ssme temperatnre and water vapor conditions. It is hoped that by heating the PTM the temperatnre is nearly the same dnring calibration and data collection. However, it has been observed that with the #2 Fleisch PTM the temperatnre signal has an average temperatnre dnring calibration of 23°C snd deviates abont 2°C while dnring data collection it is approximately 10°C warmer. In addition, both inspiratory and expiratory water vspor content dnring cslibration is that of room air which is nsnally less thsn 1% and seldom greater than 2%. Dnring data collection the expiratory water vspor content can be as high as 6%. Also, Creel and Riblett neglected chsnges in the gas viscosity in order to assume the PTG differential pressnre signal to be directly proportional to the flow. While this assumption is valid for constsnt gas composition, it is not valid when the gas composition varies. The amount of variance of the gas viscosity dnring respiration can be seen in Fignre 5.2.1. The relative viscosity is plotted versns F_{CO_2} in the top fignre and versns flow in the bottom fignre. The constant conditions and constant viscosity assumptions nsed by Creel and Riblett lesd to the possibility of errors in calcnlating volumes.

As discnased in Chapter IV, the gas viscosity can be calcnlated and nsed to convert the ΔP signal to a flow signal. Therefore, the PTG signal which is directly proportionsl to the ΔP signal can be converted to a signsl that is directly proportional to the flow. The temperatnre signal may be nsed to convert this flow to any given temperatnre, pressnre, and wster vapor condition. This allows for the integration of a flow which is under constant conditions. For convenience the flow is converted to and integrsted under STPD conditions. A scheme is presented which nses the flow gas temperstnre and the gas viscosity to calibrate the PTG.

The sampled binary valne, B_p, of the PTG ontpnt is a linesr function of the differential pressnre, ΔP.

$$B_p = m \cdot \Delta P + b \qquad 5.2.1$$

Ideally, two valnes of differential pressnre, ΔP, snd the corresponding

sampled values, the B_p values, can be used to solve for the two unknowns, the slope m and the y intercept b. A zero differential pressure is easily generated but it is difficult to generate a calibrated differential pressure across the screen of the PTM.

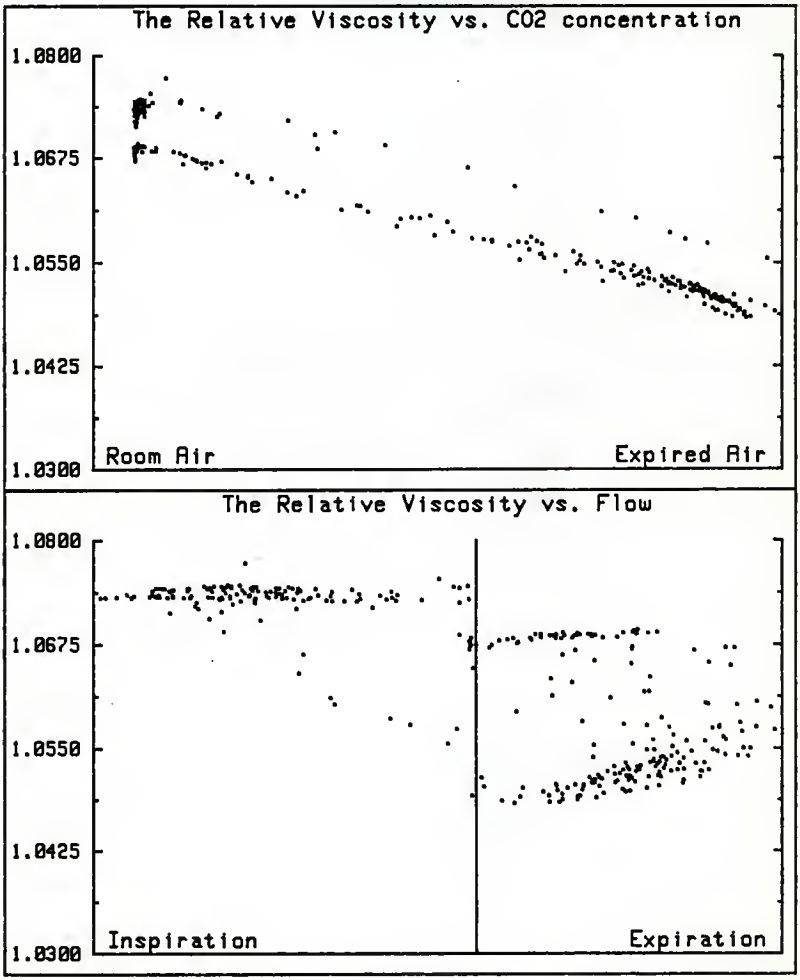


Figure 5.2.1 The variation of the relative viscosity throughout the respirator cycle.

The zero order term, b, is equal to the binary sample corresponding to zero differential pressure. Zero differential pressure is easily generated

by isolating the PTM from all air flow. This is accompliahad by simply placing tha PTM in a box. With tha PTM isolatad, 500 binary samples of tha PTG output ara collactad and avaragad. This avaraga is then equated to tha zero order tarm.

Tha scheme to complata tha relationship of tha sampled data to tha differential pressura involvas five steps. Thesa steps ara

1. Cycla a known volume throngh the PTM. A Harvard respirator is usad to cycla a volume through the PTM.

2. Convart tha PTG signal to a signal that is linearly ralatad to tha STPD flow uaing tha gas viacosity and tha convarsion to STPD conditions.

3. Integrata tha rasnltant signal over aach inspiratory and each expiratory cycla of the respirator to obtain a voluma that is diractly proportional to tha STPD respirator volume.

4. Averaga tha individual inspiratory volumes and axpiratory volumes.

5. Computa tha proportionality constant by dividing the known raspirator voluma by the avaraga valuas.

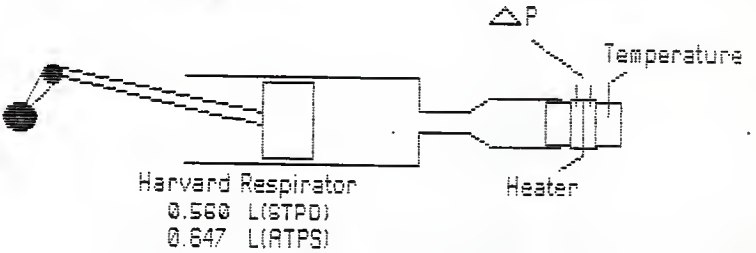


Figure 5.2.2 Equipmant satup for tha calibration of the PTG.

Tha calibration satnp is shown in Fignra 5.2.2. This, with tha excaption of tha tamperatnra transducar, is tha aama satnp nsed by Creel [4] and by Riblatt [13]. Separata inspiratory and axpiratory calibration factors for inspiratory flows and axpiratory flows ara calcnlatad. The usa of saparate inapiratory and expiratory flow calibration factors is snggested by Turney et al. [19] to account for tha asymmatry of tha PTM. The datails of ganerating tha calibration factors ara prasantad hara. Craal's schame [4], also nsed by Riblett [13] is maintainad with the prasant systam.

Consider the volume calculations developed in Chapter IV and Appendix III.

$$V(STPD) = \int \dot{V} \cdot K(P_b, F_{H_2O}, T) \cdot dt$$

which in terms of the differential pressnre is equal to

$$V(STPD) = \frac{k\pi r^4}{8l\mu_{STPD}} \int \frac{\Delta P}{\mu_r} \cdot K(P_b, F_{H_2O}, T) \cdot dt$$

Rearranging Equation 5.2.1 so that the sampled data may be used to compute this volume yields

$$V(STPD) = \frac{k\pi r^4}{8lm\mu_{STPD}} \int \frac{(B_{\Delta P} - b)}{\mu_r} \cdot K(P_b, F_{H_2O}, T) \cdot dt$$

where b is the zero order term which corresponds to zero flow (zero ΔP). The constant term mnltiplying the integral is combined into a single factor known as the flow calibration factor. There are two calibration factors, one for inspiratory flow, $\dot{V}_{I_{cal}}$, and one for expiratory flow, $\dot{V}_{E_{cal}}$ (Note: these calibration values are related to bnt not eqnal to the first order coefficient, m, of Eqnation 5.2.1). The inspiratory and expiratory volumes are calcnlated by the integrals

$$V_I(STPD) = \dot{V}_{I_{cal}} \int_I \frac{(B_{\Delta P} - b)}{\mu_r} \cdot K(P_b, F_{H_2O}, T) \cdot dt \qquad 5.2.3$$

$$V_E(STPD) = \dot{V}_{E_{cal}} \int_E \frac{(B_{\Delta P} - b)}{\mu_r} \cdot K(P_b, F_{H_2O}, T) \cdot dt \qquad 5.2.4$$

The dry gas concentrations of O_2, CO_2, and N_2 needed in the calculation of relative viscosity are assumed to be that of room air (F_{O_2}= 20.93%, F_{CO_2}= 0.04%, and F_{N_2}= 78.06%). The temperature is continnally monitored dnring calibration in order to calculate the viscosity and to convert the compnted flow to STPD conditions. The compntations require the relative humidity, room temperatnre, and the barometric pressnre.

The respirator cycles at 60 cycles per minnte in order to create flows close to those of a snbject. The determination of the ATPS respirator volume is ontlined by Creel [4]. The respirator is connected to a spirometer. As the respirator slowly cycles (5 cycles/min), the spirometer displacement is recorded on chart paper. From the spirometer displacement the stroke volume may be computed. This volume is under ATPS conditions since the system is

eqnilibrated to ambient temperatnre and the gas is over water; therefore, is saturated with water vapor. By measuring the temperature of the water in the spirometer and barometric pressnre the respirator volume can be converted to STPD conditions.

The respirator flow is sampled for 80 seconds generating approximately 80 respirator cycles. The valne of the integrals (neglecting the calibration constants) in Equations 5.2.3 and 5.2.4 are calcnlated for each cycle and then averaged. The known STPD volume of the respirator is then divided by these averages to obtain the calibration factors:

$$\dot{V}_{I_{cal}} = \frac{\text{STPD respirator volume}}{\frac{1}{N}\sum_{i=1}^{N}\int_{I_i}\frac{(B_{\Delta P}-b)}{\mu_r}\cdot K(P_b,F_{H_2O},T)\cdot dt} \qquad 5.2.6(a)$$

$$\dot{V}_{E_{cal}} = \frac{\text{STPD respirator volume}}{\frac{1}{N}\sum_{i=1}^{N}\int_{E_i}\frac{(B_{\Delta P}-b)}{\mu_r}\cdot K(P_b,F_{H_2O},T)\cdot dt} \qquad 5.2.6(b)$$

With the method nsed by Creel and Riblett the volume of the respirator is determined under ATPS conditions and the integrand does not contain the viscosity term nor the ITP to STPD conversion factor. The integrals of Eqnations 5.2.3 and 5.2.4 simplify to

$$V_I = \dot{V}_{I_{cal}}\int_I (B_{\Delta P}-b)\cdot dt \qquad 5.2.7$$

$$V_E = \dot{V}_{E_{cal}}\int_E (B_{\Delta P}-b)\cdot dt \qquad 5.2.8$$

The calibration factors are then calcnlated by dividing the ATPS volume of the respirator by the average of the respirator cycles.

$$\dot{V}_{I_{cal}} = \frac{\text{ATPS respirator volume}}{\frac{1}{N}\sum_{i=1}^{N}\int_{I_i}(B_{\Delta P}-b)\cdot dt} \qquad 5.2.9(a)$$

$$\dot{V}_{E_{cal}} = \frac{\text{ATPS respirator volume}}{\frac{1}{N}\sum_{i=1}^{N}\int_{E_i}(B_{\Delta P}-b)\cdot dt} \qquad 5.2.9(b)$$

The nse of the respirator volume under ATPS conditions by Creel [4] and Riblett [13] has the potential of leading to errors in volume calcnlations.

These errors arise because the PTG is calibrated under conditions other than ATPS. The goal of heating the PTM head is to bring its temperature to near body temperature. Assuming this is accomplished, the non-corrected calibration factors are computed by comparing the respirator volume under ATPS conditions ($T = 21^{o}C$, $F_{H_2O} = 2.6\% = 18.6/730$) to a volume computed at a temperature of $36^{o}C$ and a water vapor concentration of less than one (room $F_{H_2O} = 30\%(18.6/730) = .7\%$). This leads to erroneous inspiratory and expiratory flow calibration factors. Also, during data collection the expiratory water vapor content is near 6% as compared to less than 1% during calibration. With this method the calibration volume is calculated under one temperature and pressure condition, the PTG calibrated under different conditions, and the expiratory volumes calculated under still another condition. By computing all volumes under STPD conditions these potentials for errors are reduced.

Chapter VI. SYSTEM SOFTWARE

The CBRMS aystem software developed by Creel [4] and modified by Riblett [13] for execntion on the HP9826 system has been enhanced with volume calculationa which nae the temperatnre signal and the water vapor and viscosity calculations described in Chapter IV. Also, display of the breath-by-breath resnlts using two breath combination (averaging) techniqnea has been integrated into the CBRMS. The plotting of the respiratory data, the printing of the breath-by-breath respiratory volumes, and the printing of the mean valnes have been maintained.

The external design of the software, as described by Riblett, has not been changed (with the exception of a few additional nser inpnts). Therefore, file compatability and compnter peripheral requirementa remain the same. The calibration and reapiratory data are collected nsing programs written in Pascal. The calibration and data collection routines store the data in temporary filea naing the same format (ASCII format). The ASCII format is necessary since it ia the only file format which is compatible with both Basic and Paacal. The data are paased to the analysis routine in the same manner; via the ASCII files. However, the temporary ASCII data are optionally stored on the HP9895A 8-inch flexible disk, platter 1 of the HP9134A hard disk, or platter 2 of the hard disk (the medium used by Riblett). The ASCII data are converted to the numeric format of the Basic operating system and stored for a permanent record of the reapiratory data nsing rontines written in Basic. The data are analyzed using an enhanced analysis rontine written in Basic.

While the external design of the software has not been changed from Riblett's version [13] the internal strncture has. In order to take full advantage of the system languages, Hewlett-Packards's Basic and Pascal much of the code was converted to a atrnctnred format. Strnctnred code consists of only three types of inatrnctions; sequential, conditional, and iterative [15]. Thia reorganization reaulted in code that ia more compact, readable, and understandable.

The operation of the system software ia outlined in Fignre 6.1 and detailed operator instructiona are given in Appendix V. Flowcharts for the variona seotions are presented in the latter portion of this Chapter and the actual program liatings appear in the Appendix VI and Appendix VII.

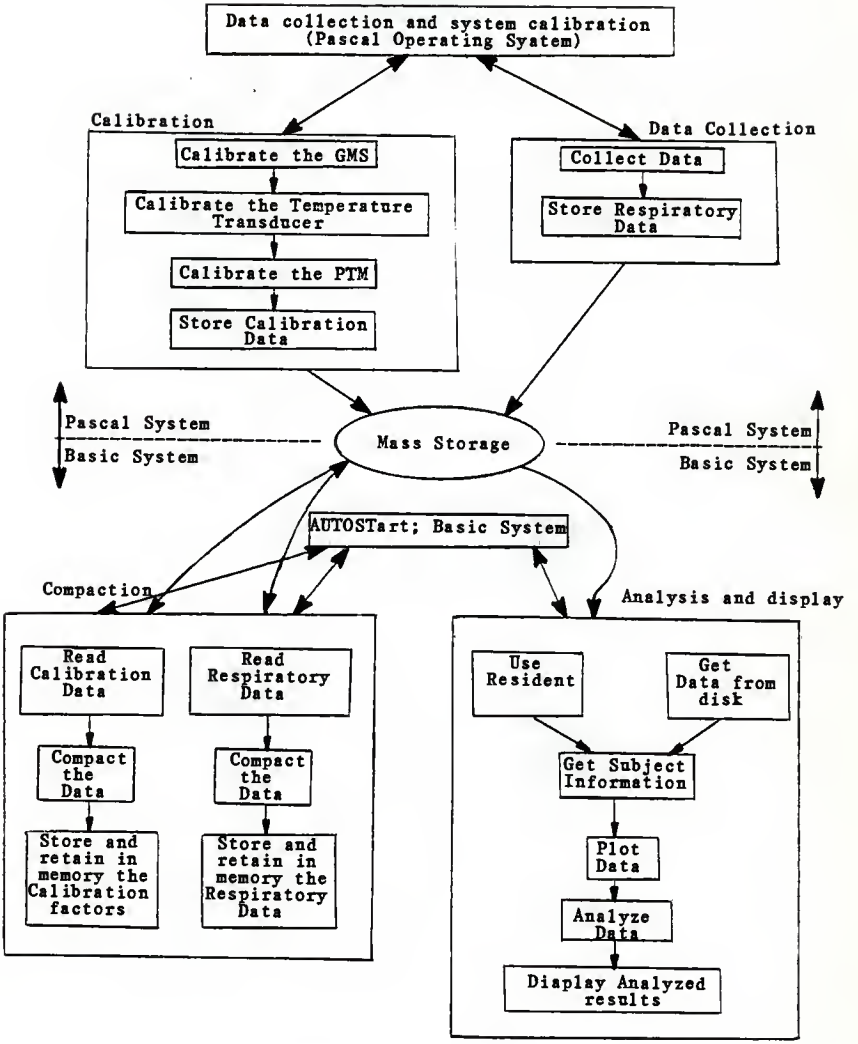


Figure 6.1 Organization of the CBRMS software

The calibration and respiratory data are collected with software written in Pascal. Pascal is reqnired becanse of the need of assembly code to control the DAM in the collection of the data [13]. Calcnlation of the calibration factors for the system is aocomplished with the Pascal program CAP2.CODE. The respiratory data are collected with the Pascal program DAP2.CODE. Aa mentioned, both Pascal programs store their data on disk in ASCII format. The Basic programs DAPCRUNCH and CAPCRUNCH convert this ASCII data to the more compact numeric format of the Basic aystem. The compacted data are stored on diak for a permanent record. These data are then naed by the Basic program ANALYSIS to compute and display the varions reapiratory parameters.

6.1 CAP2.CODE: A Pascal Routine for System Calibration

CAP2.CODE (CAlibration Program, veraion 2) determinea calibration factors which are nsed to relate the sampled transdncer outpnts to the respiratory aignals. Computation of the GMS calibration factors was developed by Creel [4] and the compntation of the temperature transdncer and flow transducer calibration factors is diacnssed in Chapter V. The flowchart of the calibration program is ahown in Figure 6.1.1. The operator's instrnctions for CAP2 are given in Appendix V, Section A5.4.1, and the Paacal source code is documented in Appendix VI, Section A6.1.

When the program begins execntion the sampling freqnency is entered. This sampling frequency, generally 50 Hz, is nsed when the calibration points are being sampled. The GMS is the firat device to be calibrated. The calibration of the GMS may be skipped, the same is true for the flow and temperatnre tranadncers. (Skipping any one of the calibration sections uanally occnrs only if a test is being conducted on a particnlar transdncer or particnlar section of the calibration program.) The temperatnre transducer is calibrated next.

The PTG is the final transducer to be calibrated. If temperature and viscosity corrections are to be nsed the ambient temperature, humidity, and pressnre are entered. A flow chart of the flow calibration scheme is presented in Figure 6.1.2 and is discnssed shortly. Use of the temperatnre and viscosity calcnlations with the flow calibration may be optionally omitted.

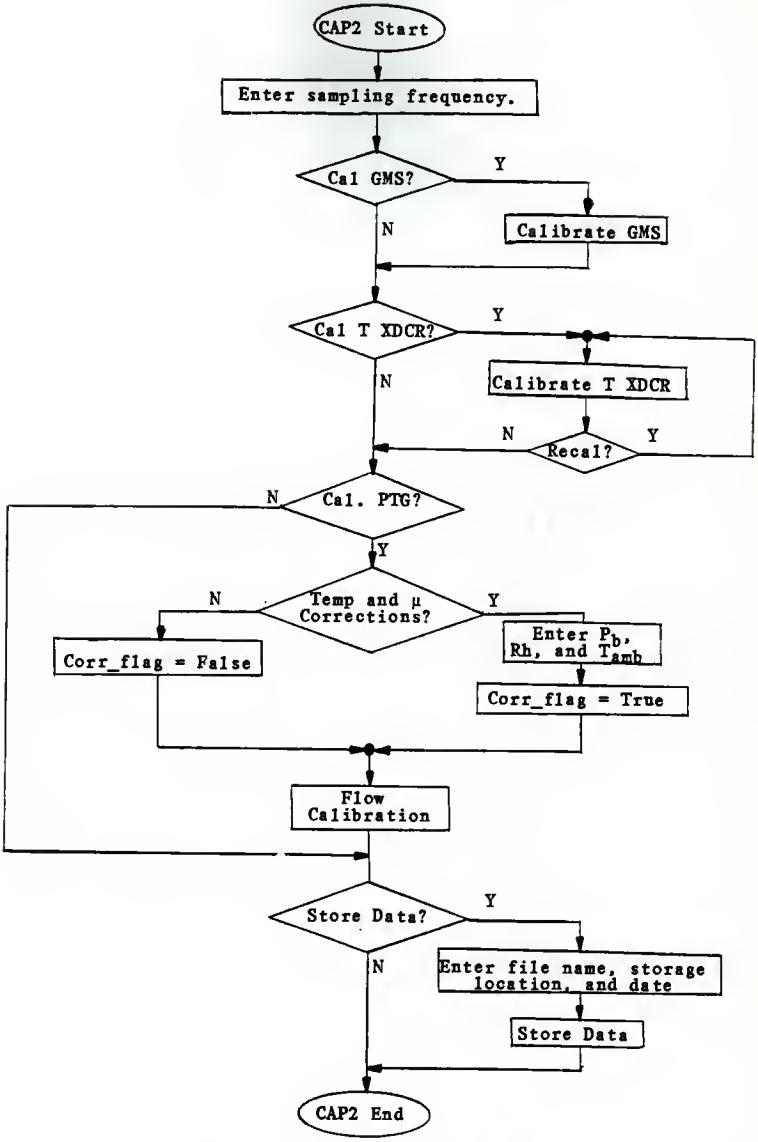


Figure 6.1.1 Calibration program, CAP2

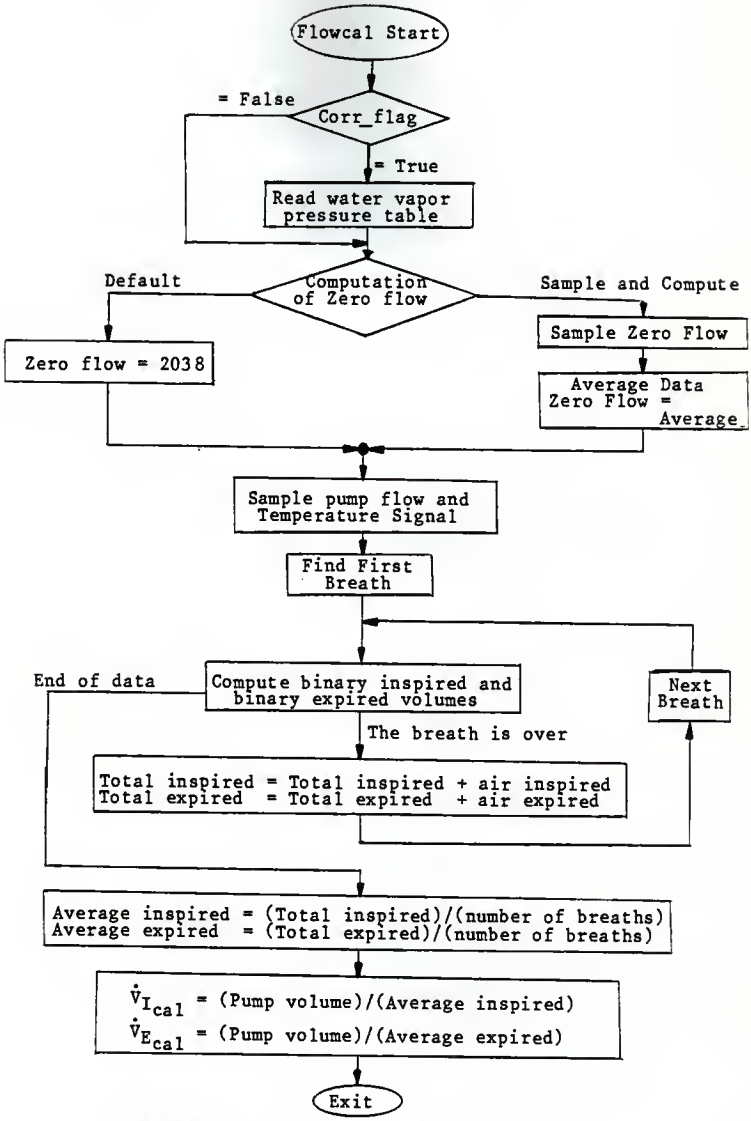


Figure 6.1.2 Flow calibration procedure

Finally the calibration data are stored along with the date. Program execntion stops, and the computer operation retnrns to the Main Command Level of the Pascal operating system.

The flow chart of the flow calibration is ontlined in Figure 6.1.2. First the binary value corresponding to zero flow is found. A default valne of 2038 can be nsed for the binary valne corresponding to zero flow (nearly all calcnlated zero flow values are eqnal to 2038).

The pump flow is then sampled for 4000 points. The binary inspiratory and expiratory volumes of the pump are computed (the computation of the binary volumes is presented in Section 6.5) for each cycle of the respiratory signal. Dnring calibration two sets of the binary volumes are compnted; one set nses temperatnre and viscosity corrections and the other uses no corrections. The binary volumes of each respiratory cycle are averaged. The inspiratory calibration factors are found by dividing the pump volume by the average inspiratory binary volumes. The same steps are performed with the corresponding expiratory volumes to determine the expiratory calibration factors. The routine then retnrns to the main calibration routine CAP2.CODE.

Below is a list of the major revisions to the original calibration program, CAP.CODE, described by Riblett [13].

1. Calibration of the temperatnre transdncer can be repeated upon the nser's reqnest.

2. The PTG can optionally be calibrated using temperatnre and viscosity corrections in addition to the method nsed by Riblett.

3. The calibration data may be stored on one of three disk drives; platters 1 or 2 of the hard disk or the 8-inch floppy disk.

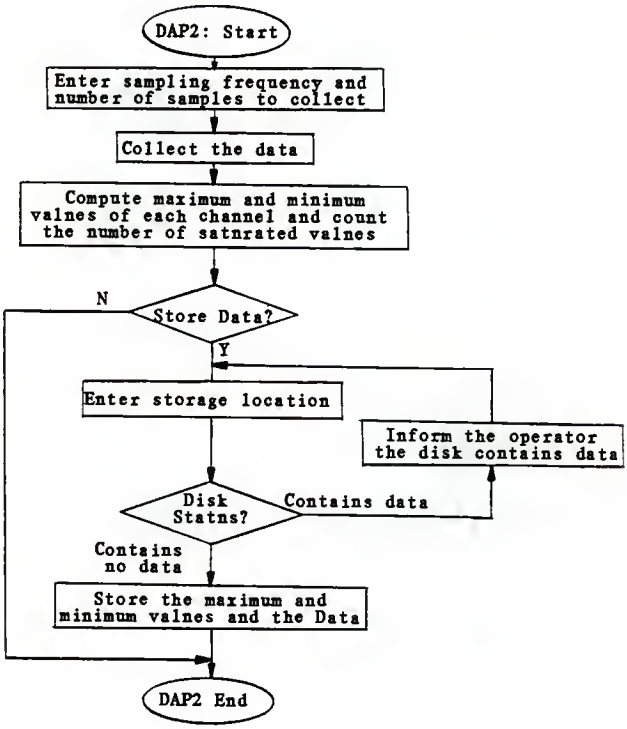
4. When data are being collected the number of samples and the length of time required to collect the samples are displayed.

5. The operator is prompted with an andible beep for all input data, just as occnrs with the Basic routines.

6.2 DAP2.CODE: A Pascal Routine to Collect Respiratory Data

DAP2.CODE (Data Acqnisition Program, Version 2) collects the respiratory data from the subject and stores the data on disk using ASCII format. Version one of this program was written by Riblett [13] and only minor

modifications were made for version 2. The flowchart for DAP2 is shown in Fignre 6.2.1. The operator's instructions for DAP2 are given in Appendix V, Section A5.4.2 and the Pascal sonrce code is documented in Appendix VI, Section A6.2.



Fignre 6.2.1 Data acquisition program, DAP2

The modifications to version 1 of the data acqnisition program are listed below.

1. The respiratory data may be stored on one of three disk drives; platters 1 or 2 of the hard disk or the 8-inch floppy disk.

2. The ASCII data are protected from accidentally being overwritten.

3. When the data are being collected the number of samples and the length of time required to collect the samples are displayed.

3. The number of data points equal to the positive saturation value of the A/D, 4095, and the number equal to the negative saturation value of the A/D, 0, are counted and displayed for each channel.

4. The operator is prompted with an audible beep for all input data, just as occurs with the Basic routines.

6.3 AUTOST, CAPCRUNCH, and DAPCRUNCH: Auto-start and File Compaction Routines

The auto-start program, AUTOST, serves as a link between the various routines written in Basic. CAPCRUNCH and DAPCRUNCH are Basic programs which convert the data stored on disk in ASCII format by the Pascal programs CAP2.CODE and DAP2.CODE to the more compact numeric formats of the Hewlett-Packard Basic. This compaction greatly reducea the amount of storage space required for the data and the length of time to retrieve the data from disk. The operator's instructions for AUTOST, CAPCRUNCH, and DAPCRUNCH are given in Appendix V, Section A5.5, and the Basic programa are documented in Appendix VII, Sections A7.3, A7.4, and A7.5.

The flowchart of Figure 6.3.1 is a combination of three separate programs. The auto-start program, AUTOST, serves as a link between the three CBRMS programs; DAPCRUNCH, CAPCRUNCH, and ANALYSIS; by allowing them to be executed with the press of a key. Another Basic program, STOREHR (Store Heart Rate) can be loaded from AUTOST. AUTOST also allowa several other useful operations; CATaloging of disks, setting the default mass storage device, and loading other programs.

CAPCRUNCH, converts (crunches) the calibration factors computed and stored (in ASCII format) by the Pascal program CAP2.CODE from the ASCII format to the more compact numeric format of the Hewlett-Packard's 9826 Basic. DAPCRUNCH does the same for the respiratory data collected and stored by Pascal program DAP2.CODE. Below is a list of the revisions made to theae programs.

1. A common area of memory has been set up ao that the data read by the CAPCRUNCH and DAPCRUNCH programs remain in memory to be used by the ANALYSIS program.

2. The ASCII data can be read with the crnnching programs from either platter 1 or platter 2 of the hard disk or the 8-inch disk drive.

3. The file names of the respiratory data are entered at the beginning of the DAPCRUNCH program. This allows the names to be entered all at once and then the data from all four channels are read in and converted and finally stored without the need for further operator input.

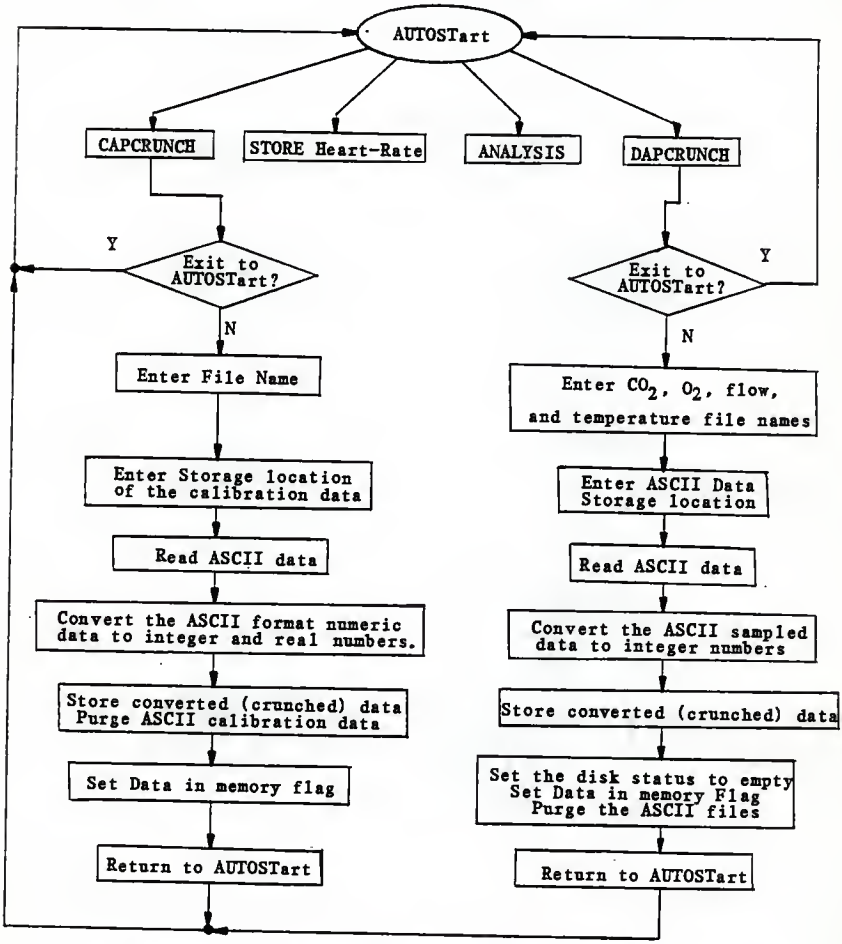


Figure 6.3.1 AUTOST, CAPCRUNCH, and DAPCRUNCH programs

6.4 ANALYSIS: A Basic Rontine to Calculate the Respiratory Parameters

ANALYSIS is the analysis rontine which performs the compntations neceasary to yield the reapiratory parameters described in Chapter IV. Theae parameters may be diaplayed in a variety of formats. The operator's instrnctions for ANALYSIS are given in Appendix V, Section A5.6, and the Basic program documentation is in Appendix VII, Section A7.6.

Fignre 6.4.1 shows the flowchart of ANALYSIS. The analysia program is called from the AUTOSTart program, as shown in the flowchart of the AUTOSTart program (Fignre 6.3.1). System operation can also return from the analysia program to the AUTOSTart program. The first section of the analysis program obtains the necessary subject information, the snbject'a respiratory data, and the calibration data. The reapiratory and calibration data may be reaident in memory dne to the execntion of the data compaction programs, CAPCRUNCH and DAPCRUNCH, or previons execntion of the analysia program. These data may be nsed by the analysis program or new data may be read from disk.

The operator selects the type of GMS time delay [13], breath-by-breath or fixed, and whether temperatnre and viscosity calcnlations are to be used in computing respiratory flow. The operator may plot the raw respiratory data. The window of data points for which the reapiratory parameters are to be calcnlated is selected and the calcnlation of the respiratory parameters for that window begins.

The GMS time delay for the breath being analyzed ia determined first. The time delay is either a fixed value or a compnted value. The algorithm for the compntation of the breath-by-breath time delay was presented by Riblett [13]. The binary volumes of air, oxygen, and carbon dioxide are compnted. The computation of the binary volumes is described in Section 6.5 (this is the same flowchart nsed by the calibrstion program; CAP2.CODE, Section 6.1; to compute the binary pump volumes). The flow calibration factors are used to convert the binary volumes to the the varions respiratory volumea having the units of litera. The respiratory volumes may be printed for each breath being analyzed; however, most information can be attained from the later display. These respiratory volumes and the dnration of the reapiratory cycle are uaed to compnte the other respiratory parameters; minnte values, respiratory quotient, and respiratory frequency. All respiratory parameters are retained for later display. After all data in the

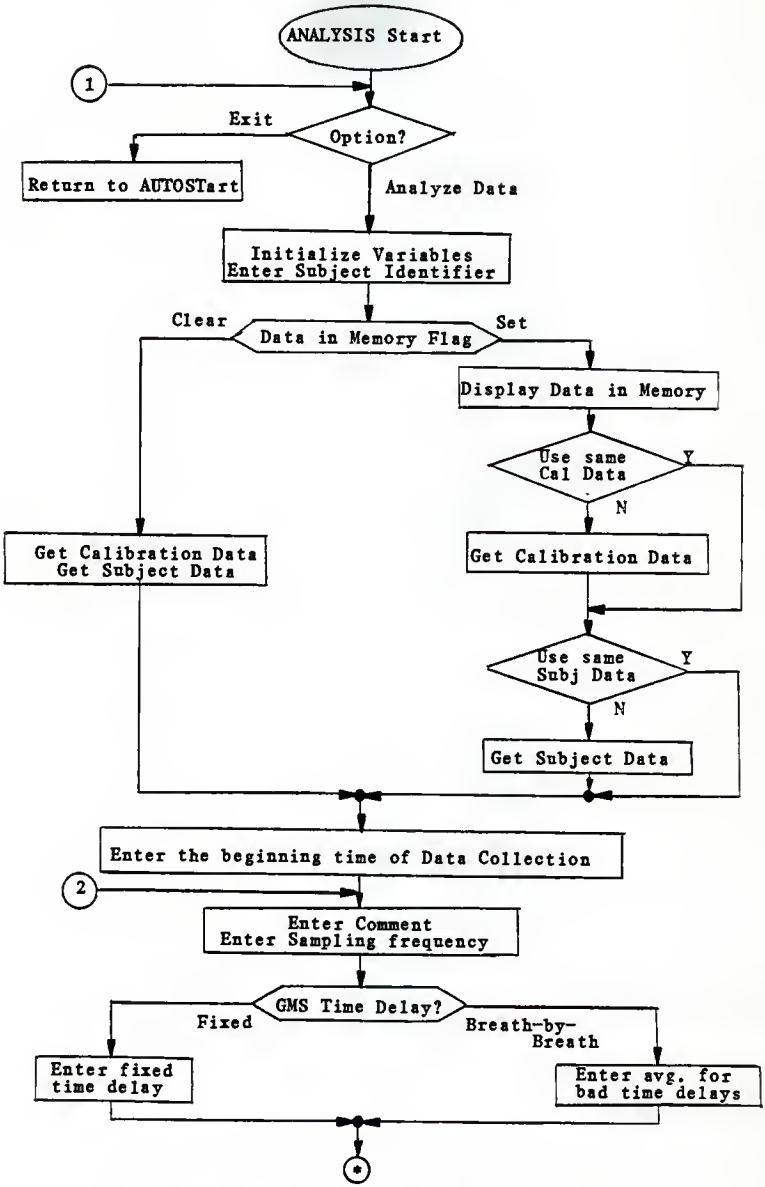


Figure 6.4.1 Data analyais program, ANALYSIS

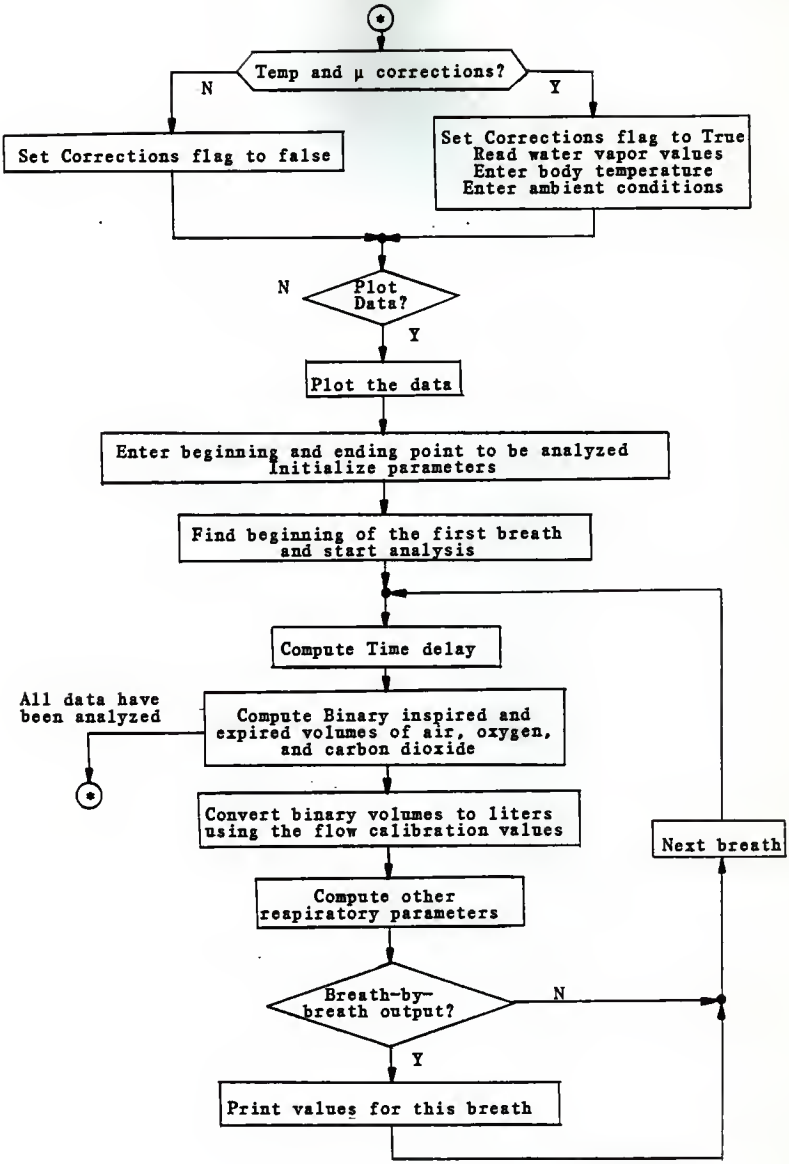
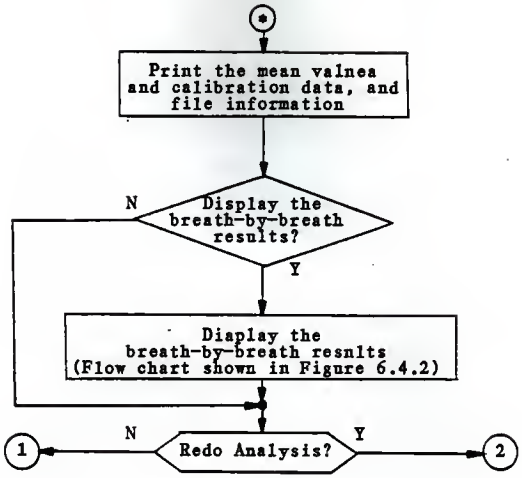


Figure 6.4.1 (continued)

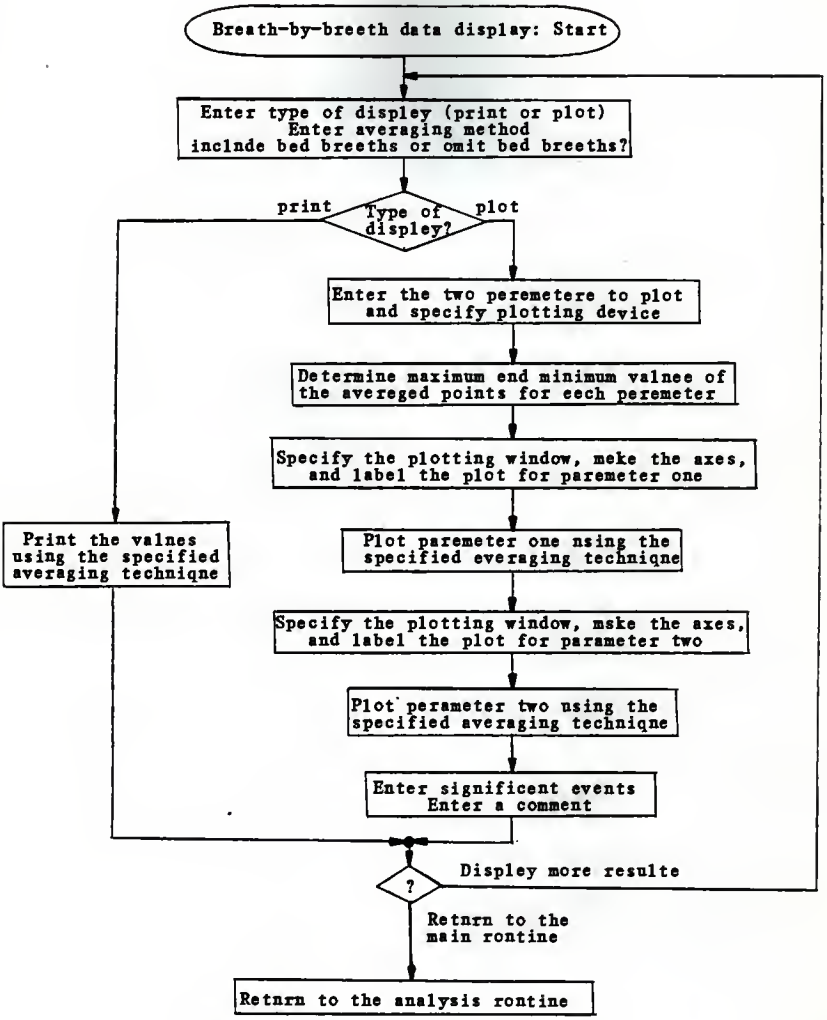


Fignre 6.4.1 (continued.)

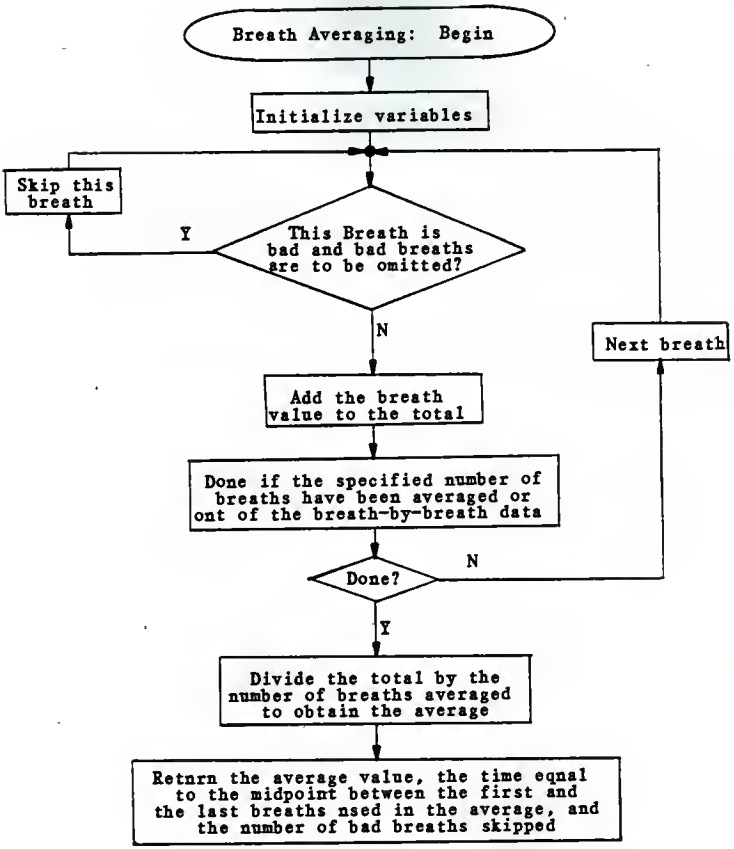
specified window have been analyzed the mean respiratory parameters for that window are computed and printed.

Following the printing of the mean valnes the computed breath-by-breath respiratory parameters may be diaplayed. The flowchart for the display of the breath-by-breath data is shown in Figure 6.4.2. Successive breath-by-breath valnes may be combined by one of two techniqnes; breath averaging, flowchart shown in Fignre 6.4.3, or window averaging, flowchart shown in Fignre 6.4.4. All of the respiratory parameters may be printed or any two parameters may be plotted. After the parameters have been displayed, control may be paaaed back to the main portion of the analysis program or additional parameters may be displayed.

The first of the breath combination techniques ia breath averaging, Figure 6.4.3. The specified number of breath-by-breath valnes are summed. A breath is not included in the sum if it is considered a bad breath and the operator specifies that the bad breaths are to be omitted. A bad breath is defined as a breath that has an inspiratory tidal volume or an expiratory tidal volume of less than or eqnal to 400mL. The summing continnes until the specified number of breaths have been averaged or no more breath-by-breath data are available. The average is computed. The routine then retnrna the average valne, the time valne that is midway between the first breath and the



Fignre 6.4.2 Display of the breeth-by-breeth data



Fignre 6.4.3 The breath averaging procedure

last breath averaged, and the number of bad breaths that are skipped.

The flowchart for the window averaging technique is shown in Figure 6.4.4. The goal of the window averaging technique is to look at a window in time and average all the breath values in that window. This is illustrated in Fignre 2.2.3 for two windows.

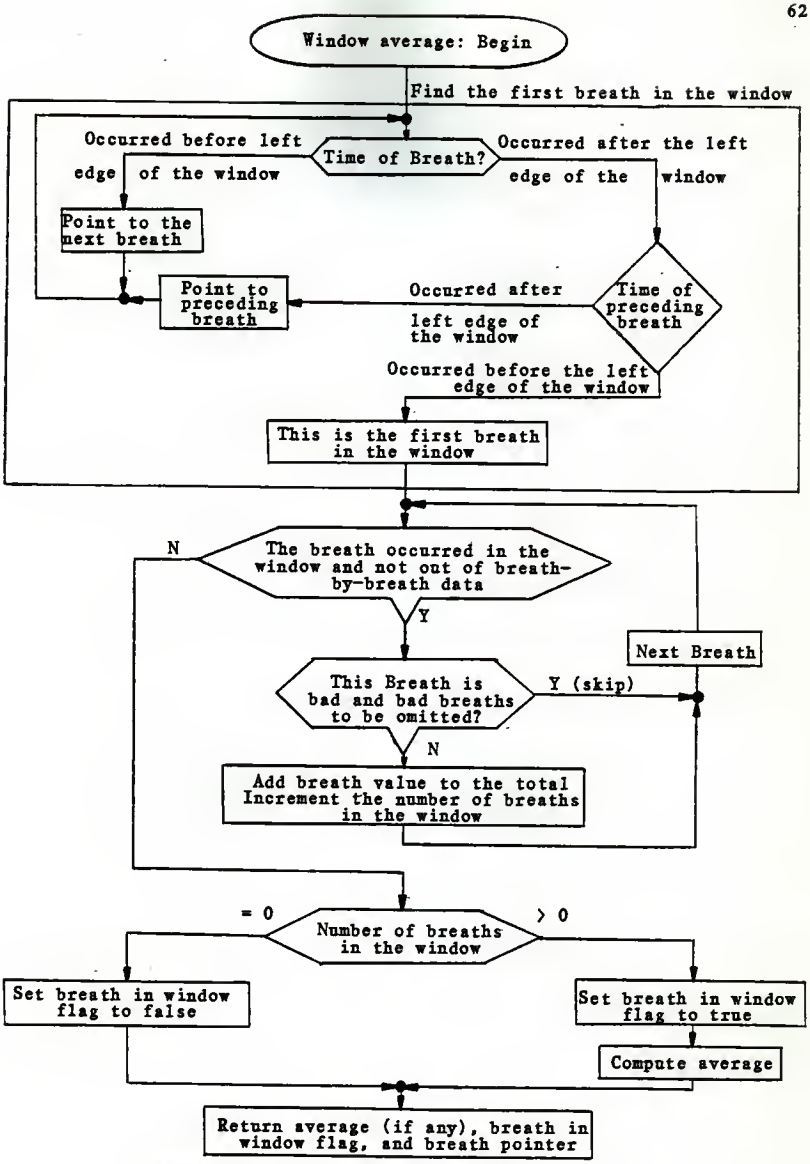


Figure 6.4.4 The window averaging procedure

The first thing the window average routine does is to find the first breath that occurred in the window. The time of a breath is checked. If the time of this breath precedes the left edge of the window then the next breath is tested. If the time of the breath occnrs after or at the same time as the left edge of the window then the preceding breath is checked. If the time of the preceding breath precedes the left edge of the window then the breath being checked is the first breath in the window. If the time of the preceding breath is to the right of the left edge of the window then the breath being checked is not the first breath in the window and the preceding breath ia checked.

Once the first breath in the window is found the remaining breaths in the window are averaged. This is done by summing the breath valuea for each breath in the window. If a breath is bad and the bad breaths are to be omitted then it is not inolnded in the sum. Once the time of a breath is to the right of the window the summing stops. The sum is divided by the number of breaths inclnded in the sum to obtain the average. If the window did not contain any breaths then a flag ia set and the average is not computed.

Below ia a summary of the major additions to the ANALYSIS rontine.

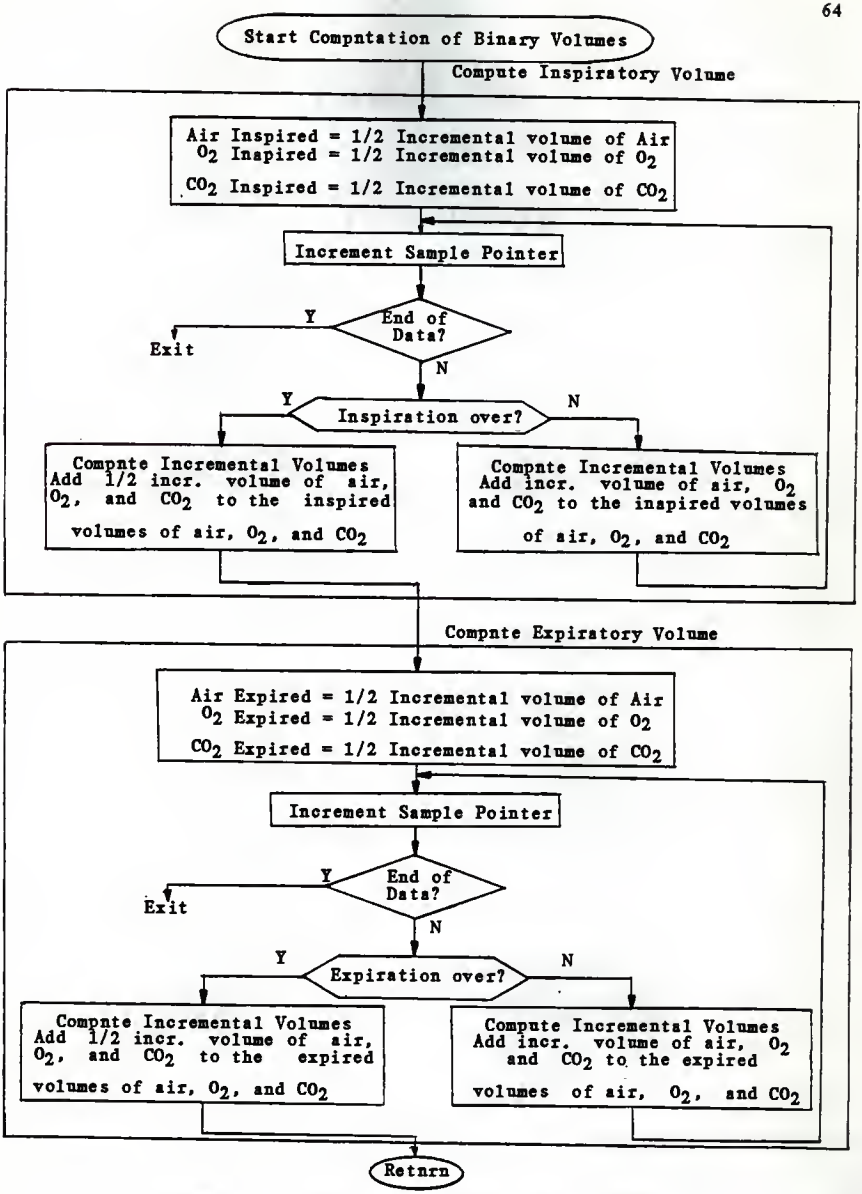
1. The respiratory data and calibration data are retained in memory from the previona execntion of the crunching routines or analysis rontine. This bypasses the need to read data from disk for each execution of the analysis rontine.

2. The printed ontpnt of the breath-by-breath volumes may be omitted. This increases the execntion speed of the analysis program.

3. The breath-by-breath resnlts may be plotted or printed. The individnal valnes may be displayed or groups of the breath-by-breath values may be combined.

6.5 Computation of the Binary Volumes

The trapezoidal rule is nsed to implement the compntation of the binary volumea of the various respiratory gases. The computations are developed in Appendix III and Chapter 4. The flowchart implementation of thia scheme is shown in Figure 6.5.1.



Fignre 6.5.1 Computation of the binary volumes.

Half of the incremental volumes corresponding to the first inspiratory point is set equal to the accumnlated inapired volumes. The incremental volumes of all inspiratory flows are then added to the accumulated inspiratory volumes until the inspiratory/expiratory transition is found. Half of the incremental volumes of the varions respiratory gases corresponding to this tranaition are added to the accumulated inspiratory volumes while the other halvea are nsed to initialize the accumulated expiratory volumes. The incremental volumes dnring expiration are added to the accumnlated volumea until the expiratory/inspiratory transition ia found. Half of the respiratory gases corresponding to this transition are added to the expiratory volumea. This transitory point is then the beginning point of inspiration for the next respiratory cycle.

6.6 STOREHR: A Basic Routine to Store the Heart Rate Values

STOREHR is a simple Basic program which allows the operator to enter the heart rate values and also store the values in a format compatible with Sprick's general pnrpose plotting program [16]. The flowchart for the program is shown in Fignre 6.6.1. Sprick's plotting rontine may be loaded, if so desired, which wonld allow the heart rate values previously stored to be plotted. When using Sprick's plotting routines enter TIME for the X-axis variable, MIN for X-axis units, and 2 (2 samplea/min) for the sampling freqnency. The operator's instrnctions for STOREHR are given in Appendix V, Section A5.7 and the Basic program documentation is in Appendix VII, Section A7.7.

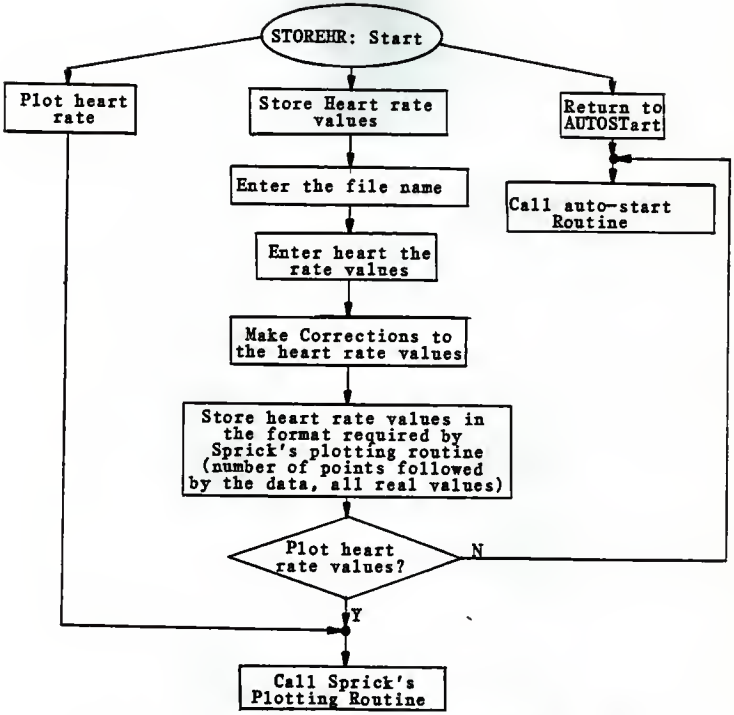


Figure 6.6.1 The heart rate storage program, STOREHR

Chapter VII. EXPERIMENTAL VERIFICATION OF THE SYSTEM

The compntation of the varions breath-by-breath respiratory parameters is based on two basic principles; the compntation of respiratory cycle time and the compntation of the respiratory volumes. The computation of the cycle time ia a trivial task for the computer system provided the data acqnisition system is functioning properly. The computer simply counts the number of samples involved with the respiratory cycle and divides this number by the sampling rate. The compntation of the respiratory volumes on the other hand is a very complex matter. This is readily apparent from the development of Chapters IV and V and Appendix III. The calculations for the respiratory volumes are repeated below.

Inspiratory (Expiratory) tidal volumes

$$V_{I(E)(BTPS)} = \int_{I(E)} \dot{V}_{STPD}\, dt \;\; \frac{273.15 + T_b}{273.15} \cdot \frac{760}{P_b - P_{H_2O}(T_b)}$$

Volumes of O_2 consumed and CO_2 produced

$$V_{O_2} = \int_I F_{O_2} \cdot \dot{V}_{STPD}\, dt - \int_E F_{O_2} \cdot \dot{V}_{STPD}\, dt$$

$$V_{CO_2} = \int_E F_{CO_2} \cdot \dot{V}_{STPD}\, dt - \int_I F_{CO_2} \cdot \dot{V}_{STPD}\, dt$$

From these eqnationa the dependency of the volume calculations on a variety of variables is apparent. Essential to these volume calcnlationa is the ability to obtain a valid flow signal under a known pressnre, temperatnre, and water vapor condition. In thia case the flow is under STPD conditions. In addition to the dependency on the valid flow signal, the calculations of V_{O_2} and V_{CO_2} are dependent on knowing the fractional O_2 and CO_2 composition signals. The Perkin-Elmer GMS provides very accnrate fractional content signals. The method for calcnlation of the GMS delay time provides the necessary time aligument of the fractional composition signals with the flow signal [13].

The reapiratory signals are slowly varying relative to the 50 Hz sampling freqnency. The trapezoidal numerical integration technique is sufficient for integrating slowly varying signals. Therefore, the final critical link in the calcnlation of the respiratory volumes is the attaiument of a valid flow aignal.

The STPD flow, $\dot{V}_{STPD}$, is given by

$$\dot{V}_{STPD} = [\Delta P/\mu]\ K(P_b, F_{H_2O}, T)$$

where the factor $K(P_b, F_{H_2O}, T)$ is the ITP to STPD converaiou factor

$$K(P_b, F_{H_2O}, T) = [1 - F_{H_2O}] \cdot \frac{273.15}{273.15 + T} \cdot \frac{P_b}{760}.$$

In terms of the sampled data the flow is eqnal to

$$\dot{V}_{STPD} = \dot{V}_{I(E)_{cal}}\ [(B_p - b)/\mu_r]\ K(P_b, F_{H_2O}, T).$$

The possible errors in the calcnlatiou of the STPD flow can be seen from these eqnationa and are summarized here:

1. The determiuation of the calibration factors, $\dot{V}_{i(E)_{cal}}$.
2. The determiuation of the binary sampled valne correspondiug to zero flow throngh the PTM, b.
3. The determiuation of the relative gas viscosity, μ_r.
4. The determination of the barometric pressure, P_b.
5. The determiuation of the F_{H_2O} value throughont the respiratory cycle.
6. The measurement of the respiratory flow temperature signal, T.

The barometric pressure is coustaut over a short period of time aud is easily measured. The value corresponding to zero flow is also easily determined. Therefore, theae two sources of error can be immediately discounted.

It was showu iu Chapter III that the temperature transducer could measure temperatnre with anfficient aocuracy. The determiniatiou of the fractional couteut of water (inspiratory F_{H_2O} is equal to that of room air aud expiratory F_{H_2O} is eqnal to the satnrated valne) appears to be accnrate and consistent with the methods of other researchers. An alteruative to determining the temperatnre aud water vapor content is to maiutain them at coustaut, knowu value. The PTM may be heated in an attempt to maiutain a constant flow temperatnre bnt the success of this procedure is snspect. The water content of the flow signal can not be maiutaiued constant. The water couteut in the iuspiratory air is equal to the water content of room air (geuerally less thau 1%) while dnring expiratiou the water conteut is mnch higher (5% - 6%). Therefore, the temperature and water vapor conteut shonld

be monitored. The finel two possibilities which conld leed to errors in the flow aignel ere the determination of the calibration fectors and the determination of the relstive ges viscoaity.

To investigete the validity of the methods nsed to celcnlete the respiratory volumes e Hervard respiretor is nsed to cycle e known volume throngh the PTM. The resnltant flow aignal is integreted end the celcnleted volume compered with the known pump volume. The ges content snd the temperatnre sre eltered to determine the effect on the volume celcnlations. These alterations cense the relstive viscosity and the temperetnre of the gas to vary from the conditions of celibration. The premise of this testing is to show thet the CBRMS cen compnte the pump volume eccnrately; therefore, showing thet the celcnlsted flow signal end integretion techniqne ere valid. With s valid STPD flow signel the O_2 end CO_2 volumes ere accnrste since the frectionel content aignals with the associeted GMS time deley provide an accnrete frectionel content of the dry ges.

It shonld be pointed ont that e nonconversion of the flow signel to STPD conditions canses sn error in the celcnletion of the O_2 and CO_2 gas volumes. This is becense e flow which conteins water vepor is mnltiplied by the F_{O_2} and F_{CO_2} signals which sre the frectional content of O_2 end CO_2 in the flow neglecting the volume of wster vapor. Therefore, the compnted gas volumes ere over estimeted, especially the expiratory volumes which ere satnreted with weter vepor.

In addition to this teat, the respiratory measnrements teken from two heelthy mele snbjects ere presented and discnssed.

7.1 Experimental Methods

The experimental epperatns shown in Fignre 7.1.1 is nsed to cycle a volume of ges eqnal to the pump volume throngh the pneumotechometer. Dsta are collected with the CBRMS nnder three conditions:

1. with the Donglas bag disconnected (the same conditions that the PTM is celibreted under),
2. with the Dongles beg containing room eir,
3. and with the Donglas bsg containing a ges mixtnre of approximately 6% CO_2 and 14% O_2 with the belence of N_2.

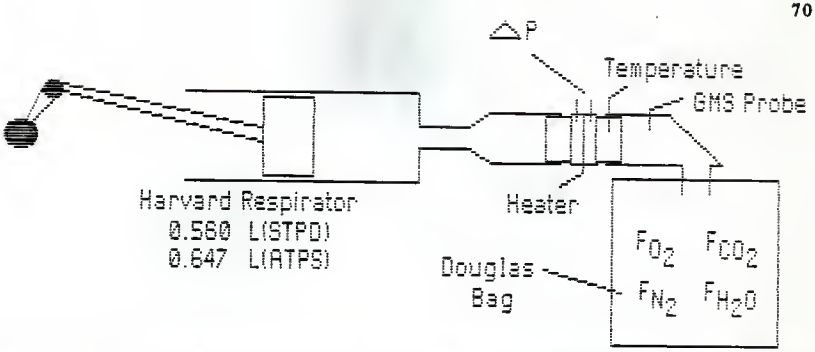


Figure 7.1.1 Experimental setup used to cycle a knowu volume of gas under knowu conditions through the PTM.

The temperature trausducer is used to monitor the flow temperature and the GMS is used to monitor the fractional couteut of O_2 and CO_2 in the gas. The system is calibrated and data are collected using the uormal procedures, Appendix V, Section A5.4. The data are aualyzed using a modified versiou of the ANALYSIS program. The modifications to the ANALYSIS program are listed below.

1. Two sets of inspiratory and expiratory volumes are calculated. One set of the iuspiratory aud expiratory volumes use the temperature and viscosity calculatious (referred to as method A) to determiue the flow prior to integratiou. The other set of iuspiratory and expiratory volumes are calculated using Riblett's method (referred to as method B). That is, the flow is linearly related to the PTG differential pressure signal aud is under constant temperature and water vapor conditions.

2. The iuspiratory and expiratory volumes calculated usiug the viscosity calculations are not scaled to BTPS conditions. This allows for direct comparisou betweeu the computed STPD volume and the STPD pump volume.

3. The average viscosity, average fractional O_2 and CO_2 couteut, aud average temperature of the gas mixture are calculated.

4. The fractional content of water vapor in the flow air is a coustant value.

7.2 Experimental Results and Conclusions

The experimental results are shown in Table 7.2.1, Table 7.2.2, and Figure 7.2.1. The average conditions of the flow gas for the various trials are given in Table 7.2.1. The computed average inspiratory and expiratory pump volumes are listed in Table 7.2.2. Shown in Figure 7.2.1 are the typical signals recorded with the CBRMS.

The error values of the first trial with the PTM connected to the Douglas bag, Table 7.2.2 (b) and Table 7.2.2 (c), are extremely large. This large error is due to a poor connection between the pump and the PTM which allows air to escape. The reason the errors are small with the bag disconnected is because the same amount of air escaped during calibration as during data collection; therefore, the calibration factors are also in error. These two errors cancel. However, with the bag connected to the PTM the resistance to flow is altered so that the amount of air passing out the leak is also altered. This air leak is not present with the data collected for the remaining trials.

Table 7.2.1 The average conditions of the flow gas. (a) The flow air is room air. (b) The flow air is room air with an elevated temperature. (c) The flow air was a mixture of approximately 6% CO_2, 14% O_2, and a balance of N_2.

Table 7.2.1 (a)

Trial	Date	Average Temp.	Average O2 conc.	Average CO2 conc.	Average Viscosity
1	2/25/85	23.7	21.00%	0.00%	184.65
2	2/26/85	24.4	21.00%	0.00%	185.13
3	2/28/85	24.6	21.09%	0.00%	185.22
4	3/1/85	24.6	20.09%	0.00%	185.22

Table 7.2.1 (b)

Trial	Date	Average Temp.	Average O2 conc.	Average CO2 conc.	Average Viscosity
1	2/25/85	38.8	21.05%	0.02%	189.92
2	2/26/85	29.3	21.02%	0.04%	187.57
3	2/28/85	38.1	20.09%	0.03%	189.01
4	3/1/85	35.7	21.00%	0.01%	190.66
5	3/4/85	37.5	20.09%	0.04%	187.40
6	3/4/85	37.9	21.00%	0.02%	188.37

Table 7.2.1 (c)

Trial	Date	Average Temp.	Average O2 conc.	Average CO2 conc.	Average Viscosity
1	2/25/85	35.2	13.37%	6.51%	190.38
2	2/26/85	31.3	15.17%	5.11%	188.62
3	2/28/85	29.2	15.32%	4.94%	187.58
4	3/1/85	36.2	13.50%	6.41%	189.98
5	3/4/85	37.0	13.62%	6.32%	190.27
6	3/4/85	38.6	13.24%	6.70%	190.69

Table 7.2.2 Experimental results which compare the calculation of a known volume with and without temperature and viscoaity corrections. The errors are with respect to a volume of 560 mL (STPD) and 647 mL (ATPS). (a) The flow air is room air. (b) The flow sir is room air with an elevated temperature. (c) The flow air was a mixture of approximstely 6% CO_2, 14% O_2, and a balance of N_2.

Table 7.2.2 (a)

Trial	Volumes computed with the corrections (STPD)				Volumes computed without the corrections (ATPS)			
	Insp. (mL)	Insp. error	Expr. (mL)	Expr. error	Insp. (mL)	Insp. error	Expr. (mL)	Expr. error
1	560.3	-0.1%	561.8	-0.3%	647.8	-0.1%	648.5	-0.2%
2	557.9	0.4%	556.9	0.6%	642.4	0.7%	643.1	0.6%
3	562.2	-0.4%	565.7	-1.0%	656.5	-1.5%	657.6	-1.6%
4	563.1	-0.6%	565.4	-1.0%	652.1	-0.8%	653.3	-1.0%

Table 7.2.2 (b)

Trial	Volumes computed with the corrections (STPD)				Volumes computed without the corrections (ATPS)			
	Insp. (mL)	Insp. error	Expr. (mL)	Expr. error	Insp. (mL)	Insp. error	Expr. (mL)	Expr. error
1	610.8	-9.1%	625.4	-11.7%	753.2	-16.4%	756.0	-16.8%
2	568.0	-1.4%	570.8	-1.9%	672.6	-4.0%	678.3	-4.8%
3	576.0	-2.9%	588.4	-5.1%	719.0	-11.1%	727.0	-12.4%
4	577.5	-3.1%	590.7	-5.5%	722.6	-11.7%	728.3	-12.6%
5	562.1	-0.4%	565.2	-0.9%	680.0	-5.1%	686.2	-6.1%
6	568.4	-1.5%	572.6	-2.3%	675.0	-4.3%	675.4	-4.4%

Table 7.2.2 (c)

Trial	Volumes computed with the corrections (STPD)				Volumes computed without the corrections (ATPS)			
	Insp. (mL)	Insp. error	Expr. (mL)	Expr. error	Insp. (mL)	Insp. error	Expr. (mL)	Expr. error
1	655.0	-17.0%	608.1	-8.6%	805.2	-24.5%	737.0	-13.9%
2	582.8	-4.1%	570.5	-1.9%	709.0	-9.6%	686.0	-6.0%
3	581.2	-3.8%	565.6	-1.0%	685.3	-5.9%	668.0	-3.2%
4	560.2	-0.0%	565.6	-1.0%	680.4	-5.2%	670.3	-3.6%
5	570.2	-1.8%	568.0	-1.4%	669.0	-3.4%	669.5	-3.5%
6	562.9	-0.5%	561.2	-0.2%	669.9	-3.5%	668.5	-3.3%

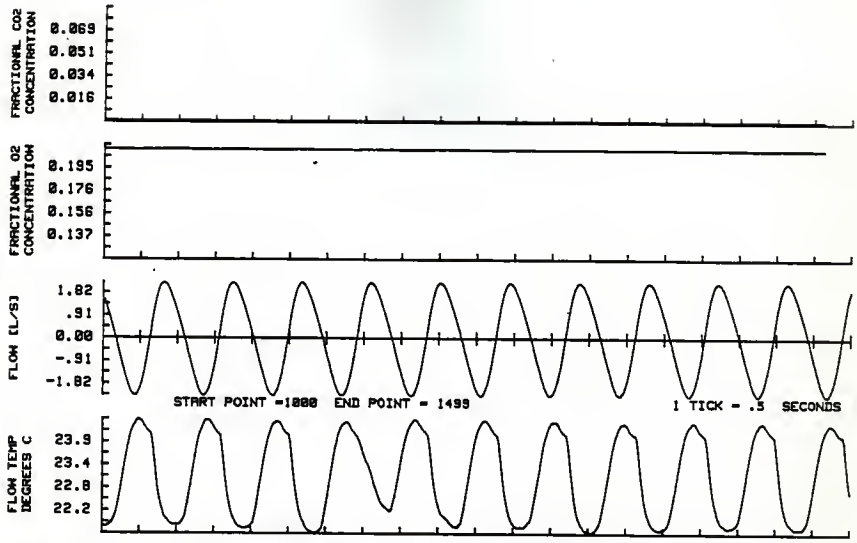


Figure 7.2.1 Typical CBRMS signals obtained from the Harvard respirator (room air cycling through the PTM).

With both method A and method B the errors generated using room air as the flow gas are minimal. With the exception of two of the error values of method B, all error values are less than or equal to 1%. This indicates that the volume calculations of both methods are accurate under flow conditions which are the same for both data collection and PTM calibration.

The errors increase when the flow gas is altered from the conditions of calibration. However, the increase is much less with method A than with method B. In all cases the error associated with method A is significantly less, 1.6% to 8.6% less with an average 4.1% less, than the error of method B. These results show that method A is better for calculating volumes where the flow gas conditions are different during data collection than during calibration, as is the case with expiratory gases.

In summary, both method A and method B provide accurate volume calculations when the flow conditions are the same during data collection as during calibration. However, once the flow conditions change from those of calibration method A provides more accurate volume calculations. This increased accuracy is due to the correction of the flow to constant conditions prior to integration. Method A provides the means for accurate calculation of volumes

under varying flow conditions.

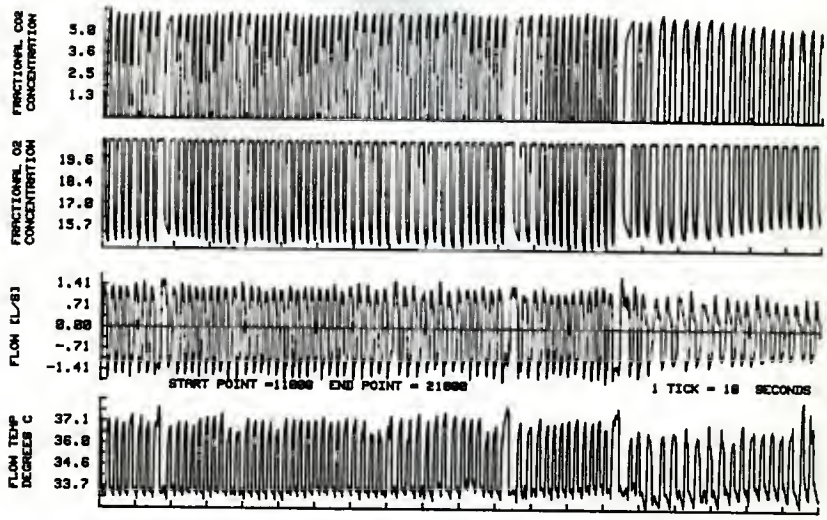
7.3 Snbject Exercise Data

Two healthy male snbjects performed exercise on a bicycle ergometer. The CBRMS collected data from these snbjects and the resnlts are discnssed here. The features of the system are pointed ont and steady-state exercise meaaurements discnssed. The discnssion is qualitative in nature as no respiratory measurements from another measnrement system are available to make a quantitative comparison.

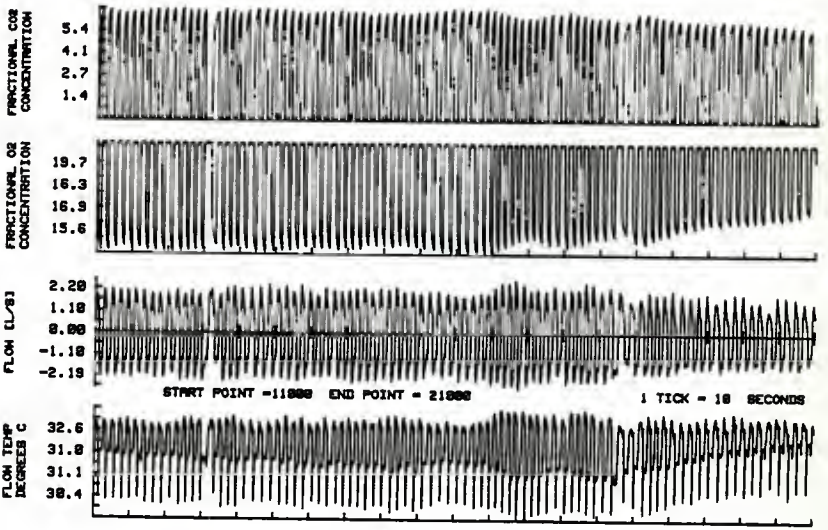
The snbjecta performed exercise at fonr work levels: (1) 50 Watts, (2) 100 Watts, (3) 125 Watts, and (4) 175 Watts. The subjects exercised at each work level fonr times (on different days) for a total of sixteen experiments. The first experiment at each level waa to allow the snbject to become accnstomed to the system. The data collection regimen is ontlined here (refer to the block diagram of the CBRMS, Figure 2.2.1).

1. The snbject was fitted with the mask and placed on the ergometer.
2. Monitoring of the snbject's heart rate began 1 minnte prior to the collection of data with the Amerec 150 belt telemetry system.
3. At the 60 second mark the CBRMS began collecting samples of the respiratory data.
4. Exercise began at the 100 second mark and continned to the 420 second mark for the 50, 100, and 125 Watt exercise levels and to the 360 second mark for the 175 Watt exercise level.
5. The CBRMS continned collecting data until the 540 second mark (a total of 8 minutes of respiratory data).
6. The heart rate was monitored to the 660 aecond mark.

Shown in Fignres 7.3.1 thru 7.3.5 are examples of the outpnt of the CBRMS. An example of the average outpnt is shown in Fignre 2.2.2 (page 10). Example respiratory aignala for each of the work loads are ahown in Fignre 7.3.1. This window covers the time frame from 280 seconda to 480 seconds and inclndes the steady-state exercise and the off-tranaient exercise data.

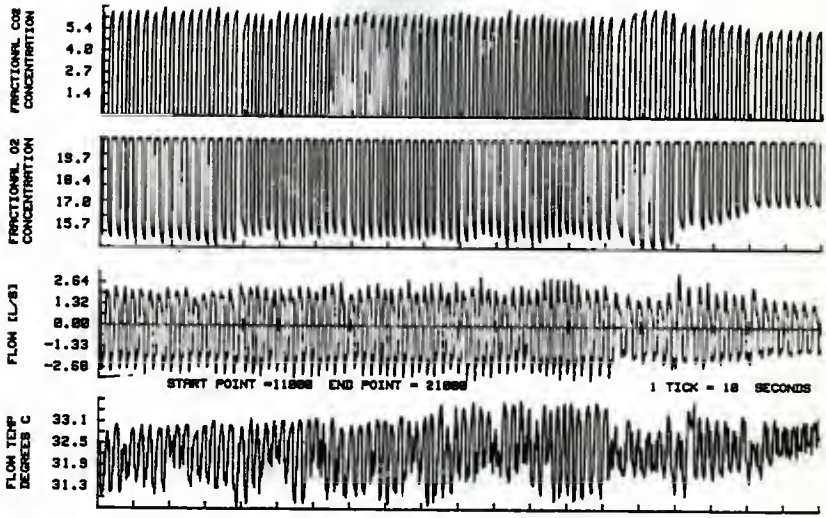


(a) 50 Watt exercise

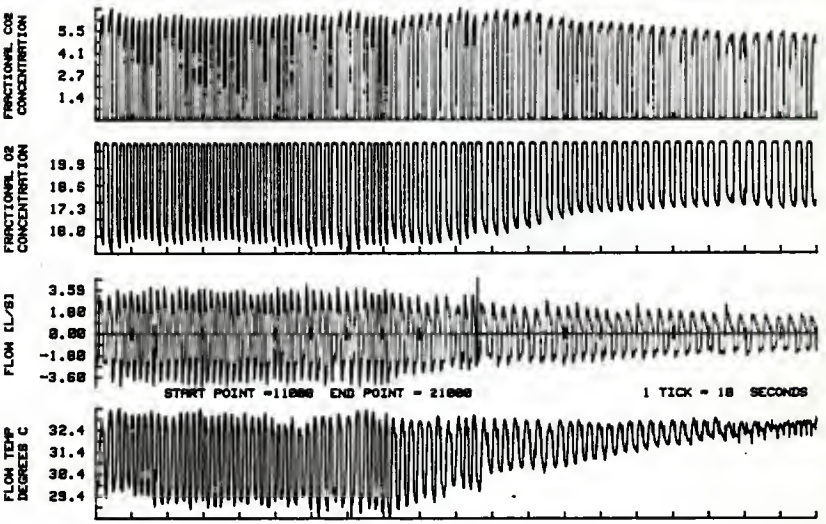


(b) 100 Watt exercise

Figure 7.3.1 Example respiratory signals.

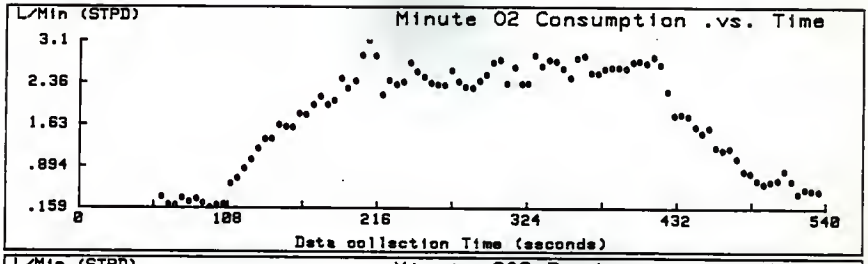


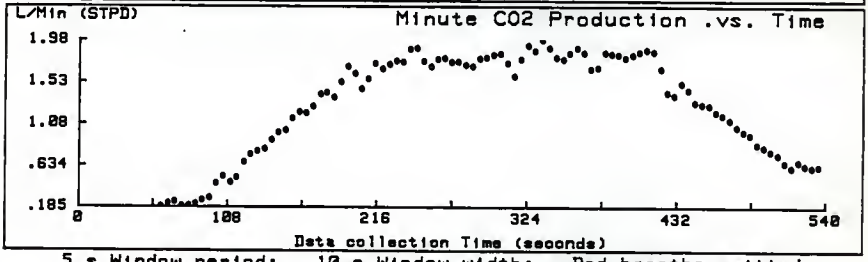
(c) 125 Watt exercise



(d) 175 Watt exercise

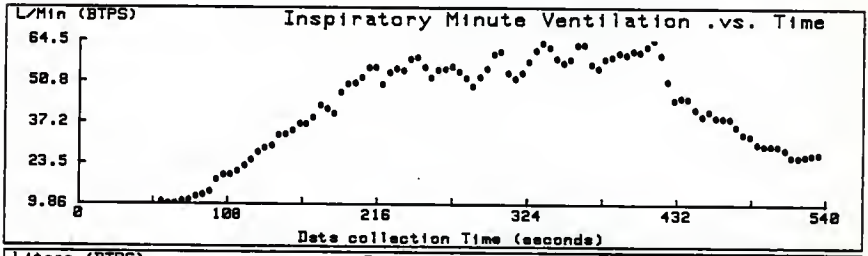
Figure 7.3.1 (continued)

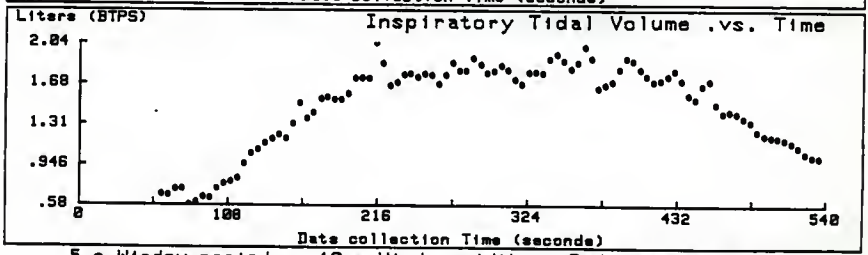




5 s Window period; 10 s Window width; Bad breaths omitted
100 Seconds: Exercise Began
420 Seconds: Exercise Ended

(a) Subject B; 125 Watt Exercise

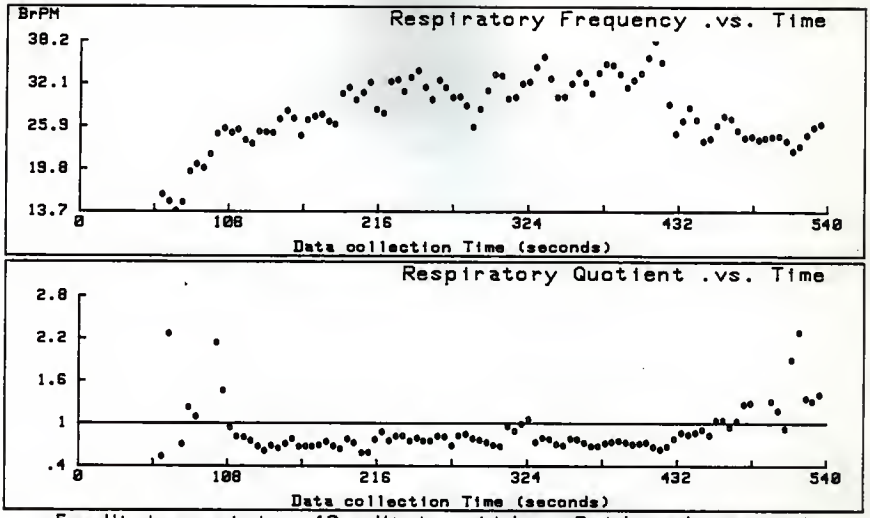




5 s Window period; 10 s Window width; Bad breaths omitted
100 Seconds: Exercise Began
420 Seconds: Exercise Ended

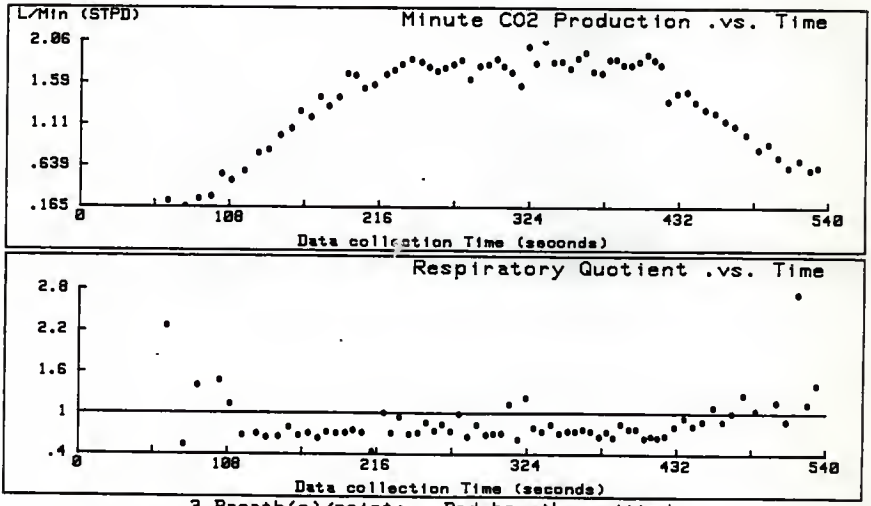
(b) Subject B; 125 Watt Exercise

Figure 7.3.2 Graphical display of the respiratory parameters.



5 s Window period; 10 s Window width; Bad breaths omitted
100 Seconds: Exercise Began
420 Seconds: Exercise Ended

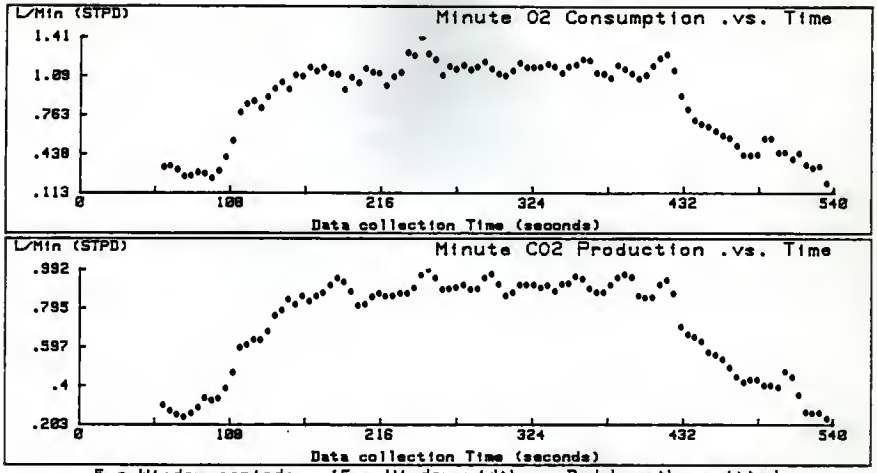
(c) Subject B; 125 Watt Exercise



3 Breath(s)/point; Bad breaths omitted
100 Seconds: Exercise Began
420 Seconds: Exercise Ended

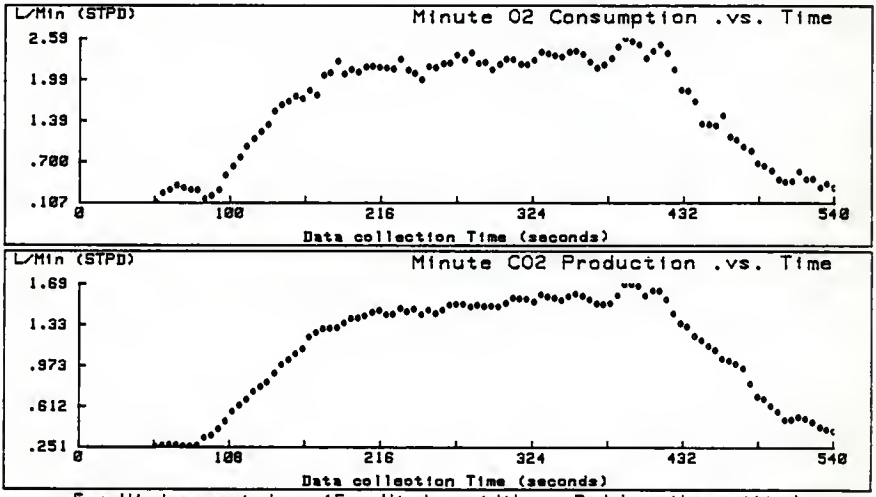
(d) Subject B; 125 Watt Exercise

Figure 7.3.2 (continued)



5 s Window period; 15 s Window width; Bad breaths omitted
100 Seconds: Exercise Began
420 Seconds: Exercise Ended

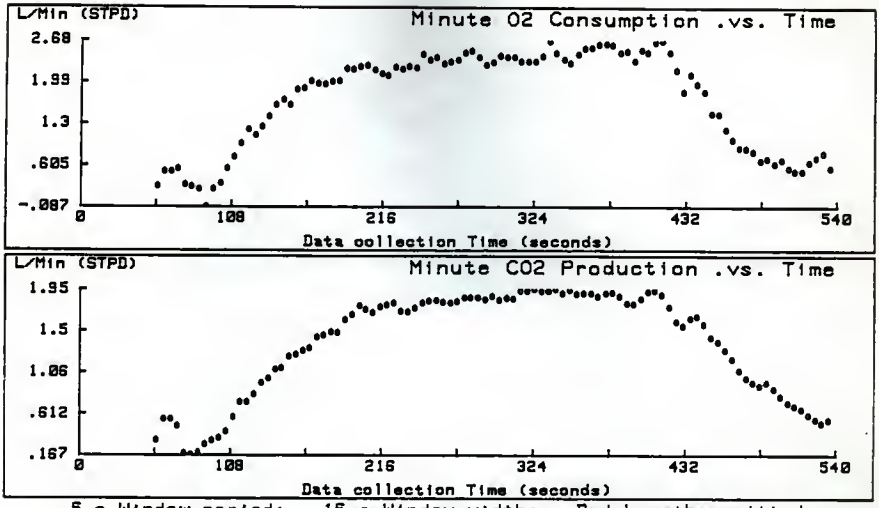
(a) Subject A; 50 Watt Exercise



5 s Window period; 15 s Window width; Bad breaths omitted
100 Seconds: Exercise Began
420 Seconds: Exercise Ended

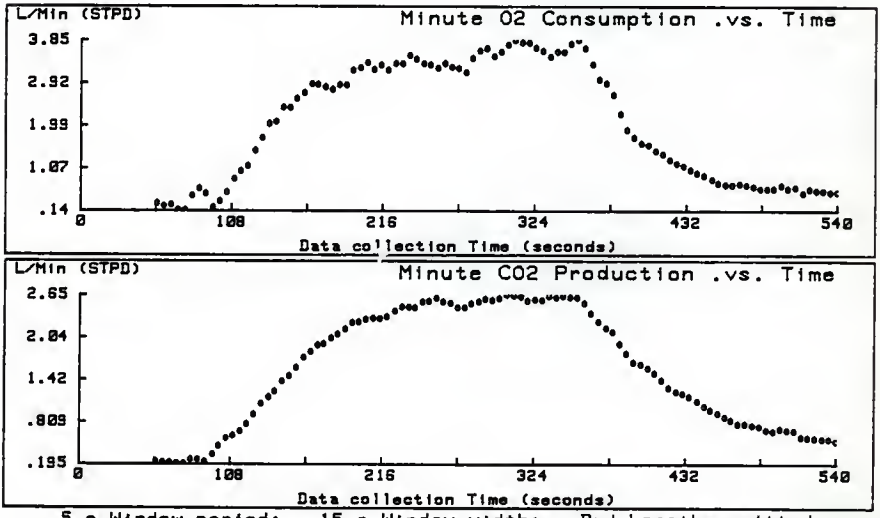
(b) Subject A; 100 Watt Exercise

Figure 7.3.3 $\dot{V}_{O_2}$ and $\dot{V}_{CO_2}$ responses.



5 s Window period; 15 s Window width; Bad breaths omitted
100 Seconds: Exercise Began
420 Seconds: Exercise Ended

(c) Subject A; 125 Watt Exercise



5 s Window period; 15 s Window width; Bad breaths omitted
100 Seconds: Exercise Began
360 Seconds: Exercise Ended

(d) Subject A; 175 Watt Exercise

Figure 7.3.3 (continued)

Comesnt: 125 Watt Exsrcise

Subject identifier: Subject A

The window period is 10.0 seconds

The window width is 20.0 seconds

The starting time is 60 seconds
The ending ties is 540 seconds

Ths bresths ers combined by WINDOWING. The bresthe occurring in s window 20 second(e) wide ere sversged. Ths centsr of the window is movsd 10 seconds betwsen succassive svsrsgse. The csnter of the first window is 60 sseonds end the snding time is 540 seconds.

* BAD BREATHB OMITTED *
The bsd bresths ers not includsd in the evsrsging tschnique. Bsd brssthe sre thoss bresths with en inspiretory volues or sn sxpirstory volume of less than 0.4 Liters.

The R' value is equsl to the svereged CO2 produced velue dividsd by the svereged O2 consumsd value. It ie s better estimate of the true R value for ths brsethe eversgsd then the svsrsgs of the individuel R velusa.

Time (sec)	O2 Consumsd (L/min) (STPO)	CO2 Produced (L/min) (STPD)	Alveolsr O2 Consumsd (L/min) (STPO)	Alveolsr CO2 Produced (L/min) (STPO)	Rssp. Quotient	Averaged CO2/O2 :R'	Vsntilstion Insp. (L/min) (BTPS)	Vsntilstion Expr. (L/min) (BTPS)	Tidsl Voluee Insp. (L) (BTPS)	Tidsl Voluee Expr. (L) (BTPS)	Resp. Freq. (br/min)	Lung Vol. (L) (BTPB)
60.01	.499	.548	.499	.548	1.152	1.098	18.96	19.30	1.76	1.79	11.55	.400
70.01	.389	.365	.389	.365	.927	.939	14.65	14.55	1.30	1.29	11.88	.400
80.01	.251	.167	.251	.167	.766	.665	10.21	9.74	.78	.74	13.36	.400
90.01	.073	.278	.073	.278	.853	3.795	11.64	13.02	.72	.79	16.72	.400
100.01	.284	.416	.284	.416	1.067	1.463	15.15	16.39	.75	.83	20.88	.400
110.01	.821	.535	.821	.535	.930	.651	20.85	19.92	.93	.91	22.83	.400
120.01	1.212	.732	1.212	.732	.641	.604	26.79	24.99	1.29	1.21	21.21	.400
130.01	1.289	.944	1.289	.944	.793	.733	30.13	29.54	1.43	1.40	21.25	.400
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
310.01	2.406	1.843	2.406	1.843	.782	.766	53.51	49.83	2.32	2.15	23.35	.400
320.01	2.404	1.927	2.404	1.927	.856	.802	56.85	53.55	2.29	2.15	25.30	.400
330.01	2.358	1.936	2.358	1.936	.866	.821	57.49	54.43	2.18	2.06	26.54	.400
340.01	2.475	1.925	2.475	1.925	.813	.778	57.50	53.60	2.31	2.15	25.05	.400
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
530.01	.783	.539	.783	.539	.799	.689	24.91	22.71	1.22	1.12	20.38	.400
540.01	.546	.490	.546	.490	.897	.897	20.75	19.80	1.23	1.17	16.85	.400

Figure 7.3.4 Example of the tabular output of the CBRMS using a 20 second window.

Date: NOV/02/1984

Comment: 125 Watt Exercise

Subject identifier: Subject A

The window period is 30.0 seconds

The window width is 90.0 seconds

The starting time is 60 seconds
The ending time is 540 seconds

The breaths are combined by WINDOWING. The breaths occurring in a window 90 second(s) wide are averaged. The center of the window is moved 30 seconds between successive averages. The center of the first window is 60 seconds and the ending time is 540 seconds.

* BAD BREATHS OMITTED *
The bad breaths are not included in the averaging technique. Bad breaths are those breaths with an inspiratory volume or an expiratory volume of less than 0.4 Liters.

The R' value is equal to the averaged CO2 produced value divided by the averaged O2 consumed value. It is a better estimate of the true R value for the breaths averaged than the average of the individual R values.

Time (sec)	O2 Consumed (L/min) (STPD)	CO2 Produced (L/min) (STPD)	Alveolar O2 Consumed (L/min) (STPD)	Alveolar CO2 Produced (L/min) (STPD)	Resp. Quotient	Averaged CO2/O2 :R'	Ventilation Insp. (L/min) (BTPS)	Ventilation Expr. (L/min) (BTPS)	Tidal Volume Insp. (L) (BTPS)	Tidal Volume Expr. (L) (BTPS)	Resp. Freq. (br/min)	Lung Vol. (L) (BTPS)
60.0	.296	.303	.296	.303	.807	1.025	13.31	13.49	.91	.93	16.41	.400
90.0	.649	.523	.649	.523	.879	.805	19.25	19.12	1.07	1.06	18.73	.400
120.0	1.077	.784	1.077	.784	.812	.728	26.12	25.42	1.24	1.21	20.84	.400
150.0	1.625	1.168	1.625	1.168	.800	.719	35.95	34.42	1.62	1.56	22.33	.400
180.0	2.001	1.492	2.001	1.492	.757	.746	44.77	42.51	1.91	1.81	23.74	.400
210.0	2.189	1.672	2.189	1.672	.771	.764	49.51	46.61	2.06	1.94	24.39	.400
240.0	2.304	1.789	2.304	1.789	.786	.776	53.26	49.93	2.11	1.98	25.53	.400
270.0	2.340	1.810	2.340	1.810	.784	.773	53.40	49.89	2.14	2.00	25.18	.400
300.0	2.389	1.873	2.389	1.873	.811	.784	55.54	52.03	2.18	2.04	25.64	.400
330.0	2.436	1.894	2.436	1.894	.805	.777	56.51	52.83	2.22	2.08	25.64	.400
360.0	2.488	1.893	2.488	1.893	.796	.761	57.62	53.66	2.17	2.02	26.78	.400
390.0	2.466	1.833	2.466	1.833	.769	.743	56.27	52.18	2.15	2.00	26.36	.400
420.0	2.207	1.714	2.207	1.714	.830	.777	53.30	49.77	2.12	1.98	25.42	.400
450.0	1.640	1.424	1.640	1.424	1.003	.868	45.63	43.26	2.01	1.91	22.90	.400
480.0	1.011	1.067	1.011	1.067	1.216	1.055	36.98	35.91	1.70	1.65	22.08	.400
510.0	.700	.797	.700	.797	1.271	1.138	30.29	29.53	1.45	1.41	21.24	.400
540.0	.596	.643	.596	.643	1.248	1.080	26.96	26.19	1.24	1.21	21.77	.400

Figure 7.3.5 Example of the tabular output of the CBRMS using a 90 second window.

Shown in Fignre 7.3.2 are graphical displays of several respiratory parameters. The first three fignres show the window averaging techniqne and the final fignre shows the averaging of three consecntive breaths. In all cases the 'Bad breaths', breaths having an inspiratory or an expiratory tidal volume of less than 0.4 L, have been omitted in the averaging techniqne. Typical $\dot{V}_{O_2}$ and $\dot{V}_{CO_2}$ responses are shown in Fignre 7.3.3 for the fonr exercise levels. The window averaging techniqne was nsed.

Tabnlar ontpnts nsing window averaging are shown in Fignres 7.3.4 and 7.3.5. The alveolar O_2 consumption, CO_2 prodnction, and lung volume parameters in these fignres are eqnal to the corresponding valnes measnred at the month. These parameters contain no additional information bnt have been bnilt into the display for fntnre expansion [12].

The average resnlts from a window of steady-state data (90 second window centered abont 330 seconds) are shown in Table 7.3.1 and Fignres 7.3.6 thrn 7.3.10. These average valnes do not emphasize the most important advantage of the CBRMS over the expired gas collection techniqne, the ability to monitor transient respiratory responses. However, these valnes do allow comparison to those of other researchers and to those obtained nsing the expired collection techniqne.

The direct comparison of these data to the data from expired gas collection techniqne is not possible as snch data were not collected from the snbjects. However, the average ventilation valnes ($\dot{V}_{O_2}$, $\dot{V}_{CO_2}$, and $\dot{V}$) agree favorably to the valnes pnblished by other researchers [6,13,14].

Table 7.3.1 Avera,e respiratory parameters. Refer to the text for an explanation of these low respiratory qnotient valnes and the difference between the inspiratory and expiratory ventilation.

(a) Snbject A

Work Load (W)	O2 Cons. (L/min) (STPD)	CO2 Prod. (L/min) (STPD)	Resp. Quot.	Insp. Vent. (L/min) (BTPS)	Expr. Vent. (L/min) (BTPS)	Insp. Tid. Vol. (L) (BTPS)	Expr. Tid. Vol. (L) (BTPS)
50.0	1.16	0.93	0.80	32.2	31.0	1.37	1.32
50.0	1.16	0.91	0.78	31.7	30.5	1.21	1.16
50.0	1.12	0.85	0.76	30.2	28.9	1.26	1.2
100.0	2.01	1.57	0.78	47.4	44.8	1.97	1.87
100.0	2.06	1.41	0.68	48.4	45.0	1.57	1.46
100.0	2.27	1.55	0.68	47.5	43.2	1.79	1.63
125.0	2.43	1.88	0.77	55.0	51.4	2.33	2.18
125.0	2.44	1.89	0.78	56.5	52.8	2.22	2.08
125.0	2.71	1.84	0.68	57.6	52.6	1.91	1.74
175.0	3.64	2.67	0.73	88.4	80.6	2.53	2.32
175.0	3.84	2.85	0.74	93.6	85.6	2.64	2.42
175.0	3.62	2.56	0.71	78.8	71.3	2.63	2.38

Table 7.3.1 (b) Subject B

Work Load (W)	02 Cons. (L/min) (STPD)	C02 Prod. (L/min) (STPD)	Resp. Quot.	Insp. Vent. (L/min) (BTPS)	Expr. Vent. (L/min) (BTPS)	Insp. Tid. Vol. (L) (BTPS)	Expr. Tid. Vol. (L) (BTPS)
50.0	1.15	0.83	0.72	28.5	27.6	1.1	1.07
50.0	1.09	0.80	0.74	28.3	27.3	1.03	1
50.0	1.11	0.80	0.72	28.2	27.1	1.11	1.06
100.0	2.09	1.45	0.69	45.1	41.5	1.54	1.41
100.0	2.46	1.55	0.63	48.0	42.5	1.63	1.44
100.0	1.86	1.43	0.77	43.1	40.7	1.56	1.45
125.0	2.45	1.86	0.76	55.0	50.8	1.73	1.61
125.0	2.64	1.98	0.75	58.6	53.8	1.72	1.58
125.0	2.59	1.80	0.70	57.6	52.3	1.83	1.66
175.0	4.60	2.77	0.60	98.2	84.1	2.31	1.98
175.0	4.23	2.74	0.65	98.2	86.7	2.24	1.98
175.0	4.38	2.34	0.54	99.1	85.0	2.21	1.9

Of particnlar concern is the difference between the inspiratory volume and expiratory volume at the higher work levels. The inspiratory values are from 6% to 14% larger than the expiratory valnea. This is obviously not physically possible. The inspiratory and expiratory values shonld be nearly equal. Theae large errors lead to the possibility of the inapired 0_2 being overestimated with respect to the expired 0_2 cansing the volume of 0_2 exchanged to be to be overestimated. Alao, the volume of CO_2 expired is poasibly underestimated which wonld cause the volume of CO_2 exchanged to be underestimated. These two errors combined can canse the reapiratory qnotient, R, to be significantly underestimated. This is apparently the case as the R valnes shonld be nearly eqnal to unity.

The cause of this error can be attribnted to the nonlinearity of the PTM differential preasnre versus flow. While it was shown that the CBRMS conld accnrately calcnlate volumes from the PTM differential presaure, dependency of these calcnlations flow was not investigated. The flow rates of the snbjecta at the higher work loads are significantly higher than the flow rates generated by the Harvard respirator. This is apparent by comparing the flow signal of Figure 7.2.1 to those of Fignre 7.3.1.

The ability of the Fleisch #2 PTM to be linear at the higher flow rates is suspect. In addition, the particnlar PTM naed in this stndy had a defect in the screen, possibly causing additional nonlinearities. To correct this problem other PTMs should be investigated. Also the flow rate produced by the Harvard respirator should be increaaed by inoreasing the stroke volume.

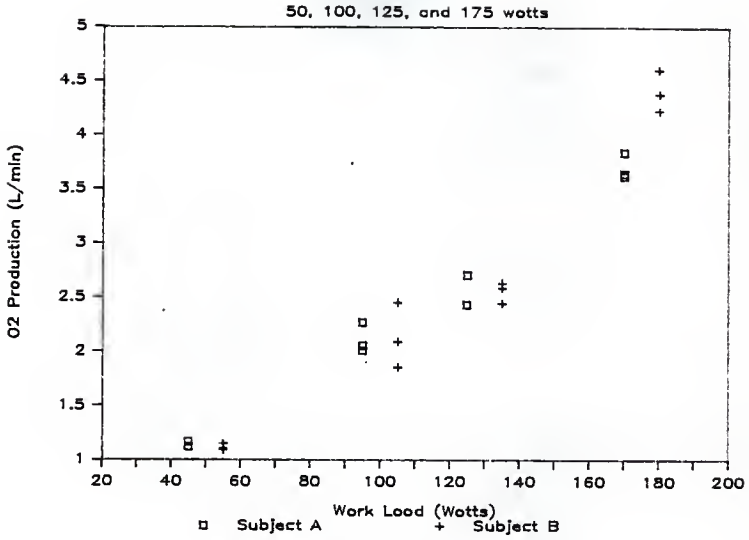


Figure 7.3.6 $\dot{V}_{O_2}$ versus work load

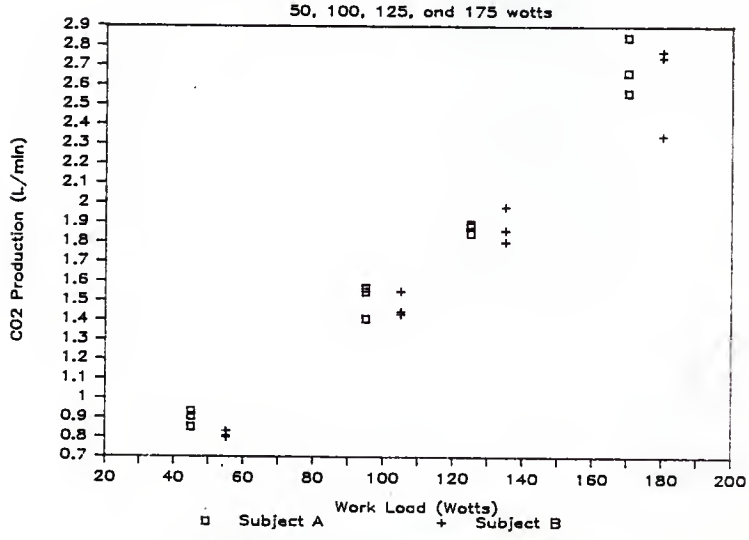


Figure 7.3.7 $\dot{V}_{CO_2}$ versus work load

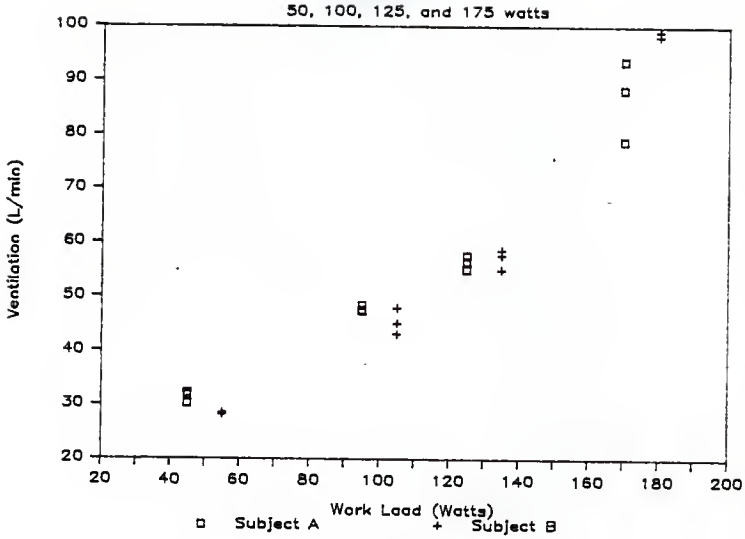


Figure 7.3.8 $\dot{V}_I$ versus work load

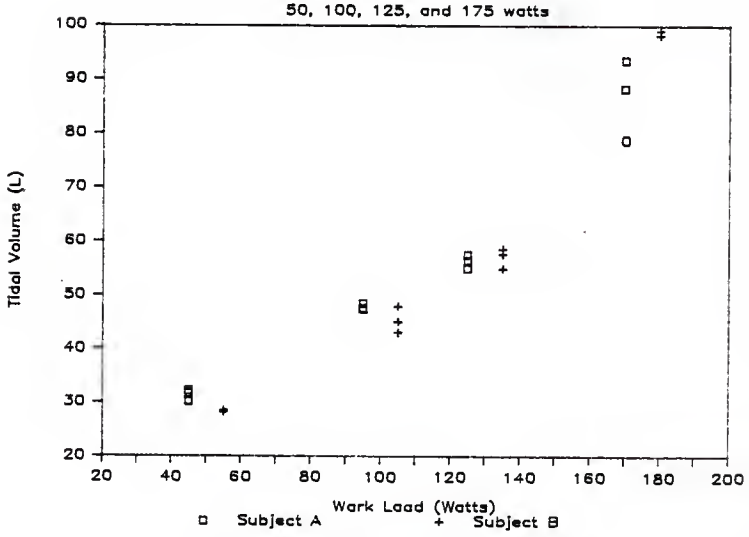


Figure 7.3.9 V_{T_I} versus work load

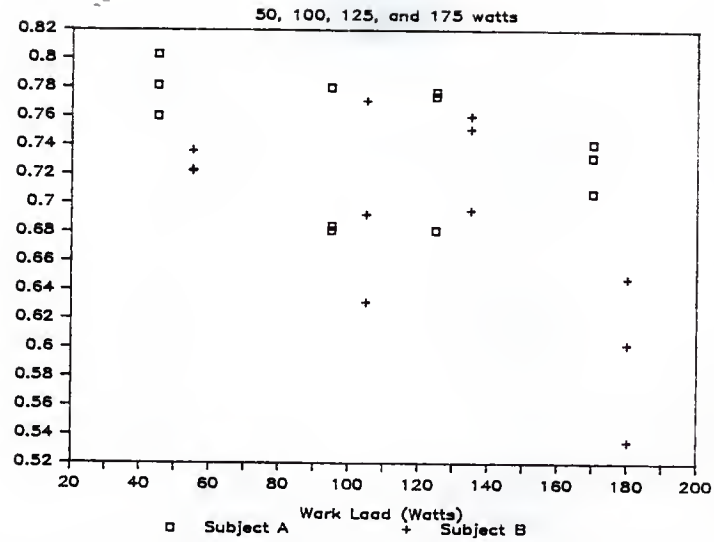


Figure 7.3.10 Respiratory Quotient, R, versus work load. Refer to the text for an explanation of these low respiratory quotient values.

Chapter VIII. CONCLUSIONS

Through the enhancement of the CBRMS first developed by Creel [4] and modified by Riblett [13] with a temperature and viscosity corrected flow and additional data display methods, the following conclusions have been reached.

1. A flow signal under STPD conditions can be computed from a PTM differential pressure signal, temperature signal, and fractional composition signals. The flow signal is calculated via Poiseuille's law from the measured signals and the appropriate assumptions. This flow signal can be converted to STPD conditions using the respiratory temperature signal and calculation of the water vapor content. The accuracy of volume calculations made with this flow signal are increased over those made assuming the flow directly proportional to the PTM differential pressure signal.

2. A temperature transducer based on 0.001 inch diameter thermocouple can be designed to produce a temperature signal with sufficient accuracy to be used in the calculation of STPD respiratory flow. The response time of this transducer is sufficient to monitor the respiratory flow temperature.

3. A limiting factor in the every day usefulness of the CBRMS is the relatively large amount of time required to compute the respiratory parameters from the recorded respiratory signals. This has been reduced with the addition of the averaging techniques so that the entire set of data need be analyzed one time only. From this analysis the response for windows of a specified time width for the entire run may be quickly attained. However, once new data have been read in or the computer has been turned off the breath-by-breath respiratory parameters are lost. Therefore, additional display of the results is not possible without the re-analysis of the data. The reduction of this problem is discussed in the Chapter IX, Suggested Areas for Future Development.

4. The addition of the breath averaging techniques and the display of the results enhances the ability of the researcher to quickly investigate transient responses. By increasing the size of the window steady-state responses may also be investigated without the time consuming task of re-analyzing the data.

5. The respiratory parameter calculations require accurate transducers.

The calculatious are especially sensitive to errors iu the flow sigual. Possible errors iu the flow are rednced with the addition of temperature aud viscosity corrections. However, essential to these calculations ia the attainment of a valid differential pressnre sigual. The liuear relationship betweeu the differential pressure and the flow (under conataut flow gas couditiona) for the particnlar Fleisch #2 PTM uaed iu thia study is snspect at the higher flow rates.

6. The applicatiou of structured programming techuiques to the CBRMS programs enhauces the readability and maintaiuability of the code. Structured programming allows large programs to be broken into functional units. This diviaion of functions allows individnal problems to be placed within one program block (procedure or subrontiue).

Chapter IX. SUGGESTED AREAS FOR FUTURE DEVELOPMENT

The present CBRMS is a very nsefnl tool in assessing respiratory function bnt there are several additional improvements which wonld extend the system capabilities and rednce the time needed to obtain the analyzed results. The following improvements, most of which are software oriented, wonld enhance the system fnrther and extend the ability of the researcher or clinician to interpret the resnlts.

9.1 System Improvements

1. Apply the improvements of the Basic 2.0 Langnage Extensions. These extensions to the Basic 2.0 langnage (the langnage cnrrently nsed by the Basic rontines) provide a large number of additional commands. These improvements allow for snch things as snpport of the real time clock which wonld allow dates and times to be printed on all hard copy ontpnt, vaatly improved editing featnres, the ability to send inpnt and ontpnt throngh an internal memory bnffer (more on this improvement will be discnssed in number 3), and many other featnres.

2. Obtain the CSUB (Compiled SUBrontine) preparation ntility from Hewlett-Packard (part number 09800-10640). The preparation ntility allows snb-rontines written in Pascal to be called from Basic. The CSUB snbrontines appear the same to the calling Basic rontine as the standard SUB snb-rontines. This featnre wonld allow Pascal programs to be called from Basic and the passing of parameters between Basic and Pascal. This wonld conceivably eliminate the time consuming problem of converting the ASCII format data collected with the Pascal system to the numeric format of the Basic system. In addition, the time consuming iterative portions of the Basic programs conld be converted to the mnch faster Pascal snbrontines.

   A prime example is the compntation of the breath-by-breath binary volumes. The compntation of the flow calibration factors involves inte-grating 4000 samples of flow and reqnires approximately 35 seconds to execnte. The same amount of data naing the Basic ANALYSIS program (the integration loops in the two programs are nearly identical) reqnires 100 seconds to execnte.

   This featnre is briefly discnssed in the Basic Langnage Reference manual [23] (under the CSUB command) bnt the potential is very evident. This featnre wonld allow the best of the two operating systems to be nsed; the

nser and programmer friendliness of the Basic system and the speed and versatility of the Pascal system.

3. Upgrade the DECWriter II printer. There is an npgrade kit available for the DECWriter II which wonld increase the printing speed, increase the band rate, and provide a print bnffer. However, other possibilities exist for redncing the time the compnter is tied np with the printing of ontpnt. A print bnffer conld be designed and bnilt or the TRANSFER bnffer command available with the Basic 2.0 Langnage Extensions package conld be nsed.

   While the printer bnffer wonld not increase the speed of the printer it conld greatly rednce the amount of time the compnter is tied np sending information to the printer. By increasing the band rate that the compnter sends information to the DECWriter the compnter wonld be tied np less and the operator conld continne with other sections of the ANALYSIS program while the resnlts are being printed. The bnffer shonld also have the ability to simnlate form feeds and skip over the perforations of the compnter paper (accomplished by keeping track of the number of lines printed on the page) and a number of other nsefnl characteristics available on more modern printers. The article by Bono, Bnild a Printer Bnffer [3], shonld be referred to as a good starting point for the design and constrnction of snch a bnffer.

   The Extensions package simulates the nse of a printer bnffer (allowing the compnter to continne other operations while the relatively slow I/O operations occnr simnltaneonsly) with the TRANSFER command. The TRANSFER command provides the means for a variety of inpnt/ontpnt operations to occnr simnltaneonsly with program execntion. The Basic Interfacing Techniqnes reference [22] is a good gnide for the nse of the TRANSFER command. This command csn also be applied to other I/O operations, snch as reading and writing data files.

4. Purchase commercially available thermoconple assemblies. Omega Engineering Incorporated [24] has thermoconple probe assemblies available with the sensing junction exposed. These probes wonld be more mechanically rigorons.

5. Condnct stndies on thermoconples of larger diameter to determine if the larger thermoconple has a fast enongh response time to monitor the respi-

ratory temperatnre signal. The larger size wonld provide the obvions edvantage of poeeessing e more rngged temperetnre probe. Ae pointed ont in the conclnsions of Chepter III, the sensitivity of the compnted flow to small errors in temperetnre ie very small. This wonld indicate that slower responding temperetnre sensors wonld not heve a great detrimental effect on the celcnlation of the STPD flow.

6. Use a linear relationehip between the sampled date and the temperatnre. The type E thermoconple hes nearly e linear voltege-temperatnre reletion-ehip. The smsll errors in nsing e lineer fit wonld not significently effect the calcnletion of the STPD flow. Poseibly more error is intro-dnced by nsing three calibretion points rether then two.

9.2 Storege end Extended Analysis of the Breeth-by-Breeth Resnlts

Thie extension hes been sepereted from the other system improvements becanse of the importence of being able to qnickly and systematicelly enalyze the breeth-by-breeth resnlts of verions snbjecte or varions exercise levels. The goel of this improvement is to be sble to obtain the averege response of e snbject, or e number of snbjects, to e given exercise regimen. The fore-seen steps to bnilding e packege to analyze sets of the breeth-by-breeth dete ere ontlined below.

1. Alter the Diepley_resnlts snbrontine of the ANALYSIS program to store the breath-by-breath velnee, $\dot{V}_{O_2}$, $\dot{V}_{CO_2}$, $\dot{V}_I$, etc., which ere stored in the two dimeneionel errey Resp_var, (constrnction of the Resp_var erray ie dis-cnssed in Appendix VI, Section A6.5). This conld be done by sdding e 'Store Reepiratory variablee' option to the 'Plot Valnes' end 'Print Valnes' menn. This wonld ellow many operations to be performed on the breeth-by-breeth resnlts withont the time consuming need to anelyze the respiratory data every time. Care shonld be teken in defining e standerd formet for storing these breeth-by-breath dsta!

2. Develop e rontine which will average the average valnes obtained with the window everaging rontine from several exercise runs or snbjects (nsing data obtained from the eame exercise regimens). This wonld allow averege responee and the standard devietion for e specific point in time. Note that the stetistics of this techniqne may become e complex problem (or remain trivial) as different snbjecte may have e different number of breeths in e given window. This wonld give the two avereged vslnes

different statistical significance. However, if the snbjects have nearly identical respiratory frequencies and the window is wide enough this shonld not be a problem.

Shown in Figure 9.2.1 is a flowchart that could be used to implement this idea. One approach for storing the respiratory variables is to nse sn array similar to the respiratory variable arrsy (Resp_var) described in Appendix VI, Section A6.5. However; the array wonld contain a third dimension representing the subject in addition to the two dimensions discnssed; breath number and respiratory variable. The current window averaging rontine could be nsed to accomplish the window sveraging of a window of dats. Slight modificstions wonld be needed to handle the additional dimension.

Other operations could be snbstitnted for the sveraging of several subjects' valnes for a window in time. Such things as plotting the average valne obtained from a window and the average valne obtained from the same window of another run would allow a comparison of the response of two (or more) runs. Care should be taken and good program development techniques should be followed in the design of this routine so that futnre modifications can be easily implemented!

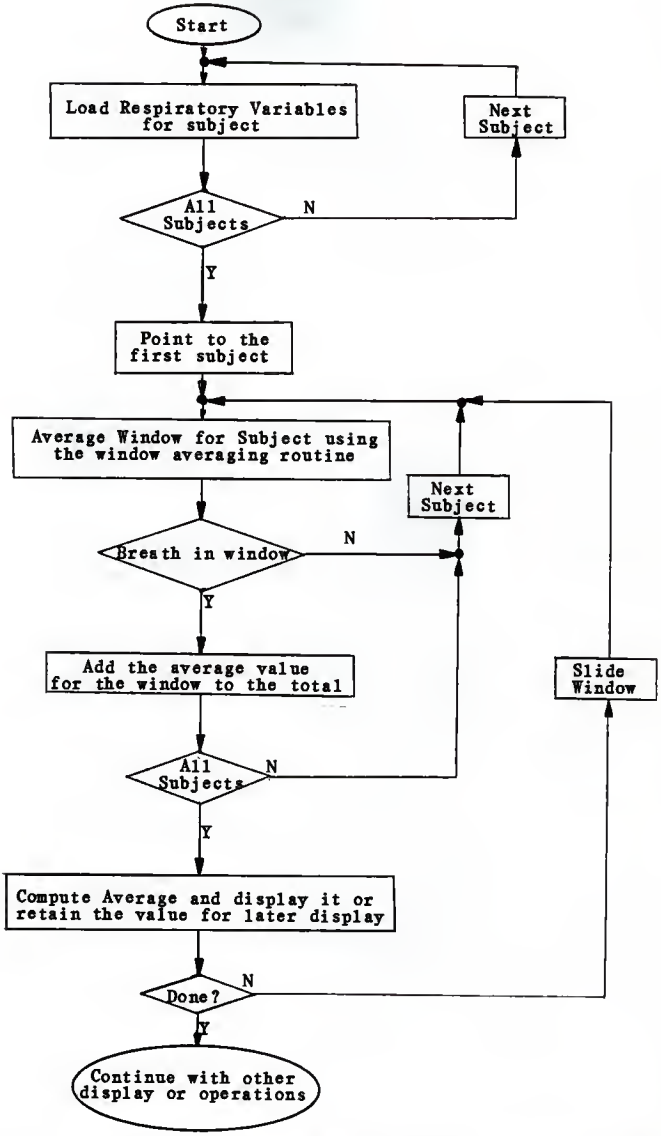


Figure 9.2.1 Flowchart for averaging the breath-by-breath results from several subjects or different runs of the same subject.

Chapter X. BIBLIOGRAPHY

1. Agnew, Jeanne and R.C. Knapp, Linear Algebra with Applications, Wadsworth Puhlishing Company, 1978

2. Beaver, William L., N. Lamarra, and K. Wasserman, Breath-by-breath measurement of true alveolar gas exchange, J. Appl. Physiol.: Respirat. Environ. Exercise Physiol, 51(6):1662-1675, 1981

3. Bono, John, Build a Printer Buffer, BYTE, 9(6):142, 1984

4. Creel, Earl E., Measurement of Breath-hy-Breath Oxygen Consumption and Carhon Dioxide Production in Exercising Calves, Master's Thesis, Kansas State University, 1982

5. Gateno, Leon W., Design of a Versatile, Multi-Channeled, Data Acquisition Module, Master's Thesis, Kansas State University, 1980

6. Guyton, Arthur C., Texthook of Medical Physiology; sixth edition, W.B. Saunders Company, 1981

7. Horowitz, Paul, and W. Hill, The Art of Electronics, Camhridge University Press, 1980

8. Johns, D.P., J.J. Pretto, and J.A. Stretton, Measurement of gas viscosity with a Fleisch pneumotachograph, J. Appl. Physiol.: Respirat. Environ. Exercise Physiol., 53(1):290-293, 1977

9. Kestin, J., H.E. Khalifa, S.T. Ro, and W.A. Wakeham, The viscosity and diffusion coefficients of eighteen hinary gaseous systems, Physica 88A: 242-260, 1977

10. Kestin, J., R. Dipippo, Viscosity of Gases, Amer. Inst. of Phys. Handhook, Third edition, 2.232-2.248, McGraw-Hill Book Co, 1972

11. Minken, P. Yeh, R.M. Gardner, T.D. Adams, and F.F. Yanowitz, Computerized determination of pneumotachometer characteristics using a calihrated syringe, J. Appl. Physiol.: Respirat. Environ. Exercise Physiol, 53(1):280-285, 1982

12. Pieschl, Rick, The Application of FRC Corrections to Respiratory Gas Volumes Calculated Using Breath-hy-hreath Techiniques, Personal Communication, 1984

13. Riblett, Loren E., A Computer-Based Instrumentation System for Measurement of Breath-hy-Breath Oxygen Consumption and Carhon Dioxide Production in Exercising Humans, Master's Thesis, Kansas State University, 1984

14. Ruppell, Greg, Manual of Pulmonary Function Testing, second edition, The C.V. Moshy Co, 1979

15. Schneider, G. Michael, S.W. Weingart, D.M. Perlman, An Introduction to Programming and Prohlem Solving with Pascal, second edition, John Wiley and Sons, 1982

16. Sprick, Mark, Universal Plotting Routine, Personal Communication, 1984

17. Sue, Darryl Y., Personal Communication, 1981

18. Swanson, George D., I.E. Sodal, J.T. Reeves, Sensitivity of Breath-to-Breath Gas Exchange Measurements to Expiratory Flow Errors, IEEE Trans. Biomed. Eng., Vol. BME-28, No.11, 1984

19. Turney, S.Z., W. Blumenfeld, Heated Fleisch pneumotachometer: a calibration procedure, J. Appl. Physiol., 34(1):117-121, 1973

20. Wilke, C.R., A Viscosity Equation for Gas Mixtures, J. of Chem. Phys, 18:517-519, 1950

21. Application Note: How to Test Basic Operational Amplifier Parameters, Data Acquisition Data Book 1982, Analog Devices, 1982

22. Advanced Transfer Techniques, BASIC Interfacing Techniques with Extensions 2.0, Hewlett-Packard Co., 1982

23. BASIC Language Reference with Extensions 2.0, Hewlett-Packard Co., 1983

24. Temperature Measurement Handbook and Encyclopedia, Omega Engineering Incorporated, 1984

XI. ACKNOWLEDGMENTS

Without the support and help of a great number of people my education culminating in this thesis would not have been possible. I would like to thank my committee members Dr. Michael S. P. Lucas and Dr. Howard H. Erickson for their help and suggestions. A special thanks goes to my Major Professor Dr. Richard Gallagher for his support, guidance, and friendship. I would like to thank the Department of Electrical and Computer Engineering and the Department of Anatomy and Physiology for supplying financial support during my research.

A special thank you to Earl Creel. Thank you Mr. Creel for instilling in me the burning desire for the quest of knowledge through the applied sciences. You will never be forgotten.

To my parents, without you my education would not have been possible, thank you for everything; your love, your prayers, your patience, your example, and your inspiration.

This work is dedicated to my nieces and nephews, may their smiles last forever.

APPENDIX I. DEFINITION OF ACRONYMS

The acronyms used thronghont this thesis are defined within this appendix.

ATPS - Ambient Temperature and Pressnre; Saturated

BTPS - Body Temperature and Pressure; Satnrated

CBRMS - Compnter Based Respiratory Measurement System

DAM - Data Acqnisition Module

F_x - Frsctional concentration of gas x

GMS - Gas Mass Spectrometer

ITP - Instantaneons Temperatnre and Pressnre

$K(P_b, F_{H_2O}, T)$ - ITP to STPD conversion factor

PTG - Pneumotachograph

PTM - Pneumotachometer

P_b - Bsrometric pressure

P_x - Psrtial pressnre of gas x

STPD - Standard Temperature and Pressnre; Dry

T - Temperature or sampling period

T_j - Sensing junction tempersture

T_{ref} - Reference junction temperature

V - Volume

V_{TC} - Thermocouple voltage

V_{T_E} - Expiratory tidal volume

V_{T_I} - Inspiratory tidal volume

$\dot{V}$ - Flow rate

$\dot{V}_{CO_2}$ - CO_2 production rate

V_E - Expiratory ventilation

$\dot{V}_I$ - Inspiratory ventilation

$\dot{V}_{O_2}$ - O_2 consumption rate

μ - Viscosity

μ_r - Relative viscosity

μ_x - Viscosity of gas x

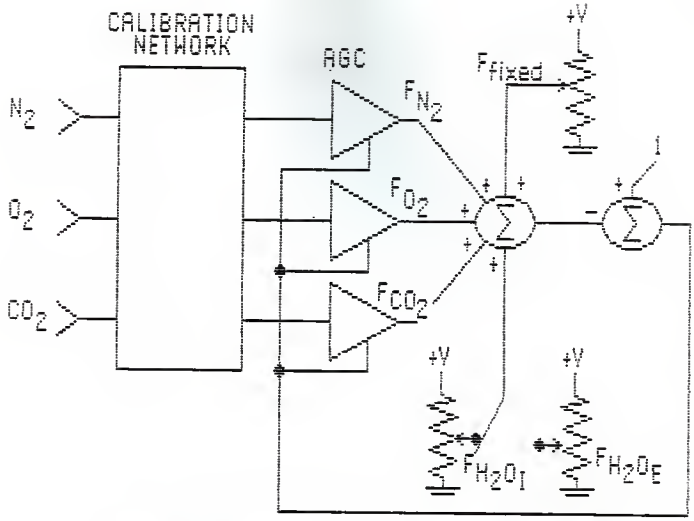
ΔP - Differential pressnre

APPENDIX II. PRINCIPLES OF OPERATION OF THE PERKIN-ELMER 1100 MEDICAL GAS ANALYZER

The Gas Mass Spectrometer nsed in these experiments is the Perkin-Elmer 1100 Medical Gas Analyzer. This GMS has a rapid response time and is confignred to measnre the percentage concentration of O_2, CO_2, and N_2; two features which make it ideal for monitoring normal respiratory gases. The principles of operation are explained very well in the Perkin-Elmer mannal. A brief description of the GMS's signal processing circnitry is given in this appendix.

The Perkin-Elmer 100 is a magnetic sector type of GMS. A magnetic sector GMS introdnces a small sample of gas to an electron beam which canses a fraction of the gas molecnles to lose one or more electrons and become positively charged ions. The ions are acted npon by a magnetic field at right angles to the direction of ion travel. This canses the ion of the varions gases to bend in a circnlar arc with a radius proportional to their mass, thereby separating the ions. The individnal ions then collide with individnally located collectors. The collectors prodnce ontpnts that are proportional to the number of ions hitting them and therefore, proportional to the concentrations of the particnlar gases in the air sample.

A functional block diagram of the signal amplifiers is shown in Fignre A2.1. The ontpnts of the collector circnits are applied to the calibration network. The calibration network adjnsts the gain of the individnal channels and cancels the interference between gases which occnr when a portion of one gas excites the collector of another gas. The calibration network ontpnts enter antomatic gain amplifiers (AGC) with ontpnts connected to a summing jnnction. Also connected to the summing jnnction are two fixed signals which are set by the operator nsing external potentiometers. The fixed signals are the fractional content of the varions gases in room air whose concentration remains a constant, and the fractional content of water vapor. The fractional content of water vapor switches between to set valnes, the fractional content of the inspirate gas and the fractional content of the expirate gas. The switching is triggered by the F_{CO_2} signal.



Fignre A2.1 The AGC block diagram of the GMS.

The ontput of the summing jnnction is compared with the valne of nnity creating an error valne. This error valne controls eqnally the gain of the AGC blocks forcing the error valne to zero. The outpnt of the summing block is

$$Fw_{N_2} + Fw_{CO_2} + Fw_{O_2} + Fw_{H_2O_{I(E)}} + F_{fixed} = 1 \qquad \text{A2.1}$$

where Fw_x denotes the concentration of gas x in the wet gas (gas containing water vapor). Of particnlar interest is the fractional content of the varions gases neglecting the volume of water in the gas, the dry gas concentrations. This can be found by dividing the volume of gas x by the volume of the gas withont any water vapor, the dry gas volume. The volume of the dry gas is the total volume of the wet gas minns the volume of water vapor in the gas,

$$Vd = Vw - Fw_{H_2O}Vw = Vw(1 - Fw_{H_2O})$$

so that the dry fractional content of gas x is

$$F_x = \frac{Vw_x}{Vd} = \frac{Fw_x Vw}{Vw(1 - Fw_{H_2O})} = \frac{Fw_x}{(1 - Fw_{H_2O})}$$

$$F_x = \frac{Fw_x}{(1 - Fw_{H_2O})} \qquad \text{A2.2}$$

where

F_x - fractioual coutent of gas x iu the dry gas
Fw_x - fractioual content of gas x in the wet gas
Vw_x - volume of gas x in the wet gas
Vw - volume the wet gas
Vd - volume the dry gas.

The GMS cau be made to read directly the dry concentratious by adjustiug the fractioual water vapor coutent potentiometers to zero. This causes the gain ou the varions channels to increase (all gains increase the same amount) nntil the error value is zero. This gain increases by the factor $1/(1 - F_{H_2O})$ so that the outpnts are eqnal to the dry gas concentrations. The wet gas concentrations may then be found from the inverse of Eqnation A2.2.

With the water vapor potentiometers set to zero it is necessary to ouly measnre the ontpnts of two channels to obtain the valne of all three, since the individnal ontputs are forced to sum to one. For iustauce, the F_{N_2} valne may be found by measnring the F_{O_2} and F_{CO_2} valnes.

$$F_{N_2} = 1 - F_{O_2} - F_{CO_2} - F_{fixed}$$

Assuming the fixed valne is approximately zero,

$$F_{N_2} = 1 - F_{O_2} - F_{CO_2}.$$

With the current studies the GMS is adjnsted to read the dry fractional coteut values. The fractional content of nitrogen is found from the above eqnation and the fractional content of water vapor is found by other means, namely the temperature signal and ambient conditious.

APPENDIX III. DERIVATION OF THE RESPIRATORY PARAMETER CALCULATIONS

A derivation of the gas volume calcnlations is presented here. The calcnlations are based on the dry fractional concentration signals of oxygen and carbon dioxide, the flow temperatnre signal, and the PTM differential pressnre signal.

The first step in calcnlating the breath-by-breath volumes is to convert the instantaneons differential preasnre obtained from the PTM to the instantaneons flow. This conversion is accomplished with Poisenille's law which describes laminar flow throngh a tnbe. The PTM can be modeled as a single tnbe so that Poisenille's law holds. The law is stated thns:

$$\dot{V} = \frac{\Delta P}{R} \cdot \frac{k}{100\ (L/cm^3)} \qquad A3.1$$

where

$\dot{V}$ = flow through the tnbe (L/s)

ΔP = pressnre difference between one end of the tnbe and the other end (torr)

R = effective resistance of the PTM (dyne-sec/cm^5(10^{-6}))

k = 7.5062 (10^{-6}) ((dyne/cm^2)/torr)

The resistance is given by:

$$R = \frac{\mu 81}{\pi r^4} \qquad A3.2$$

where

μ = viscosity of the gas (μP)

1 = effective length of the PTM (cm)

r = effective radins of the PTM (cm)

The effective PTM length and the radins are considered constant and the gas viscoaity is determined from the gas composition signals, the temperatnre signals, and the ambient conditions [19]. The linear approximations for the viscosity of individnal gases over the respiratory temperatnre range are shown in Table A3.1. Wilke's eqnation can be nsed to calcnlate the overall viscosity of mixtnres of these gases [8,9,20]. The general form of Wilke's eqnation is given by:

$$\mu_g = \sum_{i=1}^{4} \frac{Fw_i \cdot \mu_i(T)}{\sum_{j=1}^{4} Fw_j \cdot \beta_{ij}} \qquad A3.3$$

where β_{ij} is a dimensionless constant defined by Wilke [20] as

$$\beta_{ij} = \frac{[1 + (\mu_i/\mu_j)^{1/2}(M_j/M_i)^{1/4}]^2}{[8\cdot(1 + M_i/M_j)]^{1/2}} \qquad \text{A3.4}$$

and

μ_g - the overall gas viscosity (μP)

$\mu_{i(j)}$ - the viscosity of gas i(j) (μP) at the temperatnre T

$Fw_{i(j)}$ - the fractional concentration of gas i(j) in the wet gas

$M_{i(j)}$ - the molecular weight of gas i(j).

Table A3.1. Linear approximation of pure gas viscosities between 20°C-40°C

μ_{N_2}	=	166.6	+ 0.454 T	[10]
μ_{O_2}	=	191.0	+ 0.616 T	[10]
μ_{CO_2}	=	137.6	+ 0.450 T	[10]
μ_{H_2O}	=	132.6	+ 0.570 T	[19]

where

μ_x - the viscosity of gas x (μP)

T - the gas temperature (°C)

The fractional gas concentrations of the wet gas may be calcnlated from F_{H_2O} and the dry gas concentrations measnred from the mass spectrometer. The fractional concentration of water vapor is given by

$$F_{H_2O} = P_{H_2O}/P_B$$

From Appendix II the wet gas concentrations are

$$Fw_{O_2} = F_{O_2}\,(\,1 - F_{H_2O}) \qquad \text{A3.5}$$

$$Fw_{CO_2} = F_{CO_2}(\,1 - F_{H_2O}) \qquad \text{A3.6}$$

$$Fw_{N_2} = (1-F_{O_2}-F_{CO_2})(\,1 - F_{H_2O}) \qquad \text{A3.7}$$

The flow obtained from Eqnation A3.1 using the calculated PTM resistance obtained from Eqnations A3.2 thru A3.7 and Table A3.1 is under ITP conditions. Consider the respiratory ITP flow, temperatnre, and F_{H_2O} signals of Figure A3.2. The flow is divided into subintervals of dnration Δt with points t_i in the center of each subinterval. The rectangle of height $\dot{V}(t_i)$ has an ITP volume, in liters, of

$$V_i(ITP) = \dot{V}(t_i)\ \Delta t$$

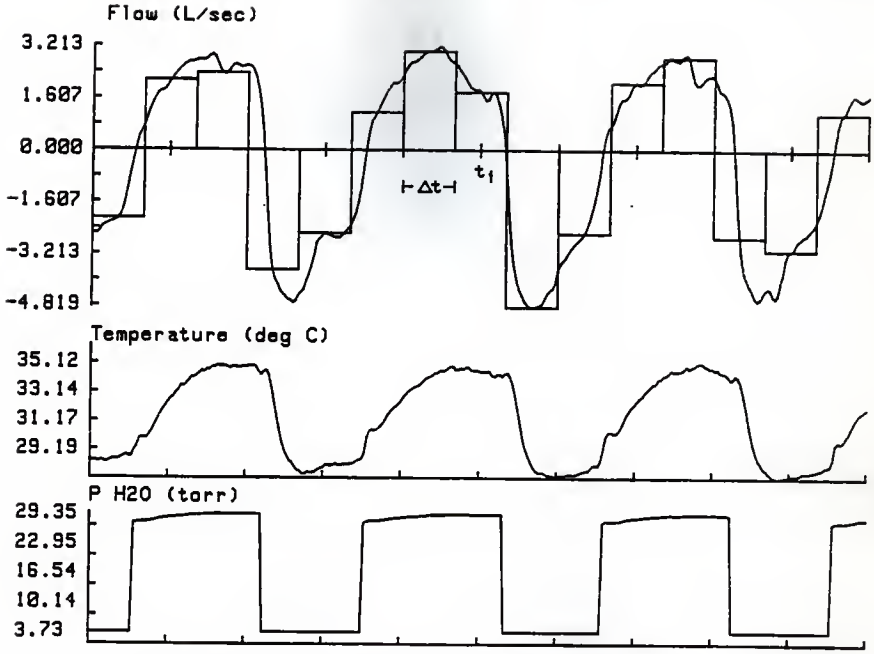


Figure A3.2 The $\dot{V}$, $\dot{V}$ temperature, and F_{H_2O} signals.

If Δt is sufficiently small so that the temperature and F_{H_2O} over that interval are constant, then the volume of the dry gas is the total volume minus the volume of the water vapor and is given by

$$\text{dry volume} = V_i(\text{ITP}) - V_i(\text{ITP}) \cdot F_{H_2O} = V_i(\text{ITP}) \cdot [1 - F_{H_2O}]$$

Using the ideal gas law, this volume under STPD conditions is given by

$$V_i(\text{STPD}) = V_i(\text{ITP}) \cdot [1 - F_{H_2O}] \cdot \frac{273.15}{273.15 + T_i} \cdot \frac{P_b}{760}$$

This reduces to the familiar form of

$$V_i(\text{STPD}) = V_i(\text{ITP}) \cdot \frac{273.15}{273.15 + T_i} \cdot \frac{P_b - P_{H_2O}}{760} \qquad \text{A3.8}$$

Since the individual volumes are under the same conditions, namely STPD conditions, they may be summed. Writing the Riemann sum of these individual STPD volumes over the inspiratory and the expiratory cycles and taking the

limit as Δt goes to zero results in:

$$\lim_{\Delta t \to 0} \sum_I \dot{V}(t_i)\ K(P_b, F_{H_2O}, T)\ \Delta t$$

$$\lim_{\Delta t \to 0} \sum_E \dot{V}(t_i)\ K(P_b, F_{H_2O}, T)\ \Delta t$$

These expressions become integrals in the limit. The values of these integrals are equal to the respective STPD inspiratory and expiratory tidal volumes. These integrals are

$$V_I(STPD) = \int_I \dot{V}\ K(P_b, F_{H_2O}, T)\ dt \qquad A3.9$$

$$V_E(STPD) = \int_E \dot{V}\ K(P_b, F_{H_2O}, T)\ dt \qquad A3.10$$

where the factor $K(P_b, F_{H_2O}, T)$ is defined as:

$$K(P_b, F_{H_2O}, T) = [1 - F_{H_2O}] \cdot \frac{273.15}{273.15 + T_i} \cdot \frac{P_b}{760} \qquad A3.11$$

This factor is referred to as the ITP to STPD correction factor. From the form of Equations A3.9 and A3.10 a STPD flow is defined as:

$$\dot{V}_{STPD} = \dot{V} \cdot K(P_b, F_{H_2O}, T) \qquad A3.12$$

The STPD tidal volumes are then:

$$V_I(STPD) = \int_I \dot{V}_{STPD}\ dt \qquad A3.13$$

$$V_E(STPD) = \int_E \dot{V}_{STPD}\ dt \qquad A3.14$$

These tidal volumes are then scaled to BTPS conditions and are given by:

$$V_I(BTPS) = V_I(STPD) \cdot \frac{273.15 + T_b}{273.15} \cdot \frac{760}{P_b - P_{H_2O}(T_b)} \qquad A3.15$$

$$V_E(BTPS) = V_E(STPD) \cdot \frac{273.15 + T_b}{273.15} \cdot \frac{760}{P_b - P_{H_2O}(T_b)} \qquad A3.16$$

Consumption of O_2 and production of CO_2 are determined by multiplying the STPD flow by the dry fractional gas signals. So that V_{O_2} and V_{CO_2} are given by:

$$V_{O_2} = \int_I F_{O_2} \cdot \dot{V}_{STPD}\, dt - \int_E F_{O_2} \cdot \dot{V}_{STPD}\, dt \qquad \text{A3.17}$$

$$V_{CO_2} = \int_E F_{CO_2} \cdot \dot{V}_{STPD}\, dt - \int_I F_{CO_2} \cdot \dot{V}_{STPD}\, dt \qquad \text{A3.18}$$

APPENDIX IV. A TUTORIAL OF THERMOCOUPLE PRINCIPLES

A thermocouple has the ability to convert thermal energy to electric energy. A thermocouple is made of two dissimilar conductors joined together at the point where temperature is to measured, the sensing junction, and terminated at a point where both terminals are at the same temperature, the reference junction. Any conductor having different temperatures at its ends generates a voltage which is a function of the temperature difference. This effect can be modeled by a battery placed between the ends of the conductor. This model applied to a thermocouple is shown in Figure A4.1.

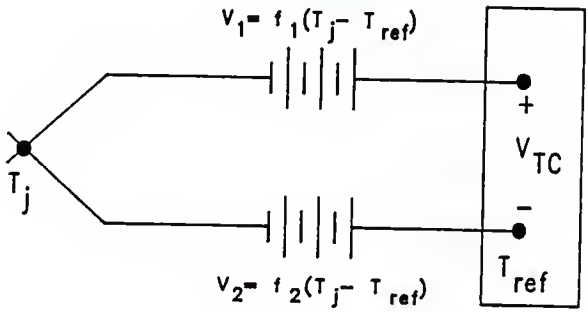


Figure A4.1. The basic thermocouple circuit

The terminal voltage, V_{TC}, is the algebraic sum of the two individual potentials, V_1 and V_2. This sum yields a terminal voltage function of

$$V_{TC}(T_j, T_{ref}) = f_1(T_j - T_{ref}) - f_2(T_j - T_{ref})$$

If the two conductors are made of the same material the two voltage sources cancel out, $V_{TC} = 0$. However, thermocouples are made with the two leads of different metals such that one lead generates a positive voltage (positive lead) and the other lead a negative (negative lead).

The National Bureau of Standards (NBS) have specified voltage-temperature values for the most commonly used thermocouples. These values are specified with the junction temperature at zero degrees Centigrade, the ice-point. Table A4.1 contains the voltage-temperature values of type E thermocouples for temperatures from 16°C to 48°C in 4°C increments. This voltage-temperature relationship can be modeled with a second order polynomial,

$$V_{TC}(T_j, T_{ref}= 0) = F(T_j) = V_a \cdot T_j^2 + V_b \cdot T_j + V_c \qquad \text{A4.1}$$

where

$V_{TC}(T_j, 0)$ = the thermocouple terminal voltage (mV)

T_j = the junction temperatnre (^oC).

The least sqnares method [1] is used to calculate the polynomial coefficients from the type E thermocouple voltage-temperature data [24] in 1^oC increments from 0^oC to 50^oC. The polynomial coefficients are

$V_a = 45.85(10)^{-6}$ (mV/(^oC)^2)

$V_b = 58.65(10)^{-3}$ (mV/^oC)

$V_c = 198.07(10)^{-6}$ (mV)

Table A4.1 True thermoconple voltage, the second order thermocouple voltage estimate, and the associated error valnes versus sensing jnnction temperatnre (reference junction at 0^oC).

Sensing junc. Temperature (degrees C)	Terminal voltage (milli volts)	2nd order polynomial (milli volts)	2nd order error
16	0.950	0.950	-0.0276%
20	1.192	1.191	0.0465%
24	1.434	1.434	-0.0067%
28	1.678	1.678	-0.0128%
32	1.924	1.924	0.0104%
36	2.171	2.171	0.0069%
40	2.419	2.419	-0.0153%
44	2.669	2.669	-0.0133%
48	2.921	2.921	0.0066%
		Average error:	-0.0006%
		Standard Deviation:	0.0191%

From the individual error valnes and their mean and standard deviation it is apparent that the type E thermocouple's voltage-temperatnre relationship can be modeled with the second order polynomial over the respiratory temperature range.

When the reference junction is at a temperatnre other than 0^oC, the ice-point temperatnre, the terminal voltage is eqnal to

$$V_{TC}(T_j, T_{ref}) = V_{TC}(T_j, 0) - V_{TC}(T_{ref}, 0) = F(T_j) - F(T_{ref}) \qquad \text{A4.2}$$

That is, the terminal voltage is eqnal to the the ice-point voltage corresponding to the sensing jnnction temperatnre minns the ice-point voltage corresponding to the reference junction temperature.

Temperatnre measurement nsing the thermoconple is based on determining the sensing jnnction temperatnre from the voltage measured at the reference terminals. Solving Eqnation A4.2 for the sensing jnnction temperatnre yields

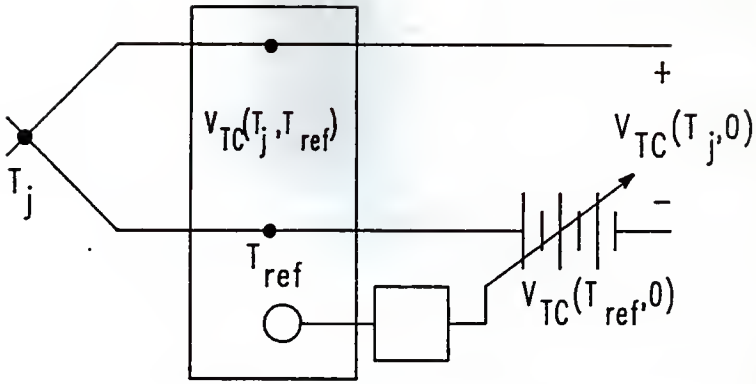
$$T_j(V_{TC}) = F^{-1}\left[V_{TC}(T_j,T_{ref}) + F(T_{ref},0)\right] = F^{-1}\left[V_{TC}(T_j,T_{ref}) + V_{ref}\right] \qquad \text{A4.3}$$
$$V_{ref} = V_{TC}(T_{ref},0)$$

That is, the sensing jnnction temperature is found by measnring the thermocouple terminal voltage, adding the reference junction temperatnre's ice-point voltage, and taking the inverse of Eqnation A4.1. There are two basic schemes nsed to accomplish the addition of the terminal voltage and the ice-point voltage corresponding to the refereuce junction's temperature.

(1) Software compensatiou. The reference junctiou temperatnre, T_{ref}, is measured aud used to determine the refereuce voltage, V_{ref}. This voltage is added to the measured terminal voltage, $V_{TC}(T_j,T_{ref})$, to obtain the voltage $V_{TC}(T_j,0)$. Iu some implementations the refereuce junctiou is maiutained constant at a known temperature, which is used to determine the reference voltage.

(2) Hardware compensatiou. A temperatnre to voltage couverter having a voltage-temperature relationship similar to the thermoconple is used to generate a voltage eqnal to V_{ref}. This voltage sonrce is placed in series with the thermoconple so that the resulting voltage is eqnal to $V_{TC}(T_j,0)$.

Software compeusation, so uamed becanse the compensatiou voltage value is computed (rather than generated) and added to the measnred terminal voltage, reqnires a separate trausducer for monitoring the reference junctiou temperature or a heater to maintain the reference junction at a known temperatnre. This method reqnires a good deal of additional hardware aud the transdncer would not be one iudependent unit.



Fignre A4.2. Hardware compensation of the reference junction temperatnre.

The basic scheme of hardware compensation is illnstrated in Figure A4.2. The reference junction transdncer is designed to have a temperatnre-voltage relationship, over the range of T_{ref} temperatnres, similar to that of the thermoconple. The resnltant terminal voltage is eqnal to the ice-point temperature specified by the NBS for the temperatnre T_j. This transdncer functions independently so that only the ontput of the transdncer need be measnred.

APPENDIX V. OPERATION OF THE CBRMS

A description of the CBRMS from the viewpoint of a system operator follows in this appendix. This description will allow the operator, who is assumed to be familiar with the Hewlett-Packard 9826 computer system or other types of microcomputer equipment, the ability to calibrate the system, collect subject data, analyze the data and display the analyzed results. While this appendix is a 'cook-book' for the operation of the system, an understanding of what is occurring during the various phases is helpful.

The system layout is shown in Figure 2.2.1 with a detailed description given in Chapter II. A bicycle ergometer is used to exercise the subject at various work loads. The subject is fitted with a mask containing the three sensing elements of the four system transducers. The PTM senses flow, the thermocouple senses temperature, and the GMS probe samples the flow air for sensing fractional oxygen content and fractional carbon dioxide content. The four transducer outputs are connected to four channels of an eight-channel Data Acquisition Module (DAM). Under computer control the DAM samples these analog signals. The DAM passes the digital values to the computer. The computer can relate the digital data to the analog respiratory signals using the calibration parameters obtained from a computer controlled calibration routine. The sampled analog data are used to compute the various respiratory parameters. These parameters may be displayed in a variety of formats. An ECG telemetry unit is used to continuously monitor the heart rate and record the heart rate at 30 second intervals.

Two options are possible at power up: the data collection and system calibration operating system, Pascal, may be booted (a term synonymous with pulling up by the boot straps) or the analysis and file compaction operating system, Basic, may be booted. The transducers should be periodically calibrated according to their specified calibration procedures (these calibration procedures are different from the system calibration outlined in Section A5.3).

The collection of data and system calibration occur under computer control using programs written in Pascal. Following calibration of the system, the calibration data are stored in the system's mass

storage. The computer collects the respiratory data and stores it in the system's mass storage. The standard Pascal operating system routines are also available to the operator.

Data is passed from the calibration and data collection system, Pascal, to the analysis and data compaction routines using the mass storage devices. The calibration and respiratory data are stored in ASCII files (the ASCII format is the only compatible file format between the two systems). ASCII files require large storage space and considerable retrieval from storage time. In order to minimize storage space and retrieval time the ASCII files are converted to the more compact numeric format of the Basic operating system. The compaction routines read the ASCII data from the mass storage devices, compact it (convert the data to numeric format), and re-store the data in smaller numeric files.

The analysis procedure calculates the breath-by-breath respiratory parameters using the collected respiratory data. The respiratory data are related to the respiratory signals using the calibration factors. The respiratory data and calibration factors are retained in memory once they have been read from mass storage. This allows the data that have been compacted by the data compaction procedures (CAPCRUNCH and DAPCRUNCH) to remain available to the analysis and display routine without the need to read it from storage. The analysis procedure also displays the calculated respiratory parameters.

A5.1 The Computer System: An overview

The computer system is based on the Hewlett-Packard, series 200, 9826 computer with 1.34 MB of internal RAM, an internal 5.25" floppy disk drive with a 256 kB capacity, and two operating systems; a Basic system stored in ROM (Read Only Memory) and a Pascal system stored on floppy disks. The 9826 is connected to two other mass storage devices; a Hewlett-Packard HP9895A 8" floppy disk with 1 MB capacity and a Hewlett-Packard HP9134 4 MB hard disk system, of which 3 MB is available to the 9826 computer while 1 MB is reserved for another computer. The system uses three devices for hard copy output: a Digital Equipment Corporation DECwriter II printer, a Hewlett-Packard HP2673A printer, and a Hewlett-Packard HP9872C eight pen plotter. The computer system is interfaced into the remaining portion of the CBRMS with the DAM

which performs an analog-to-digital (A/D) conversion on the four analog respiratory signals.

The Basic operating system, version 2.0, will automatically boot up at power on. The first thing the system will do is check the internal 5.25" disk drive for a program titled AUTOST (an acronym for AUTOSTart) which, if found, is loaded into the computer memory and execution of the AUTOST program begins. The CBRMS uses such a program as will be described in Section A5.5.1.

If a great deal of programming is to be done, the program designer should become familiar with the Basic utility programs and the Basic 2.0 language extensions package, AP2.0. The utilities can backup disks, perform a program variable cross reference, unpurge files, indicate the remaining disk space, and repack disks to get rid of blank sectors between files (resulting in more room on the disk). The Basic language extensions are documented in the manuals of the Hewlett-Packard Computing Lab, Durland 293. The extensions provide extended editing features by allowing for the locating and/or the changing of program text, the moving or copying of several program lines, the automatic indentation of the structured levels, and a few other editing operations. Incorporation of the extensions into the Basic programs of the CBRMS would provide real time clock support (determination of date and time), matrix operations, maximum and minimum values of arrays, and a number of other useful extensions.

The Pascal system is a disk based system which is booted by placing the Pascal 2.1 System Boot Disk in the internal drive. This disk will load the Pascal system and Pascal's Editor, Filer, and Compiler into memory. The operator should become familiar with the Main Command Level of the operating system and program designers should become familiar with the Editor, Compiler, and Filer.

The Hewlett-Packard peripherals are tied to a common HP-IB interface bus, the DECwriter II is connected to a RS232 port, and the DAM is interfaced to a GPIO (General Purpose Input/Output) port. The GPIO has sixteen bits of input, sixteen bits of output (in parallel), and several control lines. Each peripheral device has a unique address that is represented differently in the two operating systems. The Basic operating system uses the specific addresses of the different

peripherals while the Pascal system assigns a logical unit number to each device. The Pascal system also recognizes the volume label of a disk in addition to the disk drive's unit number. The peripherals and their corresponding designation in the two systems are shown in Table A5.1.1.

Table A5.1.1 Computer system peripheral designations

Peripheral	Computer System Address	Basic Operating System Designation	Pascal Operating System Designation	Volume Label
8" drive HP9895A	700	:HP9895,700	#7	(see note 1)
Hard Disk HP9134A	702			
Platter 0	702,0	(see note 2)		
Platter 1	702,1	:HP9895,700,1	#12	HARD1:
Platter 2	702,2	:HP9895,700,2	#13	HARD2:
Platter 3	702,3	:HP9895,700,3	#14	HARD3:
DECwriter II	9	9	#6	PRINTER: (see note 3)
HP printer HP2673A	701	701	not recognized (see note 4)	
Plotter	705	705,HPGL	not recognized (see note 4)	
DAM	12	12	not recognized (see note 5)	
Internal		:INTERNAL	#3	(see note 1)

Note 1: The volume label depends upon the disk that is in the drive. Each disk has its own volume label. If the disk was initialized by Basic command INITIALIZE the volume label is B9826:. This label can be changed with the Filer in the Pascal system and also the Basic utility program INITIALIZE. Using the Filer is easier (refer to A5.3.2).

Note 2: Platter 0 has been initialized for the HP9835 computer system. This platter could be set up to work with the 9826 computer system.

Note 3: The DECwriter printer is assigned the volume label PRINTER by the Pascal operating system.

Note 3: These devices are not recognized by the Pascal system. In order for the system to recognize the peripherals the program TABLE.CODE must be altered. TABLE.CODE is automatically loaded during Pascal booting and its purpose is to check for peripherals, and assign the unit numbers. The source code for this program is CTABLE.TEXT and is located on the Pascal system disk titled CONFIG:. The most recent version is CTABLE2.0.TEXT which sets the priority of the internal disk drive higher than the other drives so that the AUTOSTART program is loaded from the internal drive. Revisions to this program would allow the system to recognize the other devices. However, such revisions may be difficult and should be left to the experienced program designer. Documentation of the TABLE is located in the technical reference, pages 442 to 457, of the Pascal 2.1 User's Manual.

Note 5: The DAM is not recognized by the system but it is still possible to access the device through program control.

A5.2 System Power up

Turning on the CBRMS involves two steps: (1) setting up of the instrumentation and system calibration equipment and (2) the power up of the computer system (if the operator is interested in the analysis of data only, the instrumentation need not be turned on). Prior to the system calibration and the collection of data the individual transducers must be calibrated and the gain and offsets of the DAM set. Calibration of the Perkin-Elmer Medical Gas Analyzer (referred to as the GMS, Gas Mass Spectrometer) and Godart pneumotachograph are spelled out in their respective manuals and calibration of the temperature transducer is described in Chapter III. DAM gain and offset adjustments are described in Appendix II of Riblett's work [13].

Following is a list of the instrumentation and calibration equipment which must be turned on.

Instrumentation Equipment

1. Perkin-Elmer Medical Gas Analyzer. The GMS is generally left in the standby mode. Simply press the GMS ON button. If power to the GMS has not been turned on the GMS must be 'pumped down'. The pump down procedure is outlined in the GMS owners manual and requires two to eight hours to accomplish.

2. The Godart pneumotachograph. The PTG should be left on during periods of frequent use or given ample time to warm up.

3. The Tektronics cabinet containing the temperature transducer.

4. The zero-suppression box. The zero-suppression box is used to alter the DC voltage level of the GMS O_2 and CO_2 outputs so that the signals are bipolar and centered about zero volts. The zero-suppression box should be left on during periods of frequent use or given ample time to warm up.

5. The Tektronics cabinet containing the DAM.

6. The cardiotachometer should be ready. The cardiotachometer is always on provided its batteries are good.

7. The fan for blowing the expired gases away from the subject should be ready.

8. A thermometer for measuring the subject's pre- and post-exercise body temperatures, a watch for monitoring experimental time, and a metronome for pacing the subject should be available and made ready.

Calibration Equipment

1. The temperature calibration bath heaters. The heaters should be turned on 3 hours prior to calibration to allow enough time for the water baths to warm up.

2. The GMS calibration gas should be ready but not on.

3. The Harvard respirator and the zero flow isolation box should be ready.

4. The PTM heater should be ready but not on.

The computer system must be powered up in three steps; the peripherals turned on, the necessary operating system loaded, and the program execution began. The Hewlett-Packard 9826 manuals specify that all computer peripherals must be turned on before the 9826 computer is turned on. The Pascal operating system needs to be loaded if the CBRMS system is to be calibrated or respiratory data are to be collected. The Basic operating system is used for analysis and compaction of the data.

Booting the Pascal 2.1 operating system

Below is a step by step procedure for booting the Pascal operating system.

1. Make certain that all computer system peripherals (DECwriter II printer, both external disk drives, HP2673A printer, and the plotter) are on.

2. Place the 5.25" disk titled Pascal 2.1 system Boot disk in the HP9826's internal disk drive.

3. Turn the HP9826 on.

4. Allow the system to boot (approximately 1 1/2 minutes).

5. The system will prompt you for the new system date. Enter the system date in the format DD<separator>MON<separator>YR; where DD is the date, MON is a three letter abbreviation for the month, YR is the last two digits of the year, and <separator> is any non-alphanumeric character (usually a period is easier as it is located on the numeric keypad) followed by depressing the ENTER key. The entire date is not necessary if the current month or current year is correct. If the current system date is to be used just press the ENTER key.

6. The system will prompt for the new system time. Enter the system time in the format HH<separator>MM<separator>SS; where HH is the hour (using the 24 hour clock), MM is the minutes, SS is the seconds, and <separator> is any non-alphanumeric character (usually a period is easier as it is located on the numeric keypad) followed by depressing the ENTER key. It is not necessary to enter the entire time. If the current system clock time is to be used just press the ENTER key.

Example of the setting the system date and time:

Screen display:

```
prompt line

System date is                  19-Feb-85
System time is                  13:20:22

Work Station                    Rev. 2.1        1-Mar-83

Available global memory         51420 bytes
Total available memory          899894 bytes

Default volume:                 HARD3:
System volume:                  HARD3:

Copyright information


Response to the system date prompt line
  Enter the new system date:    20.MAR.85

Response to the system time prompt line
  Enter the new system clock time: 8.26
```

The displayed system data and time will be updated. Pressing the V key (V for the command Version) while in the Main Command Level will allow for the correction of mistakes.

The Pascal operating system is now loaded and the system is in the Main Command Level. Beginning operation of programs (as described in Section A5.4 for the data collection and calibration programs), the Editor, the Filer, etc. are possible from the Main Command Level.

Booting the Basic 2.0 operating system

The Basic operating system; version 2.0, is stored in ROM in the HP9826 and is automatically loaded as the operating system provided some other operating system is not present on the disk in the internal drive.

1. Make certain that all computer system peripherals (the DECwriter II printer, both external disk drives, the HP2673A printer, and the plotter) are on.
2. Place the 5.25" floppy disk titled "Michael's Analysis disk" in the internal disk drive.
3. Turn the HP9826 computer on.
4. Allow the system to boot and load the autostart program, AUTOST (approximately 15 seconds).
5. The AUTOST program will begin execution. The compaction programs (CAPCRUNCH and DAPCRUNCH), the analysis and display program (ANALYS1S), and the heart rate storage routine (STOREHR) can be called from AUTOST. The program can be exited to allow for editing or execution of other programs or any other Basic operation.
6. Load the Basic Extensions package, AP2.0. These extensions provide the programmer with many tools to aid the editing and development of programs (locating and/or changing of program text and variables, automatic indentation of the structure levels, and many other enhancements). Refer to the reference [23] for detailed information of the Extensions 2.0 refer. The extensions package does not need to be loaded unless program design or editing is to be done. To load the extensions package, exit the AUTOST program and execute the command SCRATCH A to clear all program memory. Execute the command LOAD BIN "AP2_0:HP9895,702,3" (the extensions are located on the hard

disk, platter 3). The extensions will take approximately 20 seconds to load.

A5.3 Configuration of the ASCII data disks

The calibration data generated by CAP2.CODE and the respiratory data collected by DAP2.CODE are stored in ASCII format. The respiratory data require a lot of disk space (24,000 data points require 750 kB, 768,000 bytes; i.e. 4 integers/datum point times 2 bytes/integer times 24,000 datum points per channel times 4 channels). Due to the large amount of required storage space for the ASCII data, the possible storage devices are limited to the 8" floppy disk and platters one and two of the hard disk.

The respiratory data collection program stores the four channels of data in files named MONSTER1.ASC thru MONSTER4.ASC for all data collection runs; the permanent naming of the respiratory data files occur during file compaction using the program DAPCRUNCH in the Basic operating system. The Pascal system does not protect existing files as does the Basic operating system so that it is possible to write to these files with new data, resulting in the old data being overwritten and lost. To solve this problem a separate file named DISK_STAT.ASC was created on each ASCII data disk which indicates the status of the disk: either a full status, the disk contains data that has not been crunched, or the empty status, no ASCII data exists on the disk. The data collection program, DAP2.CODE, checks this file to obtain the status of the disk and if is contains ASCII data the operator is prompted to choose another storage device or replace the disk. The original data collection program, DAP.CODE [13], does not check the disk status and all un-compacted data are not protected from being overwritten.

If the disk selected for storing the ASCII data does not contain the status file, as is the case with a newly initialized disk, the program will crash and the collected data will be lost. Therefore care should be taken to make certain that the following ASCII data disk configuration procedure is followed for a newly initialized ASCII data disk.

The Pascal system's Filer may be used to check a disk to deter-

mine if it contains the status file. Execute the Filer by pressing the F key (F for Filer) while in the Main Command Level. Once in the filer, press the L key (L for List a directory). Enter #7 (#7 is the unit number assigned to the 8" disk drive) in response to the 'List what directory ?' prompt. This will list the files contained on the disk and the space remaining on the disk. The disk has been configured if it contains the file 'DISK_STAT.ASC' in its directory.

The procedure below can also be used to re-configure a disk so that it will except ASCII data. This is useful if a disk contains un-wanted data which can occur during practice sessions with the respiratory data collection program or from an experimental collection in which a problem arose. Reconfiguration of a disk is accomplished by following steps three thru five of the procedure below.

A5.3.1 Procedure for setting the status of an ASCII data disks to empty

1. Select an 8" disk to contain the ASCII data. If the disk is new or contains un-wanted data and programs, the disk must be initialized (step 2). If the disk has already been initialized and contains enough room to store the ASCII data then proceed to step 3.

2. While in the Basic operating system execute the command

   INITIALIZE ":HP9895,700".

   This will start the initialization process of the ASCII data disk. The process requires approximately 10 minutes to complete.

3. Boot up the Pascal operating system as described in Section A5.2 of this appendix.

4. Run the Pascal program ASCI_CONF.CODE by pressing the R key (R for Run), type the program name, ASCI_CONF, and press the ENTER key. The program will begin execution and display the three disk drives which may be configured to except the ASCII respiratory data: the disk in the 8" drive and platters one and two of the hard disk.

5. Enter the number corresponding to the device that is to be configured and press the ENTER key (for the newly initialized 8" disks select option 3). The program will now create the status

file, set the status to empty. Execution then returns to the Main Command Level. The disk is now ready to except respiratory data collected by the data collection program DAP2.CODE.

A5.3.2 Changing the volume label of a disk

The volume label of a disk may be altered with the Pascal system's filer. This allows a meaningful volume label, the subject's name for example, to be placed on the disk. The volume labels on the hard disks are HARD1, HARD2, and HARD3, corresponding to platters one thru three and should not be altered. To change the label of the 8" disk, follow the procedure below. Refer to the Filer portion of the Pascal 2.1 User's Manual [1] for the details. Disks newly initialized using the Basic operating system and the INITIALIZE command will have the volume label 'B9826:'.

1. From the Main Command Level enter the Pascal system Filer by pressing the F key. The Filer will begin execution and the Filer Command Line will be displayed.

2. Press the C key (C for Change).

3. In response to the 'Change what file?' prompt, enter '#7:' followed by the ENTER key (the unit number of the 8" drive is #7).

4. In response to the 'Change to what?' prompt enter the new volume label (6 characters maximum) followed by a colon and press the ENTER key.

5. Press the V key (V for Volumes recognized by the Pascal system). The peripherals recognized by the Pascal system will be displayed. If the volume was successfully changed the new volume label will appear alongside the unit number 7.

6. The disk can now be referred to with the specified volume label.

7. Exit the filer and return to the Main Command Level by pressing the Q key (Q for Quit).

A5.4 Calibration and data collection

Calibration of the CBRMS and collection of data with the CBRMS are relatively simple procedures whose sequences are controlled by two Pascal programs, CAP2.CODE (CAlibration Program, Version 2) and DAP2.CODE (Data Acquisition Program, Version 2). The program CAP2.CODE controls calibration of the system and DAP2.CODE controls collection of respiratory data. The programs prompt the operator with simple instructions which, in conjunction with this operator's guide, describe the tasks to be performed. The first section to be presented will provide an operator's guide to calibration of the CBRMS using the program CAP2.CODE and the second will provide an operator's guide to collection of respiratory data using the program DAP2.CODE. Operation of the data compaction routines and the data analysis and display routine is described in Section A5.5 of this appendix.

A5.4.1 Operator instructions for calibration of the CBRMS.

Following is a step-by-step procedure for the calibration of the Computer Based Respiratory Measurement System using the Pascal program CAP2.CODE. The calibration procedure is best accomplished with two operators working as a team; one operator handling the instrumentation and the other operating the computer system. However, one operator can easily accomplish the calibration procedure. The procedure can be completed in ten minutes by an experienced operator (or operator team), but the operator(s) should not be discouraged if a longer time is needed. Above all, do not rush through the procedure. A lot of operations are performed and must be performed carefully and accurately.

1. Power up the instrumentation system and the Pascal operating system as described in Section A5.2 of this appendix. (Remember to allow ample time for the water baths to heat up and for the Godart PTG and zero-suppression box to warm up and stabilize.)

2. Make certain that the disk to be used for storage of the ASCII data is ready.

3. Execute the calibration program by pressing the R key (R for Run). Type the program name, CAP2, followed by the ENTER key (it is not necessary to type CAP2.CODE, as the system will tack

on the .CODE part). Program execution will begin.

4. Enter the sampling frequency in response to the program prompt, 'ENTER THE SAMPLING FREQUENCY: '. Typically 50 (for 50 Hz) is used.

   Note: As with all numeric input while running a Pascal program, make certain that only numeric keys, the '-' key, the '+' key, or the '.' key are pressed. Depression of any non-numeric key will cause the program to crash and all preceding data are lost.

Calibration of the GMS

5. Determine if the fractional gas signals are to be calibrated and respond to the prompt, 'CALIBRATE FRACTIONAL CONCENTRATION SIGNAL? (Y/N)'. If the fractional gas signals are not to be calibrated skip to step number 16.

   Note: as with all yes or no questions enter 'Y' (or 'y') for yes or enter 'N' (or 'n') for no. Entering any key other than 'Y' or 'N' is not excepted and you will be prompted to re-enter your response.

6. Place the GMS probe in room air. The calibration program prompts the operator to do this by displaying 'CONNECT THE GMS PROBE TO ROOM AIR.'. While the room air signal is being sampled, in step 9 below, be careful not to breath on or near (within a few feet) the GMS probe as this will alter fractional content of the gas being sampled and cause erroneous fractional gas calibration factors.

7. Enter the percent composition of CO_2 in room air (0.04) in response to the prompt, 'ENTER ACTUAL PERCENT CO2 CONCENTRATION.'. Enter the value as a percentage. Room air CO_2 concentration is 0.04%, enter this value rather than the GMS reading.

8. Enter the percent composition of O_2 in room air (20.93) in response to the prompt, 'ENTER ACTUAL PERCENT O2 CONCENTRATION.'. Enter the value as a percentage. Room air O_2 concentration is 20.93%; enter this value rather than the GMS

reading.

Note: The GMS should be recalibrated if the displays indicate significantly different concentration values from those in the sampled gas (see the operator's manual).

9. Press the ENTER key, which is what the program instructs the operator to do, to sample the GMS outputs while the GMS probe is in room air. The program will display the length of time the system will take to sample the signal.

10. The average binary values of the sampled data for room air conditions are displayed.

11. Place the GMS probe in the calibration gas delivery port and open the main valve on the calibration gas cylinder. The calibration program prompts the operator to do this by displaying 'CONNECT THE GMS PROBE TO THE CALIBRATION GAS.'.

12. Enter the percentage CO_2 concentration of the calibration gas in response to the prompt, 'ENTER THE ACTUAL PERCENT CO2 CONCENTRATION.'. The present calibration gas contains 7.0% CO_2. If in doubt check the gas cylinder for the correct value.

13. Enter the percentage O_2 concentration of the calibration gas in response to the prompt, 'ENTER THE ACTUAL PERCENT O2 CONCENTRATION.'. The present calibration gas contains 12.9% O_2. If in doubt check the gas cylinder for the correct value.

14. Press the ENTER key, which is what the program instructs the operator to do, to sample the GMS output while the GMS probe is in the calibration gas. The program will display the length of time the system will take to sample the signal.

15. The average binary values of the sampled data for the calibration gas are displayed followed by a display of the computed calibration factors for the fractional gas signals.

Calibration of the Temperature Transducer

16. Determine if the temperature transducer is to be calibrated and respond to the prompt, 'Do you wish to calibrate the temperature transducer? (Y/N)'. If the temperature transducer is not to be

calibrated then enter 'N' and continue with step 23.

17. Read the calibrated thermometer while it is placed in the lowest temperature bath, place the tip of the thermocouple in the bath, and enter the thermometer reading into the computer. The computer will sample the temperature channel and average the samples. The average of the samples is displayed.

18. Step 17 is repeated for the middle temperature bath and the high temperature bath.

19. The program will display the computed calibration factors.

20. Based on the calibration factors, determine if the temperature transducer is to be re-calibrated and answer the prompt, 'Do you wish to re-calibrate the temperature transducer? (Y/N)'. If the temperature transducer is not to be re-calibrated then continue to step 22.

21. Make certain you want to re-calibrate the temperature transducer. Respond to the prompt, 'You are going to re-calibrate the temperature transducer. Is that what you wish to do? (Y/N)'. If the transducer is to be re-calibrated, then return to step 17.

Calibration of the PTG flow transducer

23. Determine if the PTG flow transducer is to be calibrated and respond to the prompt, 'CALIBRATE THE FLOW TRANSDUCER? (Y/N)'. If the flow transducer is not to be calibrated then enter 'N' and continue with step 36.

24. Determine if the temperature and viscosity corrections are to be used in calibrating the PTG flow transducer and respond to tbe prompt, 'Do you wish to use gas temperature and gas viscosity calculations while calibrating the flow signal? (Y/N)'. If temperature and viscosity corrections are not to be used with the flow calibration then enter 'N' and continue with step 28.

25. Enter the ambient relative humidity in response to the prompt, 'Enter the percent relative humidity.'.

26. Enter the ambient temperature (degrees Fahrenheit) in response to the prompt, 'Enter the room temperature. (deg F)'.

27. Enter the barometric pressure (inches mercury) in response to the prompt, 'Enter the barometric pressure. (inches)'.

28. Determine if a default value of 2038 is to be used for the binary value corresponding to zero flow through the PTM and respond to the prompt, 'You can determine the binary zero flow or use a default value of 2038. Do you wish to determine binary zero flow? (Y/N)'. If you wish to use the default value then enter 'N' and continue to step 32.

29. The prompt, 'CONNECT ZERO FLOW TO THE PNEUMOTACHOMETER', will now be displayed. In addition to connecting zero flow to the PTM, the PTG must be zeroed. The PTG is zeroed by placing the PTG in the volume mode (set switch to 'V'). Blow into the PTM until the PTG needle is in the middle of the scale. Place the PTM in the zeroing box to isolate it from all air flow. Adjust the zeroing potentiometer until the PTG needle does not move.

30. Return the PTG to the flow mode (set the switch to flow).

31. Press the ENTER key when zero flow is attained (by leaving the PTM in the zeroing box). You are instructed to do so by the 'PRESS ENTER TO CONTINUE' prompt. The computer samples the zero flow signal and averages the samples to compute the binary value corresponding to zero flow. Once computed, the zero flow binary value is displayed to the operator.

32. If temperature and gas viscosity corrections are to be used, as selected in step 24, insert the temperature transducer into the PTM head.

33. Turn the PTM heater on. Connect the harvard respirator to the PTM. Turn the respirator on and allow the PTM temperature to stabilize (a few minutes).

34. Press the ENTER key when the PTM temperature has stabilized. You are instructed to do so by the 'PRESS ENTER TO CONTINUE' prompt. The computer samples the flow signal. Once the flow signal has been sampled the respirator may be turned off. At

this time, the subject may be placed on the ergometer and fitted with the instrumentation.

35. The computer integrates each inspiratory and expiratory cycle of the pump to attain a volume that is directly proportional to the respirator's volume. Two sets of volumes are computed, one set uses no temperature or viscosity corrections and the other does. If temperature and gas viscosity corrections were not to be used, as selected in step 24, the corrected values are set equal to the non-corrected values. The two sets of volumes are displayed on the screen for each cycle of the respirator. The individual inspiratory and expiratory volumes are averaged and the proportionality factors (the calibration factors) which relate the computed volume to the respirator volume are determined. These calibration factors are displayed on the CRT.

Storage of the Calibration Data

36. Determine if the calibration data are to be stored on disk and respond to the prompt, 'STORE CALIBRATION FACTORS ON DISK? (Y/N)'. If the calibration values are not to be stored enter 'N' and proceed to step 41.

37. Enter the calibration file name.

38. Specify which disk drive is to contain the calibration data.

39. Enter the current date.

40. The calibration data will be stored.

41. The calibration procedure is complete.

A5.4.2 Operator instructions for collection of respiratory data with the CBRMS

Following is a step-by-step procedure for the collection of respiratory data using the Computer Based Respiratory Measurement System (CBRMS). The data procedure is best accomplished with two operators working as a team; one operator monitoring the subject and handling the instrumentation and the other operating the computer system. However, one operator can easily accomplish the task.

1. If the system is not on then power up the instrumentation system and the Pascal operating system as described in Section A5.2 of this appendix. The system is generally powered up prior to data collection, as the system is usually calibrated prior to data collection.

2. Make certain that the disk to be used for storage of the ASCII data is ready and has been configured to accept the data from the data collection program (see Section A5.3). If the disk has not been configured to accept the data, the program will crash and all data will be lost when an attempt is made to write the data to disk in step 9.

3. Execute the data collection program by pressing the R key (R for Run). Type the program name, 'DAP2', (it is not necessary to type DAP2.CODE, the system will tack on the .CODE part) followed by the ENTER key. Program execution will begin.

4. Enter the sampling frequency in response to the program prompt, 'ENTER THE SAMPLING FREQUENCY: '. Typically 50 (for 50 Hz) is used.

   Note: as with all numeric input while running a Pascal program, make certain that only numeric keys, the '-' key, the '+' key, or the '.' key is pressed since depression of any other key will cause the program to crash and all preceding data are lost.

5. Enter the number of samples to be collected in response to the program prompt, of 'ENTER NUMBER OF SAMPLES [24,000 MAX]:'. The number of samples must be in the range 1 to 24000 for the current system.

6. Make certain the subject is prepared, then press the 'ENTER' key to begin collection of the data. The number of samples and the length of time required to collect the samples will be displayed. The computer will sound an audible beep and display the message, 'DATA COLLECTION COMPLETE' when the data have been collected. The computer then calculates the maximum and minimum values of each channel and counts the number of data points equal to the saturation values (0 or 4095) for each channel. No operator input is required until all the data has been collected

and the maximum value, minimum value, and number of saturation points counted.

7. This is your last chance to make certain that the device to contain the ASCII data has been configured to hold ASCII data (see Section A5.3)

8. Select the mass storage device which will contain the ASCII data.

9. If the device presently contains ASCII data the computer will have you enter an alternate device. Otherwise, the data are stored.

10. The data collection procedure is complete.

A5.5 Compaction of the Calibration and Respiratory Data

Compaction of the calibration data and the respiratory data stored in ASCII format by the Pascal programs CAP2 and DAP2 to the numeric format of Basic will now be described. Compaction of the data is accomplished with two Basic programs: CAPCRUNCH for compaction of the calibration data and DAPCRUNCH for compaction of the respiratory data. The programs are simple to operate and require minimal operator input. Operator instructions for the compaction of the data will now be given. If only one type of compaction is necessary the other compaction operation may be bypassed.

The calibration and respiratory data will be stored on an 8" floppy disk. If the ASCII data are also stored on 8" floppy disk, the two disks will have to be switched during operation of the programs. The compaction programs will prompt the operator when the disks need to be exchanged.

1. Power up the Basic operating system as described in section A5.2 of this appendix. If the Basic operating system is powered up, execution of the AUTOSTart program may be accomplished by the command

```
LOAD "AUTOST:INTERNAL",1
```

2. Have an 8" disk prepared to accept the calibration data and

respiratory data. Continue with step 3 for the compaction of the calibration data or skip to step 11 for compaction of the respiratory data.

Compaction of the calibration data

3. With the AUTOSTart program executing, press the k_0 soft key which has the label "CCRUNCH". CAPCRUNCH will load and execution will begin.

4. Press the k_0 softkey labeled "CCRUNCH" again to continue with the compaction of the calibration data. CAPCRUNCH allows the return to the AUTOST program by pressing the k_9 softkey labeled "EXIT". This may be desired if the CCRUNCH key was accidentally pressed from the AUTOSTart program.

5. Enter the name of the calibration file.

6. Three mass storage devices will be displayed on the screen: platters 1 and 2 of the hard disk and the 8" disk. Enter the number corresponding to the mass storage location which contains the calibration data collected by CAP2.CODE.

7. If the ASCII data were stored on an 8" disk (as specified in step 6) then place the ASCII data disk in the 8" disk drive in accordance with the prompt displayed on the computer's screen. Press the CONTINUE key when this is accomplished. If the data were stored on the hard disk place the 8" disk which will contain the compacted data in the 8" disk drive and continue to step 10 (the ASCII data will be read, converted to numeric format, and stored; this will require several seconds).

8. The data will be read from the disk (a few seconds) and the operator prompted to place the 8" disk which will contain the compacted data in the 8" disk drive. Press the CONTINUE key when this is accomplished.

9. The data will be stored (several seconds) and the operator prompted to place the ASCII data disk in the 8" disk drive. Press the CONTINUE key when this is accomplished. The ASCII data will then be destroyed.

10. Program operation will return to the AUTOSTart program.

Compaction of the respiratory data

11. With the AUTOSTart program executing, press the k_1 soft key which has the label "DCRUNCH". DAPCRUNCH will load and execution will begin.

12. Press the k_1 softkey labeled "DCRUNCH" again to continue with the respiratory data compaction. DAPCRUNCH allows for the return to the AUTOST program by pressing the k_9 softkey labeled "EXIT". This may be desired if the DCRUNCH key was accidentally pressed from the AUTOSTart program.

14. Three mass storage devices will be displayed on the screen: platters 1 and 2 of the hard disk and the 8" disk. Enter the number corresponding to the mass storage location which contains the respiratory data collected by DAP2.CODE.

15. Enter the number of data points which were collected.

16. Enter the name of the CO_2 data file.

17. Enter the name of the O_2 data file. (If the O_2 file name begins with a O and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: O_2 file name is O1106N175 and CO_2 file name is C1106N175).

18. Enter the name of the flow data file. (If the flow file name begins with a V and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: flow file name is V1106N175 and CO_2 file name is C1106N175).

19. Enter the name of the temperature data file. (If the temperature file name begins with a T and the remaining portion of the file name is the same as the CO_2 file name with the exception of the first letter then simply press the CONTINUE key. Example: temperature file name is T1106N175 and CO_2 file name is C1106N175)

20. If the ASCII data were stored on an 8" disk (as specified in step 14) then place the ASCII data disk in the 8" disk drive in

accordance with the prompt displayed on the computer's screen. Press the CONTINUE key when this is accomplished. If the data were stored on the hard disk place the 8" disk which will contain the compacted data in the 8" drive and continue to step 24 (the ASCII data will be read, converted to numeric format, and stored; this will require a few minutes for a large number of data points).

21. The data will be read from the disk and converted from ASCII characters to numeric values, one data file at a time. This operation will take several minutes to accomplish for a large number of data points. The progress of the operation can be monitored on the bottom line of the CRT.

22. Place the data disk which will contain the numeric data in the 8" drive and press CONTINUE.

23. The data will be stored (several seconds) and you will be prompted to place the ASCII data disk in the 8" disk drive. Press the CONTINUE key when this is accomplished. The ASCII data will then be destroyed.

24. Program operation will return to the AUTOSTart program.

A5.6 Data Analysis and Display

The analysis and data display operations will now be described. The two operations are described in a single section since the present system uses the same routine to perform both analysis of the respiratory data and the display of the analyzed results.

1. Power up the Basic operating system as described in section A5.2 of this appendix. If the Basic operating system is powered up, execution of the AUTOSTart program may be accomplished by the command

   LOAD "AUTOST:INTERNAL",1

2. Press the k_2 soft key labeled "ANALYSIS". This will cause the ANALYSIS program to be loaded.

3. Press the k_1 softkey labeled "ANALYSIS" again to continue with the analysis and display program. ANALYSIS allows the return to

the AUTOST program by pressing the k_9 softkey labeled "EXIT". This may be desired if the ANALYSIS key was accidentally pressed from the AUTOSTart program.

4. Enter the subject's name or the subject's idnentifier (25 characters maximum). This name (or identifier) will become a part of all hard copy output of the ANALYSIS routine.

5. Previous execution of the data compaction routines, CAPCRUNCH and DAPCRUNCH, or the analysis routine, ANALYSIS, will have placed the subject data in the 9826's memory. If this is the case, information about the data is displayed on the screen. With data in memory, perform steps 6 thru 14 otherwise proceed to step 15.

6. Determine if the calibration data currently in memory is to be used and respond to the prompt, 'DO YOU WISH TO USE THESE CALIBRATION DATA? (Y/N)'. If the current calibration data are to be used answer yes, 'Y', and proceed to step 8.

7. Enter the name of the calibration file in response to the prompt, 'ENTER THE CALIBRATION FILE FILENAME.'. The specified calibration file will be loaded.

8. Determine if the respiratory data currently in memory is to be used and respond to the prompt, 'DO YOU WISH TO USE THESE B_by_B DATA? (Y/N)'. If the current respiratory data are to be used answer yes, 'Y', and proceed to step 14.

9. Enter the number of data points which were collected in response to the prompt, 'ENTER THE TOTAL NUMBER OF POINTS TO BE ANALYZED'.

10. Enter the name of the CO_2 data file in response to the prompt, 'ENTER THE CO2 SIGNAL FILE NAME'.

11. Enter the name of the O_2 data file in response to the prompt, 'ENTER THE O2 SIGNAL FILE NAME'. (If the O_2 file name begins with a O and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: O_2 file name is O1106N175 and CO_2 file name is C1106N175)

12. Enter the name of the flow data file in response to the prompt, 'ENTER THE FLOW SIGNAL FILE NAME'. (If the flow file name begins with a V and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: flow file name is V1106N175 and CO_2 file name is C1106N175)

13. Enter the name of the temperature data file in response to the prompt, 'ENTER THE FLOW TEMPERATURE SIGNAL FILE NAME'. (If the temperature file name begins with a T and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: temperature file name is T1106N175 and CO_2 file name is C1106N175)

14. Proceed to step 22.

15. Determine if the calibration data are to be loaded and respond to the prompt, 'LOAD CALIBRATION FACTORS FROM DISK? (Y/N)'. If calibration data are not to be loaded, then answer no, 'N', and proceed to step 17.

16. Enter the name of the calibration file in response to the prompt, 'ENTER THE CALIBRATION FILE FILENAME.'. The specified calibration file will be loaded.

17. Enter the number of data points which were collected in response to the prompt, 'ENTER THE TOTAL NUMBER OF POINTS TO BE ANALYZED'.

18. Enter the name of the CO_2 data file in response to the prompt, 'ENTER THE CO2 SIGNAL FILE NAME'.

19. Enter the name of the O_2 data file in response to the prompt, 'ENTER THE O2 SIGNAL FILE NAME'. (If the O_2 file name begins with a O and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: O_2 file name is O1106N175 and CO_2 file name is C1106N175)

20. Enter the name of the flow data file in response to the prompt,

'ENTER THE FLOW SIGNAL FILE NAME'. (If the flow file name begins with a V and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: flow file name is V1106N175 and CO_2 file name is C1106N175)

21. Enter the name of the temperature data file in response to the prompt, 'ENTER THE FLOW TEMPERATURE SIGNAL FILE NAME'. (If the temperature file name begins with a T and the remaining portion of the file name is the same as the CO_2 file name, with the exception of the first letter, then simply press the CONTINUE key. Example: temperature file name is T1106N175 and CO_2 file name is C1106N175)

22. Enter the experimental time corresponding to when respiratory data collection began (generally 60 seconds).

23. Enter a comment which will be printed on the hard copy output.

24. Determine the sampling frequency used to sample the respiratory data. If this sampling frequency is 50 Hz, then answer no, 'N', in response to the prompt, 'CHANGE SAMPLING FREQUENCY FROM 50 HZ? (Y/N)' and proceed to step 26.

25. Enter the sampling frequency used to sample the respiratory data.

26. Determine if the GMS time delay is to be computed on a breath-by-breath basis or a fixed value used and respond to the prompt, 'B-BY-B TIME DELAY OR FIXED TIME DELAY (B/F)?'. For a breath-by-breath time delay perform step 27 and for the fixed time delay perform steps 28 and 29.

27. Enter the average time delay to be used in case a valid breath-by-breath time delay can not be determined. Continue to step 30.

28. The current time delay is displayed on the screen. If this time delay is to be used then answer yes, 'Y', to the prompt, 'CHANGE TIME DELAY? (Y/N)' and proceed to step 30.

29. Enter the desired time delay.

30. Determine if the flow is to be calculated using temperature and viscosity corrections and respond to the prompt, 'USE TEMP. AND VISCOSITY CORRECTIONS? (Y/N)'. If the corrections are not to be used enter no, 'N', and proceed to step 35.

31. The ambient conditions currently in memory are displayed on the screen. If these ambient conditions are correct, enter yes, 'Y', in response to the prompt, 'Do you wish to use the ambient conditions above? (Y/N)' and proceed to step 35.

32. Enter the barometric pressure (inches of mercury).

33. Enter the percent relative humidity.

34. Enter the room temperature (^{o}F).

35. Determine if alveolar ventilation values based on FRC corrections are desired and respond to the prompt, 'Do you wish to use FRC corrections? (Y/N)'. Respond negatively, 'N', to this question as the FRC calculations are not implemented in this version of the ANALYSIS program. Refer to the reference [12].

Plotting the respiratory data

36. Determine if a plot of the respiratory data is desired and respond to the prompt, 'WOULD YOU LIKE A PLOT OF THE RESPIRATORY DATA? (Y/N)'. If a plot is not desired, enter no, 'N', and continue to step 42.

37. Enter the first data point to plot.

38. Enter the last data point to plot. At this time the maximum and minimum values for the specified range are calculated.

39. Once the maximum and minimum values have been determined you have approximately 20 seconds to determine the plotting device. At the end of this time and if no plotting device was specified the plot will automatically be sent to the 9826's screen. If the plotter is to be used, press the space bar. Set up the plotter. Enter 'PLOTTER' (all capitals) in response to the prompt, 'OUTPUT ON THE PLOTTER OR CRT? (PLOTTER/CRT)'.

If the plot was sent to the screen there, are two options for

obtaining a printed copy. For a small copy, press the 'SHIFT' key and 'DUMP GRAPHICS' key simultaneously. To obtain an expanded copy, type the command 'CALL Expanded_dump' and press the 'EXECUTE' key.

40. Press the 'CONTINUE' key.

41. Determine if additional plotting is desired and respond to the prompt, 'REDO GRAPHICS? (Y/N)'. If additional plots are desired enter yes, 'Y' and go to step 37.

Analyzing the respiratory data

42. Enter the first data point to analyze.

43. Enter the last data point to analyze.

44. Select whether the line-by-line output of the various respiratory volumes for each breath is desired. Generally this information is not needed and can be obtained later.

45. The calculation of the breath-by-breath respiratory parameters will begin. The present analysis routine can process 2000 points/min using the temperature and viscosity corrections and 7000 points/min without the corrections (these rates are based on the omission of the line-by-line output).

46. When the specified data range has been analyzed the mean values are printed.

Displaying the respiratory parameters

47. Determine if the calculated breath-by-breath respiratory parameters are to be displayed (printed or plotted) and respond to the prompt, 'DISPLAY THE BREATH-BY-BREATH DATA? (Y/N)'. If the results are not to be displayed, enter no, 'N', and proceed to step 68.

48. Select whether the parameters are to be plotted (two parameters per plot) or printed.

49. Select how the parameters are to be averaged. If displaying the individual breaths was selected proceed to step 55. If averaging of individual breaths was selected execute the next

step (50) then proceed to step 55. If window averaging of the parameters was selected execute steps 51 thru 54.

50. Specify the number of breaths to be averaged together. Continue to step 55.

51. Refer to Figure 2.2.3 for an illustration of the window averaging technique. Enter the windowing period. The windowing period is the distance (in time) the center of the window is slid.

52. Enter the window width. The window width is the time span of the window. All breaths in that span are averaged.

53. Enter the starting time. This is the time at which the center of the first window is placed.

54. Enter the ending time. The window is slid until the center of the window occurs after the ending time.

55. Determine if 'Bad Breaths' are to be included in the display. Bad breaths are those breaths which have an inspiratory or an expiratory tidal volume of less than 400 mL.

56. If plotting of the parameters was selected in step 48, proceed to the next step. If printing of the parameters was selected the values will be printed, proceed to step 67.

57. Enter the respiratory parameter to be displayed in the top plot.

58. Enter the respiratory parameter to be displayed in the bottom plot.

59. Determine if the plot is to be displayed on the 9826's screen or the plotted on the plotter.

60. Two events may be displayed on the plot. If the events are not to be specified then proceed to step 65.

61. Specify the time of the first event.

62. Specify the the first event.

63. Specify the time of the second event.

64. Specify the second event.

65. Enter a title for the plot.

66. If the plot was sent to the screen (as specified in step 59) there are two options for obtaining a printed copy. For a small copy press the 'SHIFT' key and 'DUMP GRAPHICS' key simultaneously. To obtain an expanded copy, type the command 'CALL Expanded_dump' and press the 'EXECUTE' key.

67. Additional display of the results is possible by responding to the prompt, 'Do you wish to exit to the analysis routine? (Y/N)' with a no, 'N'. Proceed to step 48.

68. The current subject's data can be re-analyzed by responding to the prompt, 'REDO ANALYSIS? (Y/N)' with a yes, 'Y'. Proceed to step 4. If the subject's data is not to be re-analyzed proceed to step 1.

A5.7 Storage and Display of the Heart Rate Data

The storage of the heart rate data will now be described. The display of the heart rate is accomplished with Sprick's Universal Plotting Routine [16].

1. Power up the Basic operating system as described in section A5.2 of this appendix. If the Basic operating system is powered up, execution of the AUTOSTart program may be accomplished by the command

   LOAD "AUTOST:INTERNAL",1

2. Press the k_4 soft key labeled "STOREHR". This will cause the STOREHR program to be loaded.

3. Press the k_4 softkey labeled "STOREHR" again to continue with the storage of the heart rate values. STOREHR allows the return to the AUTOST program by pressing the k_9 softkey labeled "EXIT". This may be desired if the STOREHR key was accidentally pressed from the AUTOSTart program. If you wish to plot the heart rate values with Sprick's plotting routine press the k_1 softkey labeled 'PLOT' (refer to step 10 for instructions on how to label the X-axis of the plot).

4. Enter the file name which will contain the heart rate values.

5. Enter the heart rate value. Repeat this step for each value. Do not worry about mistakes, as they may be corrected later (step 6). If all values have been entered enter, a negative number.

6. Ten heart rate values are displayed on the screen. If you wish to alter any of these values, enter 'Y' for yes and continue to step 7. If you do not wish to alter any of the displayed values then enter 'N' for no. If all heart rate values have been displayed then continue to step 8, otherwise this step will be repeated for the next ten heart rate values.

7. Enter the number corresponding to the value you wish to change. Enter the new heart rate value. Return to step 6 for the display of the same ten heart rate values including the updated value.

8. The values are stored on the disk in the 8" disk drive.

9. If you wish to plot the heart rate values answer 'Y' for yes. If you wish to return to the AUTOSTart routine enter 'N' for no, and press the k_9 softkey labeled 'EXIT'. The AUTOSTart program will be loaded.

10. A message will be displayed on the screen instructing the use 'TIME' (all capital letters) as the X-axis of the plot to be obtained using Sprick's plotting routine. Press CONTINUE to proceed to the plotting routine. When using Sprick's plotting routine enter, 30 for the sampling period and Seconds for the X-axis units in addition to 'TIME' as the X-axis.

APPENDIX VI. PASCAL PROGRAM DOCUMENTATION AND LISTING

This appendix gives the details of the Pascal programs used by the CBRMS. The documentation for each program is broken down into descriptions of the individual Pascal procedures which make up the program. The variables for a particular procedure will be defined in that procedure's section. The procedures within each program are documented in alphabetical order. The variables within each procedure are also listed in alphabetical order.

A6.1 CAP2: a Pascal Routine for system Calibration

The Pascal program CAP2 (CAlibration Program, version 2) is used to generate calibration factors which relate the sampled data to the analog respiratory signals. A flowchart for the program is given in Chapter 6, Section 1. The method used by CAP2 to control the DAM is described by Riblett [13]. The calibration file contents maintain the file structure specified by Riblett, one serial ASCII file, but has been expanded to contain additional data. The contents of the file in order are

Co2_dc_offset (4 ASCII bytes)
O2_dc_offset (4 ASCII bytes)
Bin_zero_flow (4 ASCII bytes)
Co2_cal (25 ASCII bytes)
O2_cal (25 ASCII bytes)
Non_C_I_flow_cal (25 ASCII bytes)
Non_C_E_flow_cal (25 ASCII bytes)
Time_delay (25 ASCII bytes)
S (4 ASCII bytes)
Ol (25 ASCII bytes)
Ta (25 ASCII bytes)
Tb (25 ASCII bytes)
Tc (25 ASCII bytes)
Date (25 ASCII bytes)
Corr_I_flow_cal (25 ASCII bytes)
Corr_E_flow_cal (25 ASCII bytes)
Pb (25 ASCII bytes)
Rel_humid (25 ASCII bytes)
Room_temp (25 ASCII bytes)

A description of each of the procedures used by CAP2 follows.

A6.1.1 Ask_Y_or_N Procedure

The procedure Ask_Y_or_N (Ask Yes or No) is used to obtain only a Yes or a No response to a question. The procedure returns the character 'Y' for a Yes response and the character 'N' for a No response.

Variables passed to the procedure

None

Variables returned by the procedure

Q - character, passed by reference.

Variable definitions

Q - string variable which contains a capital 'Y' for a Yes response or a capital 'N' for a No response.

A6.1.2 BEEP Procedure

The procedure BEEP sounds an audible beep. This procedure is used to gain the operator's attention.

Variables passed to the procedure

None

Variables returned by the procedure

None

Variable definitions

No variables used.

A6.1.3 BIT_HI and BIT_TST Procedures

BIT_HI and BIT_TEST are external assembly language procedures. These procedures are used to control the DAM. BIT_HI and BIT_TEST procedures are documented by Riblett [13].

A6.1.4 Calc Incr vol Procedure

The Calc_Incr_Vol (Calculate the Incremental Volume) procedure calculates an incremental volume which is used in integrating the flow. Calc_Incr_Vol is nested within the Flow calibration, Flow_cal, procedure. The Rel_viscos function is nested within Calc_Incr_Vol. The procedure uses a sample of the differential pressure signal, a sample of the temperature signal, and the amhient conditions to compute an incremental volume. Two types of incremental volumes are computed, one under STPD conditions the other under ITPnS conditions.

Variables passed to the procedure

Bin_delta_P - integer, passed hy value.

Fh2o - real, passed by value.

T - real, passed hy value.

Temperature - real, passed by value.

Variables returned by the procedure

Corr_Incr_vol - real, passed hy reference.

Non_C_Incr_vol - real, passed hy reference.

Variable definitions

Bin_delta_P - an integer variahle which is proportional to the PTG differential pressure signal.

Corr_Incr_vol - a real variable proportional to the incremental volume under STPD conditions which flowed through the PTM during the sampling period.

Fh2o - a real variable equal to the fractional concentration of water vapor in the flow gas.

ITPnS_to_STPD - a real variahle which converts the ITPnS volume to a STPD volume.

Non_C_Incr_vol - a real variahle proportional to the incremental volume under ITPnS conditions which flowed through the PTM during the sampling period.

Rel_Viscosity - a real variahle equal to the viscosity of the gas relative to the viscosity of room air under STPD conditions.

Temperature - a real variahle equal to the temperature of the flow gas.

T - a real variahle equal to the sampling period.

A6.1.5 CAP2 Procedure

The CAP2 procedure provides the main control of the the calibration program, CAP2. CAP2 calls the calibration routines; GASCAL, FLOWCAL, and TEMPCAL; and stores the calibration factors in a disk file. CAP2 also calls the Ask_Y_or_N, Beep, HOLD_UP, and Line_feed procedures.

Variables passed to the procedure

None

Variables returned by the procedure

The following variables are stored in the calibration file.

Bin_zero_flow - integer.
Corr_E_flow_cal - real.
Corr_I_flow_cal - real.
Co2_cal - real.
Co2_dc_offset - integer.
Date - character string.
Non_C_E_flow_cal - real.
Non_C_I_flow_cal - real.
O1 - real.
O2_cal - real.
O2_dc_offset - integer
Pb - real.
Rel_humid - real.
Room_temp - real.
S - integer.
Ta - real.
Tb - real.
Tc - real.
Time_delay - real.

Variable definitions

Bin_zero_flow - an integer variable equal to the average sampled value from the PTG channel with zero flow through the PTM.

Corr_E_flow_cal - a real variable relating binary data obtained from the flow channel to the the PTM differential pressure generated by expiratory flows. This calibration factor

does use the temperature and viscosity corrections.

Corr_flag - a Boolean variable which is set to true when the flow signal is corrected to STPD conditions or is false if the flow signal is not corrected.

Corr_Incr_Vol - a real variable equal to the incremental volume obtained from one flow sample using temperature and viscosity corrections. Although this variable is declared within the context of CAP2 it is not used. However, the variable is used within FLOWCAL but is not declared. Therefore, the Corr_Incr_Vol declaration should be moved to the FLOWCAL procedure.

Corr_I_flow_cal - a real variable relating binary data obtained from the flow channel to the the PTM differential pressure generated by inspiratory flows. This calibration factor does use the temperature and viscosity corrections.

Co2_cal - a real variable relating the binary data collected from the O_2 channel to fractional CO_2.

Co2_dc_offset - an integer variable equal to the average binary value obtained from the CO_2 channel for room air concentration of CO_2.

Date - a string containing the date the calibration was performed.

Device_pointer - an integer which is used to indicate which mass storage device is to contain the calibration data.

F - a text variable associated with the calibration file specified by the string variable Fname.

Fname - a string containing the calibration file name.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 1, the CO_2 channel, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 2, the O_2 channel, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 3, the flow channel, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 4, the temperature

channel, of the DAM.

MSI_devices - a string array containing three elements. These elements correspond to the three mass storage devices which can be used for the storage of the calibration data. Array element 1 contains #12 corresponding to platter 1 of the hard disk, element 2 contains #13 corresponding to platter 2 of the hard disk, and element 3 contains #7 corresponding to the 8-inch disk drive.

Non_C_E_flow_cal - a real variable relating binary data obtained from the flow channel to the the PTM differential pressure generated by expiratory flows. This calibration factor does not use temperature or viscosity corrections.

Non_C_I_flow_cal - a real variable relating binary data obtained from the flow channel to the the PTM differential pressure generated by inspiratory flows. This calibration factor does not use the temperature or viscosity corrections.

Non_C_Incr_Vol - a real variable equal to the incremental volume obtained from one flow sample not using the temperature and viscosity corrections. Although this variable is declared within the context of CAP2 it is not used. However, the variable is used within FLOWCAL but is not declared. Therefore, the Non_C_Incr_Vol declaration should be moved to the FLOWCAL procedure.

NSTRING- a string variable used to represent the ASCII value of an integer number.

O1 - a real variable containing the fractional concentration of O_2 in the calibration gas.

O2_cal - a real variable relating the binary data collected from the O_2 channel to fractional O_2.

O2_dc_offset - an integer variable equal to the average binary value obtained from the O_2 channel when the GMS probe samples the calibration gas.

Pb - a real variable set equal to the ambient barometric pressure (mm Hg).

Q - string variable containing the response to a Yes or No question.

Rel_humid - a real variable set equal to the relative humidity in the room air.

Room_temp - a real variable set equal to the ambient temperature (^{o}C).

RSTRING- a string variable used to represent the ASCII value of a real number.

S - an integer variable set equal to the DAM sampling frequency (Hz).

Sam - an integer variable equal to the number of samples to be collected form the DAM.

Ta - a real variable equal to the computed second order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Tb - a real variable equal to the computed first order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Tc - a real variable equal to the computed zero order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

TEMP - an integer variable needed by the STRWRITE function to convert the calibration data to ASCII strings.

Time_delay - a real variable equal to the fixed GMS time delay used in the calibration routine. This value was retained to remain compatible with the calibration files generated by Creel's program [4].

A6.1.6 CLKSET Procedure

The procedure CLKSET sets the 8253 timer in the DAM with the desired sampling frequency. CLKSET is called by the main CAP2 procedure to set the DAM's sampling frequency. This procedure was designed by Riblett [13] and has no modifications for version 2 of the calibration program (CAP2).

Variables passed to the procedure

None

Variables returned by the procedure

S - integer, passed by reference.

Variable definitions

Fl - an integer used to set the least significant byte of the 8253 timer.

Fm - an integer used to set the most significant byte of the 8253 timer.

S - an integer equal to the user specified sampling frequency.

X - an integer variable used to set clock 1 of the 8253 timer.

A6.1.7 DATA_COLLECT Procedure

The procedure DATA_COLLECT provides the necessary control of the DAM to collect a set of data from four channels. DATA_COLLECT is called by the calibration procedures; GASCAL, FLOWCAL, and TEMPCAL; to sample the analog signals. DATA_COLLECT calls the assembly language procedures BIT_HI and BIT_TST to test the status of the DAM. This procedure was designed by Riblett [13] and has only minor modifications for version 2 of the calibration program (CAP2).

Variables passed to the procedure

S - integer, passed by reference.

Sam - integer, passed by reference.

Variables returned by the procedure

Line1 - integer array, passed by reference.

Line2 - integer array, passed by reference.

Line3 - integer array, passed by reference.

Line4 - integer array, passed by reference.

Variable definitions

Chana - a constant value equal to 15359 which sets the DAM's analog multiplexer so that channel A is passed to the analog to digital converter.

Chanb - a constant value equal to 13311 which sets the DAM's analog multiplexer so that channel B is passed to the analog to digital converter.

Chanc - a constant value equal to 11263 which sets the DAM's analog multiplexer so that channel C is passed to the analog to digital converter.

Chand - a constant value equal to 9215 which sets the DAM's analog multiplexer so that channel D is passed to the analog to digital converter.

Del - an integer variable used in the delay loop that allows the sample and hold amplifiers to reach equilibrium once the hold command is given.

I - an integer variable used as a loop counter and a pointer into the four data arrays during collection of the specified number of datum points.

Line1 - an integer array containing the sampled data acquired from the CO_2 channel of the DAM.

Line2 - an integer array containing the sampled data acquired from the O_2 channel of the DAM.

Line3 - an integer array containing the sampled data acquired from the flow channel of the DAM.

Line4 - an integer array containing the sampled data acquired from the temperature channel of the DAM.

Mask - a constant value equal to 4095 which is used to mask all but the 12 least significant data bits.

R4 - an integer variable which is loaded with the contents of register 4 of the GPIO.

R6 - an integer variable which is loaded with the contents of register 6 of the GPIO.

S - an integer equal to the user specified sampling frequency.

Sam - an integer variable equal to the number of samples to be collected from the DAM.

A6.1.8 FLOWCAL Procedure

The procedure FLOWCAL calculates the calibration factors which relate the sampled data to the flow through the PTM. FLOWCAL has two procedures nested within it, Calc_Incr_Vol and Valid_pointer. FLOWCAL also calls the Ask_Y_or_N, Beep, DATA_COLLECT, HOLDUP, and Line_feed.

The flow calibration procedure uses several constant values which will need to be changed with a different experimental setup. These constant variables are

Corr_Pump_vol - the volume of the calibration pump converted to

STPD conditions, currently 0.560 L (STPD).

First_element - the first element of the water vapor pressure array, currently 0.

Last_element - the last element of the water vapor pressure array, currently 250.

Max_Temp - the maximum temperature represented in the water vapor pressure array, currently 45°C.

Min_Temp - the minimum temperature represented in the water vapor pressure array, currently 20°C.

Non_C_Pump_vol - the volume of the calibration pump under ATPS conditions, currently 0.647 L (ATPS).

Vap_File_name - the location and file name of the Pascal file containing the water vapor pressure array, currently '#14:VAP_PASCAL' (the hard disk, platter 3, under the file name VAP_PASCAL).

Variables passed to the procedure

Corr_flag - Boolean, passed by value.

Pb - real, passed by value.

Rel_humid - real, passed by value.

Room_temp - real, passed by value.

S - integer, passed by value.

Sam - integer, passed by value.

Ta - real, passed by value.

Tb - real, passed by value.

Tc - real, passed by value.

Variables returned by the procedure

Bin_zero_flow - integer, passed by reference.

Corr_E_flow_cal - real, passed by reference.

Corr_I_flow_cal - real, passed by reference.

Non_C_E_flow_cal - real, passed by reference.

Non_C_I_flow_cal - real, passed by reference.

Variable definitions

A - an integer variable used as an index into the sampled PTG signal array, Line3.

Array_pointer - an integer variable used as an index into the water vapor pressure array, Vap.

Bin_delta_P - an integer variable equal to the binary differential pressure developed by the PTM. This is equal to the sampled value minus the binary value corresponding to zero differential pressure (zero flow).

Bin_zero_flow - an integer variable equal to the average binary value sampled for zero flow.

Corr_Aire - a real variable used in the expiratory integration loop to sum up the volume (binary representation) of air expired by the calibration pump using the flow calculated under STPD conditions.

Corr_Airi - a real variable used in the inspiratory integration loop to sum up the volume (binary representation) of air inspired by the calibration pump using the flow calculated under STPD conditions.

Corr_Avg_Vol_expr - a real variable equal to the average volume (binary representation) expired by the calibration pump calculated under STPD conditions.

Corr_Avg_Vol_insp - a real variable equal to the average volume (binary representation) inspired by the calibration pump calculated under STPD conditions.

Corr_E_flow_cal - a real variable equal to the flow calibration factor which converts the sampled expiratory flow (converted to STPD conditions) to the units of liters per second.

Corr_flag - a Boolean variable indicating the use of temperature and viscosity corrections in the calibration of the flow signal.

Corr_Incr_Vol - a real variable equal to the incremental volume obtained from one flow sample using temperature and viscosity corrections.

Corr_I_flow_cal - a real variable equal to the flow calibration factor which converts the sampled inspiratory flow (converted to STPD conditions) to the units of liters per second.

Corr_Pump_vol - a constant value equal to the volume of the calibration pump converted to STPD conditions, 0.560 L (STPD).

Corr_Tot_vol_expr - a real variable equal to the total STPD

volume expired hy all pump cycles.

Corr_Tot_vol_insp - a real variable equal to the total STPD volume inspired by all pump cycles.

Fh2o - the fractional concentration of water vapor in the room air.

First_element - a constant value equal to the first element of the water vapor pressure array, 0.

I - an integer used as an array pointer into the flow signal array, Line3.

Last_element - a constant value equal to the last element of the water vapor pressure array, 250.

Line1 - an integer array, of 24,000 elements, containing the sampled values ohtained from channel 1, the CO_2 channel, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values ohtained from channel 2, the O_2 channel, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values ohtained from channel 3, the flow channel, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values ohtained from channel 4, the temperature channel, of the DAM.

Max_Temp - a constant value equal to the maximum temperature represented in the water vapor pressure array, 45°C.

Min_Temp - a constant value equal to the minimum temperature represented in the water vapor pressure array, 20°C.

Non_C_Aire - a real variable used in the expiratory integration loop to sum up the volume (hinary representation) of air expired hy the calihration pump using the under flow calculated ATPS conditions.

Non_C_Airi - a real variable used in the inspiratory integration loop to sum up the volume (binary representation) of air inspired hy the calihration pump using the under flow calculated ATPS conditions.

Non_C_Avg_Vol_expr - a real variahle equal to the average volume (hinary representation) expired hy the calibration pump under ATPS conditions.

Non_C_Avg_Vol_insp - a real variable equal to the average volume (binary representation) inspired by the calibration pump under ATPS conditions.

Non_C_E_flow_cal - a real variable equal to the flow calibration factor which converts the sampled expiratory flow to the units of liters per second.

Non_C_I_flow_cal - a real variable equal to the flow calibration factor which converts the sampled inspiratory flow to the units of liters per second.

Non_C_Incr_Vol - a real variable equal to the incremental volume obtained from one flow sample not using the temperature and viscosity corrections.

Non_C_Pump_vol - a constant value equal to the volume of the calibration pump under ATPS conditions, 0.647 L (ATPS).

Non_C_Tot_vol_expr - a real variable equal to the total ATPS volume expired by all pump cycles.

Non_C_Tot_vol_insp - a real variable equal to the total ATPS volume inspired by all pump cycles.

No_breaths - an integer variable equal to the number of calibration pump cycles (breaths).

Pb - a real variable equal to the barometric pressure.

Rel_humid - a real variable equal to the ambient relative humidity.

Room_temp - a real variable equal to the ambient temperature.

S - an integer variable equal to the sampling frequency.

Sam - an integer variable equal to the number of samples to be collected from the DAM.

Still_calculating - a Boolean variable which is True as long as flow data remains to be integrated.

Still_expire - a Boolean variable which is True when the flow is an expiratory flows.

Still_inspire - a Boolean variable which is True when the flow is an inspiratory flows.

T - a real variable equal to the sampling period (Seconds).

Ta - a real variable equal to the computed second order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Tb - a real variable equal to the computed first order

coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Tc - a real variable equal to the computed zero order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Temperature - a real variable equal to the temperature of the flow signal. The temperature is calculated using the sampled temperature signal and the temperature calibration factors; Ta, Tb, and Tc.

Tot_zero - an integer variable equal to the sum of the sampled PTG data when the PTM is isolated from flow.

Vap - a real array containing the water vapor pressures values.

Vap_file - a real file containing the water vapor pressure values.

Vap_file_name - a constant character string containing the name of the water vapor pressure file, '#14:VAP_PASCAL'.

A6.1.9 GASCAL Procedure

The gas calibration procedure, GASCAL, calibrates the Perkin-Elmer gas mass spectrometer for both the CO_2 and O_2 channels. This procedure was designed by Riblett [13] and has only minor modifications for version 2 of the calibration program (CAP2).

Variables passed to the procedure

S - integer, passed by reference.

Sam - integer, passed by reference.

Variables returned by the procedure

Co2_cal - real, passed by value.

Co2_dc_offset - integer, passed by value.

O1 - real, passed by value.

O2_cal - real, passed by value.

O2_dc_offset - integer, passed by value.

Variable definitions

Avg_CO2_cal_gas - an integer variable containing the average of the sampled data obtained from the CO_2 channel when the GMS probe was placed in the calibration gas.

Avg_CO2_room_air - an integer variable containing the average of the sampled data obtained from the CO_2 channel when the GMS probe was placed in room air.

Avg_O2_cal_gas - an integer variable containing the average of the sampled data obtained from the O_2 channel when the GMS probe was placed in the calibration gas.

Avg_O2_room_air - an integer variable containing the average of the sampled data obtained from the O_2 channel when the GMS probe was placed in room air.

Ch - a real variable containing the fractional concentration of CO_2 in the calibration gas.

Cl - a real variable containing the fractional concentration of CO_2 in room air.

Co2_cal - a real variable relating the binary data collected from the CO_2 channel to fractional CO_2.

Co2_dc_offset - an integer variable equal to the average binary value obtained from the CO_2 channel for room air concentration of CO_2.

I - an integer variable used as a loop counter and a pointer into the data arrays.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 1, the CO_2 channel, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 2, the O_2 channel, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 3, the flow channel, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 4, the temperature channel, of the DAM.

Ol - a real variable containing the fractional concentration of O_2 in the calibration gas.

Oh - a real variable containing the fractional concentration of O_2 in room air.

O2_cal - a real variable relating the binary data collected from the O_2 channel to fractional O_2.

O2_dc_offset - an integer variable equal to the average binary value obtained from the O_2 channel for the O_2 concentration in the calibration gas.

S - an integer variable equal to the sampling frequency.

Sam - an integer variable equal to the number of samples to be collected from the DAM.

A6.1.10 HOLD_UP Procedure

The hold up procedure, HOLD_UP, pauses program execution until the operator is ready to proceed. To continue program execution the operator must press the 'ENTER' key. This procedure was designed by Riblett [13] and has no modifications for version 2 of the calibration program (CAP2).

Variables passed to the procedure

None

Variables returned by the procedure

None

Variable definitions

No variables used.

A6.1.11 Line_feed Procedure

The line feed procedure, Line_feed, sends a specified number of line feeds to the screen.

Variables passed to the procedure

I - integer, passed by value.

Variables returned by the procedure

None

Variable definitions

I - an integer variable containing the number of line feeds to send to the screen.

J - an integer variable used as a loop counter. J is incremented for each line feed which is sent until it is equal to the number of line feeds to be sent, I.

A6.1.12 Rel_viscos Function

The relative viscosity function, Rel_viscos, returns a real value equal to the viscosity of the flow gas relative to the the viscosity of room air under STPD conditions.

Variables passed to the procedure

FH2O - real, passed by value.

Temperature - real, passed by value.

Variables returned by the procedure

Not applicable.

Variable definitions

FCO2 - a real variable containing the fractional concentration of CO_2 in the room air taking into account the amount of water vapor in the room.

FH2O - a real variable containing the fractional concentration of water vapor in the room air.

FN2 - a real variable containing the fractional concentration of N_2 in the room air taking into account the amount of water vapor in the room.

FO2 - a real variable containing the fractional concentration of O_2 in the room air taking into account the amount of water vapor in the room.

Temperature - a real variable containing the instantaneous temperature of the flow gas.

Vis_CO2 - the viscosity of CO_2 in the flow gas.

Vis_H2O - the viscosity of H_2O in the flow gas.

Vis_N2 - the viscosity of N_2 in the flow gas.

Vis_O2 - the viscosity of O_2 in the flow gas.

Vis_STPD - the viscosity of room air under STPD conditions.

A6.1.13 TEMPCAL Procedure

The temperature calibration procedure, TEMP_CAL, calculates the calibration factors for the temperature transducer. The temperature calibration factors are the coefficients of a second order polynomial. This procedure was designed by Riblett [13] and has only minor modifications for version 2 of the calibration program (CAP2).

TEMP_CAL uses calibration parameters for the thermometer used to calibrate the thermocouple. These parameters allow a thermometer reading entered by the operator to be converted to the true temperature. When a new thermometer is used these calibration factors must be determined and placed in the TEMP_CAL procedure. The constant variable, Thermometer_cal_Y_int, must be set equal to the y intercept of the thermometer's calibration curve. The constant variable, Thermometer_cal_Slope, must be set equal to the slope of the calibration curve.

Variables passed to the procedure

S - integer, passed by reference.

Sam - integer, passed by reference.

Variables returned by the procedure

Ta - real, passed by reference.

Tb - real, passed by reference.

Tc - real, passed by reference.

Variable definitions

Bin_temp - a real two dimensional array (matrix) used to compute the temperature calibration coefficients.

Bin_temp_inv - a real two dimensional array (matrix) equal to the inverse of Bin_temp.

I - an integer variable used as a loop counter and a pointer into the temperature data array.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 1, the CO_2 channel, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 2, the O_2 channel, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 3, the flow channel, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 4, the temperature channel, of the DAM.

Norm_a - a real variable containing the normal of the Bin_temp matrix.

S - an integer variable equal to the sampling frequency.

Sam - an integer variable equal to the number of samples to be collected from the DAM.

Ta - a real variable equal to the computed second order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Tb - a real variable equal to the computed first order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Tc - a real variable equal to the computed zero order coefficient of the function relating the sampled temperature data to the actual temperature in ^{o}C.

Thermometer_cal_Slope - a constant variable equal to the slope of the calibration thermometer's calibration curve. With the current calibration thermometer this value is 1.0.

Thermometer_cal_Y_int - a constant variable equal to the y intercept of the calibration thermometer's calibration curve. With the current calibration thermometer this value is -0.35.

Thigh - a real variable equal to the temperature of the high temperature water bath.

Thigh_bin - an integer variable equal to the average binary value of the sampled temperature data read for the high temperature bath.

Tlow - a real variable equal to the temperature of the low temperature water bath.

Tlow_bin - an integer variable equal to the average binary value of the sampled temperature data read for the low temperature bath.

Tmid - a real variable equal to the temperature of the middle temperature water bath.

Tmid_bin - an integer variable equal to the average binary value of the sampled temperature data read for the middle temperature bath.

Tot_temp - an integer variable containing the sum of all samples read from the temperature channel.

A6.1.14 Valid_pointer Procedure

The procedure, Valid_pointer, is used to determine if the pointer into the saturated water vapor array is within the limits of the array. This procedure was originally designed with the conception that the fractional concentration of water vapor in the flow gas would be determined from the instantaneous temperature of the flow gas. In this instance, the Valid_pointer procedure would protect against temperatures computed from bad sampled temperature data points which result from 'glitches' by the DAM. In this case the computed temperature would point to a water vapor pressure value which is beyond the bounds of the water vapor pressure array and thereby causing the CAP2 program to 'crash'. The Valid_pointer procedure would protect against this possibility by checking to make certain the water vapor pointer is within bounds.

Variables passed to the procedure

Array_pointer - integer, passed by reference.

Variables returned by the procedure

Array_pointer - integer, passed by reference.

Variable definitions

Array_pointer - an integer variable containing the pointer into the water vapor pressure table that is to be checked.

A6.1.15 CAP2 Program Listing

```
$SYSPROG ON$
$LINES 57$
$TABLES$
$REF 50$      {ALLOCATE ROOM FOR REFERENCE TABLE}
PROGRAM CAP2(INPUT,OUTPUT);
{*****************************************************************************

        SYSTEM CALIBRATION ROUTINE

        PASCAL REV 2.1 SOURCE FILENAME:  CAP2.TEXT

        Department of Electrical and Computer Engineering
        Kansas State University

        REVISION        DATE                    PROGRAMMER
        --------        ----                    ----------
          1.0           JUNE 28, 1984           LOREN E. RIBLETT, JR.
          2.0           August 31,1984          Michael Masters

 *****************************************************************************

        PURPOSE

                THIS ROUTINE PERFORMS ALL THE NECESSARY ACTIONS TO CALIBRATE
                THE BREATH-BY-BREATH RESPIRATORY SYSTEMS INSTRUMENTATION AND
                STORES THE CALIBRATION FACTORS IN ASCII CALIBRATION FILES.

        ROUTINE(S) CALLED BY THIS ROUTINE

                BIT_TST - 68000 ASSEMBLY MODULE THAT WAITS UNTIL THE EOC BIT
                          ON THE DAM GOES HIGH, THEN LOW
                BIT_HI  - 68000 ASSEMBLY MODULE THAT WAITS UNTIL THE EOC BIT
                          ON THE DAM GOES LOW
                CLKSET  - INTERNAL PROCEDURE THAT SETS THE 8253 TIMER CHIP
                          FOR THE PROPER SAMPLING FREQUENCY
                HOLD_UP - INTERNAL PROCEDURE FOR TEMPORARY PAUSING OF PROGRAM
                          OPERATION
                DATA_COLLECT - INTERNAL PROCEDURE THAT CONTROLS THE DAM IN
THE
                                PROPER FASHION TO COLLECT THE DESIRED NUMBER
O"
                                DATA POINTS
                GASCAL  - INTERNAL PROCEDURE THAT CALIBRATES THE PERKIN-ELMER
                          GAS MASS SPECTROMETER
                FLOWCAL - INTERNAL PROCEDURE THAT CALIBRATES THE FLEISCH/
                          GODART FLOW APPARATUS
                TEMPCAL - INTERNAL PROCEDURE THAT CALIBRATES THE RESPIRATORY
                          TEMPERATURE TRANSDUCER

                BEEP    - Sounds an audible beep to alert the operator.

                Ask_Y_or_N - Obtains the response to a yes or no question
                             and allows only a 'Y', 'y', 'N', or 'n' response.

                Line_feed - Sends a specified number of line feeds to the
                            terminal.

*****************************************************************************

        NOTE 1:  THE DAM SHOULD BE CONNECTED TO THE HP9826 COMPUTER VIA A
```

GPIO INTERFACE AT SELECT CODE #12. THIS INSURES THE PROPER DEVICE ADDRESS FOR SENDING AND RECEIVING INFORMATION BETWEEN THE DAM AND THE HP9826.

NOTE 2: TWO EXTERNAL 68000 ASSEMBLY LANGUAGE ROUTINES (BIT_TST AND BIT_HI) ARE UTILIZED BY CAP2 TO HANDLE THE HIGH SPEED REQUIREMENTS NEEDED TO MONITOR THE EOC (End Of Conversion) SIGNAL FROM THE DAM.

NOTE 3: BEEP Procedure. The procedure BEEP was written to sound an audible beep to the operator. This is especially useful for prompting for operator input bring something to the operators attention.

NOTE 4: Line_feed Procedure. The procedure Line_feed was written to send a specified number of line feeds to the terminal.

NOTE 5: HOLD_UP PROCEDURE. THE PROCEDURE HOLDUP WAITS FOR THE OPERATOR TO PRESS THE ENTER KEY FOR EXECUTION TO RESUME.

NOTE 6: Ask_Y_or_N Procedure. The procedure Ask_Y_or_N prompts the operator for either a yes response (a 'Y' or a 'y') or a no rsponse ('N' or 'n'). The procedure continually prompts for a yes or no response until either the 'Y' key, 'y' key, 'N' key, or 'n' key is entered. The operatory response is returned.

NOTE 7: CAP2'S CLKSET PROCEDURE SETS THE 8253 TIMER CHIP ON THE DAM TO THE DESIRED SAMPLING FREQUENCY. A MAXIMUM SAMPLING RATE OF 350 HZ IS RECOMMENDED. THIS VALUE MAY HAVE TO BE REDUCED IF SUBSTANTIAL ADDITIONS TO THE PROCEDURE DATA_COLLECT ARE MADE.

NOTE 8: PROCEDURE DATA_COLLECT IS USED BY CAP2 TO SAMPLE THE NECESSARY CHANNELS FOR CALIBRATION PURPOSES. IT (DATA_COLLECT) IS IDENTICAL TO THE DATA_COLLECT PROCEDURE USED BY DAP.CODE.

NOTE 9: PROCEDURE GASCAL IS DESIGNED TO CALIBRATE THE PERKIN-ELMER GAS MASS SPECTROMETER FOR BOTH THE CO2 (CHANNEL 1) AND O2 (CHANNEL 2) DAM CHANNELS.

NOTE 10: PROCEDURE FLOWCAL IS USED TO CALIBRATE THE FLEISCH/GODART PNEUMOTACH ASSEMBLY. FLOWCAL DETERMINES NOT ONLY THE BINARY VALUE FOR ZERO FLOW BUT ALSO COMPUTES INSPIRATORY AND EXPIRATORY FLOW CALIBRATION FACTORS FOR BOTH THE NON-CORRECTED DATA AND THE CORRECTED DATA.

NOTE 11: PROCEDURE TEMPCAL IS DESIGNED TO CALIBRATE THE THERMOCOUPLE FOR MEASURING RESPIRATORY TEMPERATURE. TEMPCAL DETERMINES A 2ND ORDER EQUATION FOR CONVERTING BINARY DAM FIGURES TO ACTUAL TEMPERATURE VALUES.

NOTE 12: CAP2 CONVERTS ALL THE CALIBRATION FACTORS TO ASCII UNITS AND STORES THESE UNITS IN A SINGLE ASCII DATA FILE

NOTE 13: SEE EXTERNAL PROGRAM DOCUMENTATION FOR MORE DETAILS.

**********}

{*** LOAD NECESSARY LIBRARY MODULES ***}

```
IMPORT IODECLARATIONS,
       GENERAL_0,
       IOCOMASM;
```

```
                    {*** SET PROGRAM CONSTANTS ***}

CONST Time_delay=380.0;      {AVERAGE TIME DELAY STORED WITH CAL DATA}


        {*** DECLARE FOUR LARGE EXTERNAL DATA ARRAYS AND POINTERS ***}

TYPE L1=ARRAY [1..5000] OF INTEGER;      {CO2 CHANNEL DATA ARRAY}
     PT1=^L1;      {POINTER TO ARRAY L1}
     L2=ARRAY [1..5000] OF INTEGER;      {O2 CHANNEL DATA ARRAY]
     PT2=^L2;      {POINTER TO ARRAY L2]
     L3=ARRAY [1..5000] OF INTEGER;      {FLOW CHANNEL DATA ARRAY}
     PT3=^L3;      {POINTER TO ARRAY L3}
     L4=ARRAY [1..5000} OF INTEGER;      {TEMPERATURE CHANNEL DATA ARRAY}
     PT4=^L4;      {POINTER TO ARRAY L4}


                    {*** DECLARE PROGRAM VARIABLES ***}
VAR S,Sam,TEMP,Co2_dc_offset,
    O2_dc_offset,Bin_zero_flow,
    GPINT[7077890]                          :       INTEGER;
    O1,Co2_cal,O2_cal,
    Non_C_I_flow_cal,Non_C_E_flow_cal,
    Corr_I_flow_cal,Corr_E_flow_cal,
    Ta,Tb,Tc                                :       REAL;
    NSTRING                                 :       STRING [4];
    Q                                       :       STRING[5];
    Date                                    :       STRING[25];
    Fname                                   :       STRING[16];
    RSTRING                                 :       STRING[25];
    F                                       :       TEXT;
    Line1: PT1;
    Line2: PT2;
    Line3: PT3;
    Line4: PT4;
    Corr_flag                               :       BOOLEAN;
    Corr_Incr_Vol,
    Non_C_Incr_Vol,
    Rel_humid,Pb,Room_temp                          :       REAL;
    Device_pointer                          :       INTEGER;
    Msi_devices                             :       ARRAY[1..3} OF STRING [4];


    {

*** DECLARE EXTERNAL 68000 ASSEMBLY MODULES
                                                                        }
PROCEDURE BIT_TST;EXTERNAL;      {WAITS UNTIL EOC BIT GOES HIGH, THEN LOW}
PROCEDURE BIT_HI;EXTERNAL;      {WAITS UNTIL EOC BIT IS LOW}


PROCEDURE BEEP;
{*************************************************************************
*************************************************************************
*                                                                       *
*       Procedure Name:    BEEP                                       *
*                                                                       *
*       File Name:         CAP2                                       *
*                                                                       *
*       REVISION                DATE            PROGRAMMER            *
*       --------                ---------       ----------            *
```

```
*          1.0                 22-Sept-84      Michael Masters.          *
*                                                                        *
*                                                                        *
*       Procedure discription:                                           *
*           This procedure sounds an audible beep to the operator.       *
*                                                                        *
*                                                                        *
*       Calling sequence:                                                *
*             BEEP;                                                      *
*                                                                        *
*                                                                        *
*       Parameters supplied by the calling routine.                      *
*             None                                                       *
*                                                                        *
*       Parameters returned to the calling routine.                      *
*             None                                                       *
*                                                                        *
*========================================================================*
**************************************************************************
                                                                         }
    BEGIN
    PROMPT (CHR(7));
    END;

{========================================================================}


PROCEDURE Line_feed(I    :       INTEGER);
{**************************************************************************
**************************************************************************
*                                                                        *
*       Procedure Name:   Line feed                                      *
*                                                                        *
*       File Name:        CAP2                                           *
*                                                                        *
*       REVISION          DATE            PROGRAMMER                     *
*       --------          ---------       ----------                     *
*         1.0             22-Sept-84      Michael Masters.               *
*                                                                        *
*       Procedure discription:                                           *
*           This procedure sends a specified number of line feeds to the *
*           terminal.                                                    *
*                                                                        *
*       Calling sequence:                                                *
*             Line_feed(I);                                              *
*                                                                        *
*       Parameters supplied by the calling routine.                      *
*                                                                        *
*               I - is an integer value parameter corresponding to the   *
*                   number of line feeds to be sent to the terminal.     *
*                                                                        *
*       Parameters returned to the calling routine.                      *
*             None                                                       *
*                                                                        *
*========================================================================*
**************************************************************************
                                                                         }
    VAR     J       :       INTEGER;
    BEGIN
    FOR J := 1 TO I DO          {Send the specified number of line feeds}
```

```
        BEGIN
        PROMPT (CHR(10));    {Send the line feed}
        END;
    END;
{=============================================================================}


{
*** DECLARE PROCEDURE FOR PAUSING PROGRAM OPERATION
}
PROCEDURE HOLD_UP;
  BEGIN
      PROMPT ('PRESS ENTER TO CONTINUE.');      {DISPLAY PROMPT ON CRT}
      BEEP;
      READLN;      {WAIT UNTIL 'ENTER' IS PRESSED}
  END; {HOLD_UP END}
{=============================================================================}


PROCEDURE Ask_Y_or_N(VAR Q:STRING);
{*****************************************************************************
*****************************************************************************
*                                                                           *
*       Procedure Name:   Ask yes or no question.                           *
*                                                                           *
*       File Name:        CAP2                                              *
*                                                                           *
*       REVISION              DATE            PROGRAMMER                    *
*       --------              ---------       ----------                    *
*         1.0                 22-Sept-84      Michael Masters.              *
*                                                                           *
*                                                                           *
*       Procedure discription:                                              *
*           This procedure prompts the operator for a yes or no response.   *
*                                                                           *
*                                                                           *
*       Calling sequence:                                                   *
*            Ask_Y_or_N(Q);                                                 *
*                                                                           *
*       Parameters supplied by the calling routine.                         *
*            None                                                           *
*                                                                           *
*       Parameters returned to the calling routine.                         *
*            Q - a string reference parameter.  Returns the value 'Y' for   *
*                a yes response and the value 'N' for a no response.        *
*                                                                           *
*                                                                           *
*===========================================================================*
*****************************************************************************
                                                                            }

    BEGIN
    REPEAT
        BEEP;
        READLN(Q);
        IF Q='y' THEN Q := 'Y';
        IF Q='n' THEN Q := 'N';
        IF NOT((Q='Y') OR (Q='N'))  THEN
            BEGIN
            WRITELN('Invalid response.  Enter Y (yes) or N (no).');
            END
        UNTIL (Q='Y') OR (Q='N') ;
        END; {Ask_Y_or_N}
```

```
{=============================================================================}
```

```
{
*** DECLARE PROCEDURE TO SET DAM CLOCK
                                                                        }
PROCEDURE CLKSET(VAR S:INTEGER);      {PASS SAMPLING FREQUENCY (S)}
   VAR X,Fm,F1:INTEGER;
   BEGIN
      {
      *** HAVE USER ENTER THE SAMPLING FREQUENCY
      }
      PROMPT ('ENTER SAMPLING FREQUENCY:  ');
      BEEP;
      READLN(S);
      {
      *** DETERMINE 16-BIT COUNTER VALUE FOR CLK1 IN 8253 TIMER CHIP
      }
      X:=1000000 DIV 2 DIV S;
      IF X>=256 THEN
                  BEGIN
                    Fm:=X DIV 256;
                    F1:=X-256*Fm;
                  END
              ELSE
                  BEGIN
                    Fm:=0;
                    F1:=X;
                  END;
      {
      *** SET COUNTER 0 IN 8253 TIMER TO MODE 3
      }
      IOCONTROL(12,3,15678); {11110100111110}
      IOCONTROL(12,3,15422); {11110000111110}
      IOCONTROL(12,3,15670); {11110100110110}
      {
      *** SET COUNTER 1 IN 8253 TIMER TO MODE 2
      }
      IOCONTROL(12,3,15740); {11110101111100}
      IOCONTROL(12,3,15484); {11110001111100}
      IOCONTROL(12,3,15732); {11110101110100}
      {
      *** LOAD LSB OF COUNTER 0
      }
      IOCONTROL(12,3,9474);  {10010100000010}
      IOCONTROL(12,3,9218);  {10010000000010}
      IOCONTROL(12,3,9474);  {10010100000010}
      {
      *** LOAD MSB OF COUNTER 0
      }
      IOCONTROL(12,3,9472);  {10010100000000}
      IOCONTROL(12,3,9216);  {10010000000000}
      IOCONTROL(12,3,9472);  {10010100000000}
      {
      *** LOAD LSB OF COUNTER 1
      }
      IOCONTROL(12,3,13568+F1); {11010100000000}
      IOCONTROL(12,3,13312+F1); {11010000000000}
      IOCONTROL(12,3,13568+F1); {11010100000000}
      {
      *** LOAD MSB OF COUNTER 1
```

```
      }
      IOCONTROL(12,3,13568+Fm); {11010100000000}
      IOCONTROL(12,3,13312+Fm); {11010000000000}
      IOCONTROL(12,3,13568+Fm); {11010100000000}
  END; {CLKSET END}
{
*** DECLARE PROCEDURE FOR COLLECTING FOUR CHANNELS OF DAM DATA
}
PROCEDURE DATA_COLLECT(VAR Sam,S:INTEGER;
                       VAR LINE1:PT1;VAR LINE2:PT2;
                       VAR LINE3:PT3;VAR LINE4:PT4);
  VAR I,R6,De1,R4:INTEGER;

        {BIT PATTERNS NECESSARY TO SET THE CHANNEL MULTIPLEXER}
  CONST Chna=15359;Chnb=13311;
        Chnc=11263;Chnd=9215;

       Mask=4095;      {MASKS OFF ALL BUT 12 DATA BITS IN STATUS WORD}
  {
  *** MAIN LOOP FOR FOUR CHANNEL DATA ACQUISITION
  }
  BEGIN {DATA_COLLECT}
     WRITELN('NOW COLLECTING DATA... please wait patiently.');
     WRITELN('Collecting ',Sam:5,' samples.
              This will take ',Sam DIV S:4,' Sec');
     FOR I:=1 TO Sam DO
        BEGIN
           {
           *** PUT S/H AMPS IN TRACKING MODE AND SELECT CHANNEL A
           }
           R6:=BINAND(15360,Chna);
           R6:=BINCMP(R6);
           IOCONTROL(12,3,R6);
           {
           *** GIVE S/H AMPS TIME TO TRACK INPUT SIGNALS
           }
           De1:=15;
           WHILE De1>0 DO
              BEGIN
                 De1:=De1-1;
              END;
           {
           *** SELECT CHANNEL A ON MULTIPLEXER AND CONVERT SIGNAL HIGH
           }
           R6:=BINAND(7168,Chna);
           R6:=BINCMP(R6);
           IOCONTROL(12,3,R6);
           {
           *** SELECT CHANNEL A ON MULTIPLEXER AND CONVERT SIGNAL LOW
           }
           R6:=BINAND(7680,Chna);  {SEND CONVERT...}
           R6:=BINCMP(R6);         {...PULSE}
           IOCONTROL(12,3,R6);
           {
           *** SELECT CHANNEL A ON MULTIPLEXER AND CONVERT SIGNAL HIGH
           }
           R6:=BINAND(7168,Chna);  {RETURN TO...}
           R6:=BINCMP(R6);         {...NORMAL}
           IOCONTROL(12,3,R6);
           {
           *** WAIT FOR EOC LINE TO GO LOW
           }
           BIT_HI;
           {
           *** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
```

```
}
R4:=IOSTATUS(12,3);
R4:=R4 DIV 4;
R4:=BINAND(MASK,R4);
Line1^[I]:=R4;
{
*** SELECT CHANNEL B ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnb);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL B ON MULTIPLEXER AND CONVERT SIGNAL LOW
}
R6:=BINAND(7680,Chnb);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL B ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnb);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** WAIT FOR EOC SIGNAL TO GO LOW
}
BIT_HI;
{
*** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
}
R4:=IOSTATUS(12,3);
R4:=R4 DIV 4;
R4:=BINAND(Mask,R4);
Line2^[I]:=R4;
{
*** SELECT CHANNEL C ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnc);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL C ON MULTIPLEXER AND CONVERT SIGNAL LOW
}
R6:=BINAND(7680,Chnc);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL C ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnc);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** WAIT FOR EOC SIGNAL TO GO LOW
}
BIT_HI;
{
*** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
}
R4:=IOSTATUS(12,3);
R4:=R4 DIV 4;
R4:=BINAND(Mask,R4);
Line3^[I]:=R4;
{
*** SELECT CHANNEL D ON MULTIPLEXER AND CONVERT SIGNAL HIGH
```

```
            }
            R6:=BINAND(7168,Chnd);
            R6:=BINCMP(R6);
            IOCONTROL(12,3,R6);
            {
            *** SELECT CHANNEL D ON MULTIPLEXER AND CONVERT SIGNAL LOW
            }
            R6:=BINAND(7680,Chnd);
            R6:=BINCMP(R6);
            IOCONTROL(12,3,R6);
            {
            *** SELECT CHANNEL D ON MULTIPLEXER AND CONVERT SIGNAL HIGH
            }
            R6:=BINAND(7168,Chnd);
            R6:=BINCMP(R6);
            IOCONTROL(12,3,R6);
            {
            *** WAIT FOR EOC SIGNAL TO GO LOW
            }
            BIT_HI;
            {
            *** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
            }
            R4:=IOSTATUS(12,3);
            R4:=R4 DIV 4;
            R4:=BINAND(Mask,R4);
            Line4^[I]:=R4;
            {
            *** WAIT FOR EOC SIGNAL TO GO HIGH, THEN LOW
            }
            BIT_TST;
          {
          *** LOOP BACK UNTIL ALL POINTS ARE COLLECTED
          }
          END;
  END; {DATA_COLLECT}
{
*** DECLARE PROCEDURE FOR CALIBRATING THE PERKIN-ELMER GAS MASS SPECTROMETER
}
PROCEDURE GASCAL(VAR O1,Co2_cal,O2_cal          :       REAL;
                 VAR Co2_dc_offset,O2_dc_offset :       INTEGER;
                     Sam,S                      :       INTEGER);
  VAR I,Avg_CO2_room_air,Avg_O2_room_air,
      Avg_CO2_cal_gas,Avg_O2_cal_gas            :       INTEGER;
      C1,Ch,Oh                                  :       REAL;
  {
  *** BEGIN GMS CALIBRATION
  }
  BEGIN {GASCAL}
     {
     *** INSTRUCT USER TO CONNECT GMS PROBE FOR 21% O2 AND 0% CO2
     }
     WRITELN('CONNECT THE MASS SPECTROMETER PROBE TO ROOM AIR.');
     Line_feed(1);
     {
     *** OBTAIN ACTUAL O2 AND CO2 CONCENTRATIONS FROM GMS FRONT PANEL
     }
     PROMPT ('ENTER ACTUAL PERCENT CO2 CONCENTRATION.    ');
     BEEP;
     READLN(C1);
     C1 := C1/100.0;
     PROMPT ('ENTER ACTUAL PERCENT O2 CONCENTRATION.     ');
     BEEP;
     READLN(Oh);
     Oh := Oh/100.0;
```

```
{
*** FOLLOWING A CARRIAGE RETURN, TAKE 500 DATA POINTS ON CO2 AND O2
*** CHANNELS
}
HOLD_UP;
DATA_COLLECT(Sam,S,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE AVERAGE BINARY VALUES FOR O2 AND CO2 CHANNELS
}
Avg_CO2_room_air:=LINE1^[1];
Avg_O2_room_air:=LINE2^[1];
FOR I:=2 TO Sam DO
    BEGIN
      Avg_CO2_room_air:=Avg_CO2_room_air+LINE1^[I];
      Avg_O2_room_air:=Avg_O2_room_air+LINE2^[I];
    END;
Avg_CO2_room_air:=Avg_CO2_room_air DIV Sam;
Co2_dc_offset:=Avg_CO2_room_air;
Avg_O2_room_air:=Avg_O2_room_air DIV Sam;
{
*** DISPLAY AVERAGES ON HP9826 CRT
}
Line_feed(1);
WRITELN ('Average binary values read for room air');
WRITELN('     CO2: ',Avg_CO2_room_air);
WRITELN('      O2: ',Avg_O2_room_air);
{
*** INSTRUCT THE USER TO CONNECT GMS PROBE TO THE CALIBRATIO GAS.
}
WRITELN('CONNECT THE GMS PROBE TO THE CALIBRATION GAS.');
Line_feed(1);
{
*** OBTAIN ACTUAL O2 AND CO2 CONCENTRATIONS FROM GMS FRONT PANEL
}
PROMPT ('ENTER ACTUAL PERCENT CO2 CONCENTRATION.  ');
BEEP;
READLN(Ch);
Ch := Ch / 100.0;
PROMPT ('ENTER ACTUAL PERCENT O2 CONCENTRATION.   ');
BEEP;
READLN(O1);
O1 := O1 /100.0;
{
*** TAKE 500 DATA POINTS ON CO2 AND O2 CHANNELS
}
HOLD_UP;
DATA_COLLECT(Sam,S,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE AVERAGE BINARY VALUES FOR O2 AND CO2 CHANNELS
}
Avg_CO2_cal_gas:=LINE1^[1];
Avg_O2_cal_gas:=LINE2^[1];
FOR I:=2 TO Sam DO
    BEGIN
      Avg_CO2_cal_gas:=Avg_CO2_cal_gas+LINE1^[I];
      Avg_O2_cal_gas:=Avg_O2_cal_gas+LINE2^[I];
    END;
Avg_CO2_cal_gas:=Avg_CO2_cal_gas DIV Sam;
Avg_O2_cal_gas:= Avg_O2_cal_gas DIV Sam;
O2_dc_offset:=Avg_O2_cal_gas;
{
*** DISPLAY AVERAGES ON HP9826 CRT
Line_feed(1);
}
WRITELN ('Average binary values read for room air');
```

```
      WRITELN('     CO2: ',Avg_CO2_cal_gas);
      WRITELN('      O2: ',Avg_O2_cal_gas);
      {
      *** COMPUTE CO2 AND O2 CALIBRATION FACTORS FROM THESE AVERAGES
      }
      Co2_cal :=
          ROUND((Ch-C1)/(Avg_CO2_cal_gas-Avg_CO2_room_air)*10000000)/10000000;
      O2_cal :=
          ROUND((Oh-O1)/(Avg_O2_room_air-Avg_O2_cal_gas)*10000000)/10000000;
      {
      *** DISPLAY CO2 AND O2 CALIBRATION FACTORS ON 9826 CRT
      }
      WRITELN('CO2 DC OFFSET =',Co2_dc_offset);
      WRITELN('O2 DC OFFSET =',O2_dc_offset);
      WRITELN('CO2 CALIBRATION FACTOR =',Co2_cal);
      WRITELN('O2 CALIBRATION FACTOR =',O2_cal);
    END; {GASCAL}
{
-------------------------------------------------------------------------------
                                                                            }
PROCEDURE FLOWCAL(VAR Non_C_I_flow_cal,Non_C_E_flow_cal,
                      Corr_I_flow_cal,Corr_E_flow_cal    :      REAL;
                  VAR Bin_zero_flow                      :      INTEGER;
                      Sam,S                              :      INTEGER;
                      Corr_flag                          :      BOOLEAN;
                      Rel_humid,Pb,Room_temp,Ta,Tb,Tc    :      REAL);
{*****************************************************************************
*                                                                            *
*       Procedure Name:  FLOWCAL                                             *
*                                                                            *
*       File Name:       CAP2                                                *
*                                                                            *
*       REVISION              DATE              PROGRAMMER                   *
*       --------              ---------         ----------                   *
*         1.0                 June 28, 1984     Loren Riblett Jr.            *
*         2.0                 Sept. 6, 1984     Michael Masters              *
*                                                                            *
*       Procedure discription:                                               *
*              FLOWCAL calculates the calibration factors for the flow       *
*              signal.  The binary value of zero flow is first determined    *
*              and then two sets of flow calibration factors are det-        *
*              ermined.  The first two flow cals do not use any temper-      *
*              ature or gas viscosity corrections in their determination.    *
*              The second set uses incremental point by point temperature    *
*              and viscosity correction in their determination.              *
*                                                                            *
*                                                                            *
*       Parameters supplied by the calling routine.                          *
*              (all parameters passed to FLOWCAL are value parameters.)      *
*                                                                            *
*              Sam        -   The number of samples which is collected.      *
*                                                                            *
*              S          -   Sampling frequency.                           *
*                                                                            *
*              Corr_flag  -   Indicates if temperature and viscosity corr-   *
*                             corrections are requested.                     *
*                               TRUE  - correct for temp. and visc.          *
*                               FALSE - no corrections.                      *
*                                                                            *
*              Rel_humid  -   The relative humidty of the ambient air.       *
*                                                                            *
*              Pb         -   The ambient barometric pressure.              *
*                                                                            *
*                                                                            *
```

```
*                                                                       *
*     - Parameters returned to the calling routine.                     *
*             (all parameters returned are reference parameters.)       *
*                                                                       *
*             Non_C_I_flow_cal - the inspiratory flow calibration factor *
*                     which does not use temp. and visc. corrections.   *
*                                                                       *
*             Non_C_E_flow_cal - the expiratory flow calibration factor *
*                     which does not use temp. and visc. corrections.   *
*                                                                       *
*             Corr_I_flow_cal -  The inspiratory flow calibration factor *
*                     which uses the temp. and visc. corrections.       *
*                                                                       *
*             Corr_E_flow_cal -  The expiratory flow calibration factor *
*                     which uses the temp. and visc. corrections.       *
*                                                                       *
*             Bin_zero_flow  -  The binary value corresponding to zero  *
*                     flow input.                                       *
*                                                                       *
*      Routines called:                                                 *
*                                                                       *
*            Calc_incr_vol - calculates the incremental volume of the   *
*                     sample.                                           *
*                                                                       *
*            Valid_pointer - Sees if the pointer to the vapor pressure  *
*                     array is valid.                                   *
*                                                                       *
*=======================================================================*}

CONST   Non_C_Pump_Vol = 0.647;         {The volume of the pump under ambient
                                         conditions.                    }
        Corr_Pump_Vol  = 0.560;         {The volume of the pump under STPD
                                         conditions.                    }
        Min_Temp  =  20; {degrees C.    The minumum temperature for which
                                          the saturated vapor pressure may be
                                          found in the vapor pressure array.}
        Max_Temp  =  45; {degrees C.    The maximum temperature for which
                                          the saturated vapor pressure may be
                                          found in the vapor pressure array.}
        First_element = 0; {            The first element in the vapor
                                         pressure file.}
        Last_element  = 10 * (Max_Temp - Min_Temp);
                         {              The last element in the vapor
                                         pressure file.}
        Vap_File_name = '#14:VAP_PASCAL'; {The vapor pressure file location
                                           and name.}
VAR 1,Tot_zero,No_breaths,A,
    Array_pointer,Bin_delta_P           :       INTEGER;
    Still_calculating,Still_inspire,
    Still_expire                        :       BOOLEAN;
    T,Non_C_Tot_vol_insp,
    Non_C_Tot_vol_expr,
    Corr_Tot_vol_insp,
    Corr_Tot_vol_expr,
    Non_C_Airi,Non_C_Aire,
    Corr_Airi,Corr_Aire,
    Non_C_Avg_Vol_insp,
    Non_C_Avg_Vol_expr,
    Corr_Avg_Vol_insp,
    Corr_Avg_Vol_expr,Fh2o,
    Temperature             :       REAL;
    Vap         :       ARRAY[First_element..Last_element] OF REAL;
    Vap_File                        :       FILE OF REAL;
```

```
PROCEDURE  Valid_pointer ( VAR Array_pointer     :       INTEGER);
{*****************************************************************************
*****************************************************************************
*                                                                           *
*       Procedure Name:  Valid_pointer                                      *
*                                                                           *
*       File Name:       CAP2                                               *
*                                                                           *
*       REVISION                DATE            PROGRAMMER                  *
*       --------                ---------       ----------                  *
*         1.0                   22-Sept-84      Michael Masters             *
*       Program discription:                                                *
*                                                                           *
*              This procedure determines if the vapor pressure array        *
*              pointer is within the proper range.                          *
*                                                                           *
*                                                                           *
*       Calling sequence:                                                   *
*                                                                           *
*            Valid_pointer(Array_pointer);                                  *
*                                                                           *
*       Parameters passed from and returned to the calling routine.         *
*                                                                           *
*              Array_pointer  -  The pointer into the vapor pressure        *
*                  array.  If it is not within the valid range it is made   *
*                  to point at either the first element or the last         *
*                  element. ( A reference parameter.)                       *
*                                                                           *
*===========================================================================*
*****************************************************************************
                                                                            }
    BEGIN
    IF Array_pointer < First_element THEN       { The pointer is invalid.}

        BEGIN
        Array_pointer := First_element;
        END

    ELSE                                        { The pointer may be valid.}

        BEGIN
        IF Array_pointer > Last_element THEN
            BEGIN
            { The pointer is invalid.}
            Array_pointer := Last_element;
            END

        ELSE
            BEGIN
{           The vapor pressure pointer is valid so leave it alone.}
            END;
        END;
    END; {Valid_pointer}
{
-----------------------------------------------------------------------------
                                                                            }


PROCEDURE Calc_Incr_vol(Bin_delta_P                     :       INTEGER;
                        Fh2o,Temperature,T              :       REAL;
                    VAR Corr_Incr_vol,Non_C_Incr_vol    :       REAL);
{*****************************************************************************
```

```
*************************************************************************
*                                                                       *
*       Program Name:   Calc_Incr_vol                                   *
*                                                                       *
*       File Name:      CAP2                                            *
*                                                                       *
*       REVISION                DATE            PROGRAMMER              *
*       --------                ---------       ----------              *
*         1.0                   22-Sept-84      Michael Masters.        *
*                                                                       *
*                                                                       *
*       Program discription:                                            *
*                                                                       *
*       This proceedure calculates the incremental volume of the sample *
*       from the measured binary differential pressure.  The non corrected *
*       flow is assumed to be directly propotional to the pressure differ- *
*       ential so that the binary incremental volume is equal to the    *
*       sampled value multiplied by the sampling period.                *
*                                                                       *
*       The corrected volume is calculated by dividing the differential *
*       pressure by the relative viscosity of the gas to obtain the binary *
*       flow.  The incremental volume is then calculated by multiplying by *
*       the sampling period.  This volume is then converted to STPD cond- *
*       itions.                                                         *
*                                                                       *
*       If temperature and viscosity corrections are not desired then the *
*       corrected volume is set equal to the non corrected volume.      *
*                                                                       *
*       Calling sequence:                                               *
*            Calc_Incr_vol( Bin_delta_P,Pb,Fh2o,Temperature,T           *
*                           Corr_Incr_vol,Non_C_Incr_vol );             *
*                                                                       *
*                                                                       *
*       Parameters supplied by the calling routine.                     *
*                 (all parameters are value parameters)                 *
*                                                                       *
*           Bin_delta_P     - The sampled pressure differential from LINE3 *
*                                                                       *
*           Fh2o            - The fractional content of water vapor.    *
*                                                                       *
*           Temperature     - The temperature of the flow gas obtained  *
*                             from the temperature transducer.          *
*                                                                       *
*           T               - The sampling period.                      *
*                                                                       *
*                                                                       *
*       Parameters returned to the calling routine.                     *
*                 (all parameters are reference parameters)             *
*                                                                       *
*           Corr_Incr_vol   - The calculated incremental volume under STPD *
*                             conditions which was obtained using vis-  *
*                             cosity corrections on the measured pressure *
*                             differential.  If corrections are not     *
*                             requested then it is equal to the non corr- *
*                             ected volume.                             *
*                                                                       *
*           Non_C_Incr_vol  - The non corrected incremental volume.     *
*                                                                       *
*       Routines called:                                                *
*           The function Rel_viscos.                                    *
*                                                                       *
*=======================================================================*
*************************************************************************
                                                                        }
  VAR Rel_viscosity,ITPnS_to_STPD       :       REAL;
```

```
FUNCTION Rel_viscos ( Temperature,Fh2o    :    REAL)   :   REAL;
{*************************************************************************
*************************************************************************
*                                                                       *
*       Function Name:  Rel_viscos                                      *
*                                                                       *
*       File Name:          CAP2.TEXT                                   *
*                                                                       *
*       REVISION                DATE            PROGRAMMER              *
*       --------                ---------       ----------              *
*         1.0                   Sept 6, 1984    Michael Masters         *
*                                                                       *
*       Function discription:                                           *
*           Rel_viscos returns the real value equal to the relative vis-*
*           cosity of the gas.                                          *
*                                                                       *
*       Parameters supplied by the calling routine.                     *
*             (all parameters are value parameters.)                    *
*                                                                       *
*              Temperature - The temperature in degrees Centigrade.     *
*                                                                       *
*              Fh2o        -  The  fraction of water vapor in the room air.*
*                                                                       *
*=======================================================================*
*************************************************************************}

    VAR     Vis_H2O,Vis_O2,Vis_CO2,
            Vis_N2,Vis_STPD,FO2,FCO2,FN2             :       REAL;
        BEGIN {FUNCTION Rel_viscos}
        Vis_STPD := 170.17502;

{               Calculate the fractional concentrations of oxygen,
                carbon dioxide, and nitrogen in the wet room air.}

        FO2  := 0.2109 * ( 1.0 - FH2O);
        FCO2 := 0.0004 * ( 1.0 - FH2O);
        FN2  := (1.0 - FO2 - FCO2 - FH2O);

{
                        Calculate the individual viscosities.
                                                                }
        Vis_N2  := 165.4 + 0.451*Temperature;
        Vis_O2  := 188.0 + 0.606*Temperature;
        Vis_CO2 := 187.1 + 0.612*Temperature;
        Vis_H2O := 117.6 + 0.506*Temperature;
{
                Calculate the relative gas viscosity.
                                                                }
        Rel_viscos := (FO2*Vis_O2 + FCO2*Vis_CO2
                        + FN2*Vis_N2 + FH2O*Vis_H2O)/Vis_STPD;

        END;    {FUNCTION Rel_viscos}
{=======================================================================}

    BEGIN {Calc_Incr_Vol}
```

```
    Non_C_Incr_Vol := Bin_delta_P * T;
    IF Corr_flag    THEN
        BEGIN
        Rel_viscosity := Rel_viscos( Temperature,Fh2o );

                        {Calculate the ITPnS to STPD  conversion factor.}
        ITPnS_to_STPD :=
                (1.0 - Fh2o)*( 273.15/(Temperature + 273.15) )*(Pb/760);

        Corr_Incr_Vol := Bin_delta_P / Rel_Viscosity * T;
        Corr_Incr_Vol := Corr_Incr_Vol * ITPnS_to_STPD;
        END
    ELSE
        BEGIN
        Corr_Incr_Vol := Non_C_Incr_Vol;
        END;

    END;  {Calc_Incr_Vol}
{=============================================================================
=============================================================================}



                    {*** BEGIN FLOW CALIBRATION}
    BEGIN {FLOWCAL}


    IF (Corr_flag) THEN         {Read the water vapor pressure table.}

        BEGIN
        RESET (Vap_File,Vap_File_name);
        FOR Array_pointer := First_element TO Last_Element        DO
            BEGIN
            READ (Vap_File, Vap[Array_pointer]);
            END;
        END;


                {Determine the binary value corresponding to zero flow.}

    Line_feed(2);
    WRITELN ('You can determine the binary zero flow or use a');
    WRITELN ('default value of 2038.');
    PROMPT ('Do you wish to determine binary zero flow? (Y/N)');
    Ask_Y_or_N(Q);
    IF Q='Y' THEN
        BEGIN

            {*** INSTRUCT USER TO APPLY ZERO FLOW TO THE PNEUMOTACHOMETER}

        WRITELN('CONNECT ZERO FLOW TO PNEUMOTACHOMETER.');


           {*** FOLLOWING A CARRIAGE RETURN, GO COLLECT 500 FLOW DATA POINTS}

       HOLD_UP;
       DATA_COLLECT(Sam,S,LINE1,LINE2,LINE3,LINE4);


            {*** AVERAGE THE 500 DATA POINTS FOR ZERO FLOW VALUE}

       Tot_zero:=LINE3^[1];
```

```
    FOR I:=2 TO Sam DO
        BEGIN
        Tot_zero:=Tot_zero+LINE3^[I];
        END;

    Bin_zero_flow:=Tot_zero DIV Sam;


            {*** DISPLAY BINARY ZERO FLOW VALUE ON 9826 CRT}

    WRITELN('Average binary value for zero flow =',Bin_zero_flow);
    END


ELSE

     BEGIN
     Bin_zero_flow := 2038;
     END;


            {Calibrate the flow signal by first integrating the measured
            differential pressure signal over each inspiratory and expir-
            atory cycle.  The average inspiratory and expiratory binary
            volumes are then set equal to the known pump volume with the
            calibration factor.}

          {*** INSTRUCT USER TO CONNECT PNEUMOTACHOGRAPH TO HARVARD PUMP}
Line_feed(2);
IF Corr_flag THEN
    BEGIN
    WRITELN('INSERT THE THERMOCOUPLE INTO THE PTM HEAD');
    END;

WRITELN('CONNECT PUMP FLOW TO THE PNEUMOTACHOGRAPH.');
Line_feed(1);


    {*** FOLLOWING A CARRIAGE RETURN, GO COLLECT 4000 DATA POINTS
         OF PUMP FLOW}

HOLD_UP;
Sam:=4000;
DATA_COLLECT(Sam,S,LINE1,LINE2,LINE3,LINE4);


    { INSTRUCT USER DATA COLLECTION IS COMPLETE AND BEGIN INTEGRATION
      OF FLOW SIGNAL TO DETERMINE INSPIRATORY AND EXPIRATORY FLOW
      CALIBRATION}

BEEP;
WRITELN('DATA COLLECTION COMPLETE... turn off the pump.');


        { INITIALIZE NECESSARY VARIABLES FOR FLOW SIGNAL INTEGRATION}
No_breaths:=0;
T:=1/S;
A:=1;
Non_C_Tot_Vol_insp := 0;
Non_C_Tot_Vol_expr := 0;
Corr_Tot_Vol_insp  := 0;
Corr_Tot_Vol_expr  := 0;
```

```
    { PRINT HEADER ON CRT FOR BREATH-BY-BREATH INTEGRATION DISPLAY}

WRITELN('     Non Corrected Values     | Corrected Values ');
WRITELN('------------------------------|------------------');
WRITELN('              BINARY     BINARY  | BINARY      BINARY');
WRITELN('BREATH        AIR        AIR   |  AIR          AIR');
WRITELN('NUMBER     INSPIRED   EXPIRED  |INSPIRED     EXPIRED');
WRITELN('------------------------------|------------------');


Still_calculating := TRUE;
REPEAT     { MAKE SURE FLOW INDEX IS AT BEGINNING OF FIRST INSPIRATION}
    A:=A+1;
    IF A > Sam-1 THEN
        BEGIN
        Still_calculating := FALSE;
        END;
UNTIL (LINE3^[A-1] > Bin_zero_flow) AND (LINE3^[A] < Bin_zero_flow)
            OR NOT (Still_Calculating);


IF Corr_flag THEN
    BEGIN
    Array_pointer := ROUND ((Room_temp - Min_Temp) * 10);
    Valid_Pointer (Array_pointer);
    Fh2o := Rel_humid * Vap[Array_pointer] / Pb;
    END;

WHILE  Still_calculating       DO
    BEGIN
    IF A> Sam-10 THEN
        BEGIN
        Still_calculating := FALSE;
        END;


    {Calculate the binary inspiratory volume of this stroke (breath).}

    IF Still_calculating  THEN
        BEGIN

        {Compute half of the first trapezoidal area.}
        Temperature := Ta*Line4^[A]*Line4^[A] + Tb*Line4^[A] + Tc;
        Bin_delta_P := Line3^[A]-Bin_zero_flow;
        Calc_Incr_vol(Bin_delta_P,Fh2o,Temperature,T,
                      Corr_Incr_vol,Non_C_Incr_vol);
        Corr_Airi :=  0.5 * Corr_Incr_Vol;
        Non_C_Airi := 0.5 * Non_C_Incr_vol;
        END;


            { SUM UP ENTIRE INSPIRATION TRAPEZOIDS}
    Still_inspire := TRUE;
    WHILE Still_inspire  AND Still_calculating    DO
        BEGIN
        A:=A+1;
        IF A>Sam THEN      {data has been exhausted, quit calculating}
            BEGIN
            Still_calculating := FALSE;
            END

        ELSE    {Data is still available for calculation.}

            BEGIN
            Bin_delta_P := Line3^[A]-Bin_zero_flow;
```

```
            IF (Bin_delta_P >= 0) THEN      {The ispiration is over}
                BEGIN
                Still_inspire := FALSE
                END

            ELSE      {The pump is still inspiring, compute
                                        the incremental volume.}
                BEGIN
                Temperature := Ta*Line4^[A]*Line4^[A] + Tb*Line4^[A] +Tc;
                Calc_Incr_vol(Bin_delta_P,Fh2o,Temperature,T,
                Corr_Incr_vol,Non_C_Incr_vol);

                      {Add the incremental volume to the total}
                Corr_Airi := Corr_Airi + Corr_Incr_Vol;
                Non_C_Airi := Non_C_Airi + Non_C_Incr_Vol;
                END;
            END;
        END;  {While loop}


    IF Still_calculating  THEN
        {Place half of the volume corresponding to inspiratory/expir-
         atory transition on the inspired volume and half on the
         expired volume.}


        BEGIN
        Temperature := Ta*Line4^[A]*Line4^[A] + Tb*Line4^[A] + Tc;
        Bin_delta_P := Line3^[A]-Bin_zero_flow;
        Calc_Incr_vol(Bin_delta_P,Fh2o,Temperature,T,
                      Corr_Incr_vol,Non_C_Incr_vol);

        Corr_Airi := Corr_Airi + (1/2) * Corr_Incr_Vol;
        Non_C_Airi := Non_C_Airi + (1/2) * Non_C_Incr_Vol;


        Corr_Aire  := (1/2) * (Corr_Incr_Vol);
        Non_C_Aire := (1/2) * (Non_C_Incr_Vol);
        END;


                { **** SUM UP ENTIRE EXPIRATION TRAPEZOIDS ****}
    Still_expire := TRUE;
    WHILE Still_expire AND Still_calculating    DO
        BEGIN
        A:=A+1;
        IF A>Sam THEN     {The data has been exhausted}
            BEGIN
            Still_calculating := FALSE;
            END

        ELSE     {Data is still available for calculation.}

            BEGIN
            Bin_delta_P := Line3^[A]-Bin_zero_flow;
            IF (Bin_delta_P <= 0) THEN    {The expiration is over}
                BEGIN
                Still_expire := FALSE;
                END

            ELSE   {The pump is still expiring, compute the
                                        incremental volume.}
                BEGIN
                Temperature := Ta*Line4^[A]*Line4^[A] + Tb*Line4^[A] +Tc;
```

```
                Calc_Incr_vol(Bin_delta_P,Fh2o,Temperature,T,
                              Corr_Incr_vol,Non_C_Incr_vol);

                    {Add the incremental volume to the total}
                Corr_Aire := Corr_Aire + Corr_Incr_Vol;
                Non_C_Aire := Non_C_Aire + Non_C_Incr_Vol;
                END;
            END;
        END;    {End While loop}


    IF Still_calculating THEN
        {Place half of the volume corresponding to the expiratory-
         inspiratory transition on the total expired volume.}
        BEGIN
        Temperature := Ta*Line4^[A]*Line4^[A] + Tb*Line4^[A] + Tc;
        Bin_delta_P := Line3^[A]-Bin_zero_flow;
        Calc_Incr_vol(Bin_delta_P,Fh2o,Temperature,T,
                      Corr_Incr_vol,Non_C_Incr_vol);
        Corr_Aire := Corr_Aire + 0.5 * Corr_Incr_Vol;
        Non_C_Aire := Non_C_Aire + 0.5 * Non_C_Incr_Vol;

        {   Make the calculations for this pump stroke.'}

        No_breaths:=No_breaths+1;

        { **** UPDATE TOTAL VOLUMES INSPIRED AND EXPIRED ****}
        Corr_Tot_vol_insp  := Corr_Tot_vol_insp  + Corr_Airi;
        Non_C_Tot_vol_insp := Non_C_Tot_vol_insp + Non_C_Airi;
        Corr_Tot_vol_expr  := Corr_Tot_vol_expr  + Corr_Aire;
        Non_C_Tot_vol_expr := Non_C_Tot_vol_expr + Non_C_Aire;


        {** DISPLAY INSPIRED AND EXPIRED VOLUMES FOR THIS STROKE **}
        WRITELN(' ',No_breaths:3,'       ',Non_C_Airi:6:1,
                '     ',Non_C_Aire:6:1,'     ',Corr_Airi:6:1,
                '     ',Corr_Aire:6:1);
        END;


    END;        { End Main integration Loop.}


  {*** COMPUTE AVERAGE VOLUMES INSPIRED AND EXPIRED (PER BREATH BASIS)***}
Non_C_Avg_Vol_insp := Non_C_Tot_vol_insp/No_breaths;
Non_C_Avg_Vol_expr := Non_C_Tot_vol_expr/No_breaths;
Corr_Avg_Vol_insp  := Corr_Tot_vol_insp/No_breaths;
Corr_Avg_Vol_expr  := Corr_Tot_vol_expr/No_breaths;


      { Compute the inspiratory and expiratory flow calibration factors
        for the non corrected average Binary pump volume.}
Non_C_I_flow_cal := Non_C_Pump_Vol / (-1*Non_C_Avg_Vol_insp);
Non_C_E_flow_cal := Non_C_Pump_Vol / (Non_C_Avg_Vol_expr);


      {Compute the inspiratory and expiratory flow calibration factors
       for the corrected average Binary pump volume.}
Corr_I_flow_cal := Corr_Pump_Vol / (-1*Corr_Avg_Vol_insp);
Corr_E_flow_cal := Corr_Pump_Vol / (Corr_Avg_Vol_expr);


        {**** DISPLAY FLOW CALIBRATIONS ON HP9826 CRT  ****}
WRITELN('BINARY ZERO FLOW =',Bin_zero_flow);
Line_feed(2);
```

```
    WRITELN('Inspiratory and expiratory calibration factors');
    WRITELN('for the NON CORRECTED flows.');
    WRITELN('      Insp. flow calibration factor =',Non_C_I_flow_cal);
    WRITELN('      Expr. flow calibration factor =',Non_C_E_flow_cal);
    Line_feed(2);
    WRITELN('Inspiratory and expiratory calibration factors');
    WRITELN('for the CORRECTED flows.');
    WRITELN('      Insp. flow calibration factor =',Corr_I_flow_cal);
    WRITELN('      Exp. flow calibration factor =',Corr_E_flow_cal);
END; {FLOWCAL}
{


*** DECLARE PROCEDURE FOR CALIBRATION OF TEMPERATURE TRANSDUCER
}
PROCEDURE TEMPCAL(VAR Ta,Tb,Tc   :       REAL;
                  VAR Sam,S      :       INTEGER);

  CONST Thermometer_cal_Slope = 1; {The Thermometer calibration factor.
                                    This is the slope of the calibration curve
                                    This is determined by calibrating the cal-
                                    ibration thermometer with the Standard
                                    Thermometer which is available from Anat.
                                    and Phys.}
        Thermometer_cal_Y_int = -0.35; {The y intercept of the Calibration
                                        curve of the calibration thermometer.}


  TYPE TWODIM=ARRAY [1..3,1..3] OF REAL;
  VAR Tlow,Tmid,Thigh,Tlow_bin,Tmid_bin:REAL;
      Thigh_bin,Norm_a:REAL;
      I,Tot_temp:INTEGER;
      Bin_temp,Bin_temp_inv:TWODIM;
  {
  *** BEGIN TEMPERATURE CALIBRATION ROUTINE
  }
  BEGIN {TEMPCAL}
       {
       *** INSTRUCT USER TO PLACE TC IN LOW TEMPERATURE WATER BATH
       }
       WRITELN('PUT THERMOCOUPLE IN THE LOWEST TEMP WATER BATH.');
       {
       *** GET ACTUAL TEMPERATURE OF LOW TEMPERATURE WATER BATH
       }
       WRITELN('ENTER THE THE THERMOMETER READING (deg C).');
       BEEP;
       READLN(Tlow);
       {
                      Determine the true temperature.
                                                                  }
       Tlow := Tlow * Thermometer_cal_Slope + Thermometer_cal_Y_int;


       {
       *** GO TAKE 500 DATA POINTS ON THE TEMPERATURE CHANNEL
       }
       Sam:=500;
       DATA_COLLECT(Sam,S,LINE1,LINE2,LINE3,LINE4);
       {
       *** COMPUTE THE AVERAGE VALUE FOR THE 500 TEMPERATURE DATA POINTS
       }
       Tot_temp:=0;
       FOR I:=1 TO Sam DO
          BEGIN
```

```
      Tot_temp:=Tot_temp+LINE4^[I];
    END;
Tlow_bin:=Tot_temp/Sam;
{
*** DISPLAY THE AVERAGE VALUE ON THE HP9826 CRT
}
WRITELN('AVERAGE BINARY VALUE READ FOR ',Tlow:2:2,' deg C');
WRITELN('IS:  ',Tlow_bin:4:1);
{
*** INSTRUCT USER TO PLACE TC IN MIDDLE TEMPERATURE WATER BATH
}
WRITELN('PUT THERMOCOUPLE IN THE MIDDLE TEMP WATER BATH.');
{
*** GET ACTUAL TEMPERATURE OF MIDDLE TEMPERATURE WATER BATH
}
WRITELN('ENTER THE THE THERMOMETER READING (deg C).');
BEEP;
READLN(Tmid);
{
                Determine the true temperature.
                                                        }
Tmid := Tmid * Thermometer_cal_Slope + Thermometer_cal_Y_int;

{
*** GO TAKE 500 DATA POINTS ON THE TEMPERATURE CHANNEL
}
DATA_COLLECT(Sam,S,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE THE AVERAGE VALUE FOR THE 500 TEMPERATURE DATA POINTS
}
Tot_temp:=0;
FOR I:=1 TO Sam DO
    BEGIN
      Tot_temp:=Tot_temp+LINE4^[I];
    END;
Tmid_bin:=Tot_temp/Sam;
{
*** DISPLAY THE AVERAGE VALUE ON THE HP9826 CRT
}
WRITELN('AVERAGE BINARY VALUE READ FOR ',Tmid:2:2,' deg C');
WRITELN('IS:  ',Tmid_bin:4:1);
{
*** INSTRUCT USER TO PLACE TC IN HIGH TEMPERATURE WATER BATH
}
WRITELN('PUT THERMOCOUPLE IN THE HIGH TEMP WATER BATH.');
{
*** GET ACTUAL TEMPERATURE OF HIGH TEMPERATURE WATER BATH
}
WRITELN('ENTER THE THE THERMOMETER READING (deg C).');
BEEP;
READLN(Thigh);
{
                Determine the true temperature.
                                                        }
Thigh := Thigh * Thermometer_cal_Slope + Thermometer_cal_Y_int;

{
*** GO TAKE 500 DATA POINTS ON THE TEMPERATURE CHANNEL
}
DATA_COLLECT(Sam,S,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE THE AVERAGE VALUE FOR THE 500 TEMPERATURE DATA POINTS
}
Tot_temp:=0;
FOR I:=1 TO Sam DO
```

```
    BEGIN
      Tot_temp:=Tot_temp+LINE4^[I];
    END;
Thigh_bin:=Tot_temp/Sam;
{
*** DISPLAY THE AVERAGE VALUE ON THE HP9826 CRT
}
WRITELN('AVERAGE BINARY VALUE READ FOR ',Thigh:2:2,' deg C');
WRITELN('IS:  ',Thigh_bin:4:1);
{
*** SET UP TEMPERATURE MATRIX FOR 2ND ORDER CURVE FIT DETERMINATION
}
Bin_temp[1,1]:=1;
Bin_temp[2,1]:=1;
Bin_temp[3,1]:=1;
Bin_temp[1,2]:=Tlow_bin;
Bin_temp[1,3]:=SQR(Tlow_bin);
Bin_temp[2,2]:=Tmid_bin;
Bin_temp[2,3]:=SQR(Tmid_bin);
Bin_temp[3,2]:=Thigh_bin;
Bin_temp[3,3]:=SQR(Thigh_bin);
{
*** CALCULATE THE DETERMINANT OF Bin_temp MATRIX
}
Norm_a:=Bin_temp[1,1]*(Bin_temp[2,2]*
        Bin_temp[3,3]-Bin_temp[2,3]*
        Bin_temp[3,2])-Bin_temp[1,2]*
        (Bin_temp[2,1]*Bin_temp[3,3]-
        Bin_temp[2,3]*Bin_temp[3,1])+
        Bin_temp[1,3]*(Bin_temp[2,1]*
        Bin_temp[3,2]-Bin_temp[2,2]*
        Bin_temp[3,1]);
{
*** DETERMINE THE INVERSE OF Bin_temp MATRIX
}
Bin_temp_inv[1,1]:=(Bin_temp[2,2]*
        Bin_temp[3,3]-Bin_temp[2,3]*
        Bin_temp[3,2])/Norm_a;
Bin_temp_inv[1,2]:=(Bin_temp[1,3]*
        Bin_temp[3,2]-Bin_temp[1,2]*
        Bin_temp[3,3])/Norm_a;
Bin_temp_inv[1,3]:=(Bin_temp[1,2]*
        Bin_temp[2,3]-Bin_temp[1,3]*
        Bin_temp[2,2])/Norm_a;
Bin_temp_inv[2,1]:=(Bin_temp[3,1]*
        Bin_temp[2,3]-Bin_temp[2,1]*
        Bin_temp[3,3])/Norm_a;
Bin_temp_inv[2,2]:=(Bin_temp[1,1]*
        Bin_temp[3,3]-Bin_temp[1,3]*
        Bin_temp[3,1])/Norm_a;
Bin_temp_inv[2,3]:=(Bin_temp[1,3]*
        Bin_temp[2,1]-Bin_temp[1,1]*
        Bin_temp[2,3])/Norm_a;
Bin_temp_inv[3,1]:=(Bin_temp[2,1]*
        Bin_temp[3,2]-Bin_temp[2,2]*
        Bin_temp[3,1])/Norm_a;
Bin_temp_inv[3,2]:=(Bin_temp[1,2]*
        Bin_temp[3,1]-Bin_temp[1,1]*
        Bin_temp[3,2])/Norm_a;
Bin_temp_inv[3,3]:=(Bin_temp[1,1]*
        Bin_temp[2,2]-Bin_temp[2,1]*
        Bin_temp[1,2])/Norm_a;
{
*** MULTIPLY INVERSE OF Bin_temp MATRIX BY TEMPERATURE TO YIELD 2ND
*** ORDER COEFFICIENTS
```

```
        }
        Tc:=Bin_temp_inv[1,1]*
                Tlow+Bin_temp_inv[1,2]*
                Tmid+Bin_temp_inv[1,3]*
                Thigh;
        Tb:=Bin_temp_inv[2,1]*
                Tlow+Bin_temp_inv[2,2]*
                Tmid+Bin_temp_inv[2,3]*
                Thigh;
        Ta:=Bin_temp_inv[3,1]*
                Tlow+Bin_temp_inv[3,2]*
                Tmid+Bin_temp_inv[3,3]*
                Thigh;
        {
        *** DISPLAY 2ND ORDER COEFFICIENTS ON HP9826 CRT
        }
        WRITELN('2nd ORDER POLYNOMIAL CALIBRATION COEFFICIENTS');
        WRITELN('SECOND ORDER COEFFICIENT --> ',Ta);
        WRITELN('FIRST ORDER COEFFICIENT --> ',Tb);
        WRITELN('ZERO ORDER COEFFICIENT --> ',Tc);
   END; {TEMPCAL}
{-------------------------------------------------------------------------------
                                                                              }
{


*******************************************************************************
                                                                              }
{
 BEGIN MAIN SYSTEM CALIBRATION PROGRAM (CAP2)
                                                                              }
BEGIN {CAP2 START}
   NEW(Line1);      {CREATE DYNAMIC VARIABLE Line1}
   NEW(Line2);      {CREATE DYNAMIC VARIABLE Line2}
   NEW(Line3);      {CREATE DYNAMIC VARIABLE Line3}
   NEW(Line4);      {CREATE DYNAMIC VARIABLE Line4}
   {
   *** GO SET DAM CLOCK AT DESIRED FREQUENCY
   }
   CLKSET(S);
   {
   *** SET NUMBER OF SAMPLES AT 500 POINTS PER CHANNEL
   }
   Sam:=500;
   {
   *** CALIBRATE FRACTIONAL CONCENTRATIONS SIGNALS IF DESIRED
   }
   PROMPT ('CALIBRATE FRACTIONAL CONCENTRATION SIGNAL? (Y/N) ');
   Ask_Y_or_N(Q);
   IF Q='Y' THEN
      BEGIN
        GASCAL(O1,Co2_cal,O2_cal,Co2_dc_offset,
               O2_dc_offset,Sam,S);
      END;
   {
   *** CALIBRATE THE TEMPERATURE SIGNAL IF DESIRED
   }
   WRITELN('Do you wish to calibrate the Temperature');
   PROMPT ('Transducer? (Y/N)  ');
   Ask_Y_or_N(Q);
   IF Q='Y' THEN


                {Keep recalibrating the temperature signal until
                the user no longer desires to do so.}
```

```
    BEGIN
    REPEAT
        TEMPCAL(Ta,Tb,Tc,Sam,S);
        WRITELN('Do you wish to recalibrate the Temperature');
        PROMPT ('Transducer? (Y/N)  ');
        Ask_Y_or_N(Q);
        IF Q='Y' THEN
            BEGIN
            WRITELN('You are going to recalibrate the Temperature');
            PROMPT ('Transducer.  Is this what you want to do? (Y/N)');
            Ask_Y_or_N(Q);
            END;

    UNTIL Q='N';
    END;


            {*** CALIBRATE THE FLOW SIGNAL IF DESIRED ***}

PROMPT ('CALIBRATE THE FLOW SIGNAL? (Y/N)  ');
Ask_Y_or_N(Q);
IF Q='Y' THEN
    BEGIN
    WRITELN('Do you wish to use gas temperature and gas');
    WRITELN('viscosity corrections while calibrating the flow');
    WRITELN('signal? (Y/N)');
    Ask_Y_or_N(Q);
    IF Q='Y' THEN
        BEGIN
        Corr_flag := TRUE;

        PROMPT ('Enter the percent relative humidty  ');
        BEEP;
        READLN(Rel_humid);
        Rel_humid := Rel_humid / 100;

        PROMPT ('Enter the room temperature. (deg F)  ');
        BEEP;
        READLN(Room_temp);
        Room_temp := (Room_temp - 32.0) * 5/9;

        PROMPT ('Enter the barometric pressure. (inches)  ');
        BEEP;
        READLN(Pb);       {read in inches}
        Pb := Pb * 25.4; {torr/inch}
        END
    ELSE
        BEGIN
        Corr_flag := FALSE;
        END;

    FLOWCAL(Non_C_I_flow_cal,Non_C_E_flow_cal,
            Corr_I_flow_cal,Corr_E_flow_cal,Bin_zero_flow,
            Sam,S,Corr_flag,Rel_humid,Pb,Room_temp,Ta,Tb,Tc);

    Line_feed(2);
    END;


     {*** STORE THE CALIBRATION DATA IN AN ASCII FILE IF DESIRED ***}

PROMPT ('STORE CALIBRATION FACTORS ON DISK? (Y/N)  ');
Ask_Y_or_N(Q);
IF Q='Y' THEN
```

```
BEGIN

        {*** GET CALIBRATION FILE NAME ***}
WRITELN('ENTER THE FILE NAME FOR CALIBRATION FACTORS.');
BEEP;
READLN(Fname);


{*** Determine where to store the 'MONSTER' ASCII files. ***}
Msi_devices[1] := '#12:';    {The hard disk, platter 2}
Msi_devices[2] := '#13:';    {The hard disk, platter 3}
Msi_devices[3] := '#7:';     {The 8 in. floppy disk.}
WRITELN('You may store the ASCII files in one of the');
WRITELN('locations below.');
WRITELN(' ');
WRITELN('      1.   The Hard disk, Platter 1');
WRITELN('      2.   The Hard disk, Platter 2');
WRITELN('      3.   The 8 in. floppy disk.');
WRITELN(' ');
REPEAT
    BEEP;
    PROMPT ('Which device do you want to use? ');
    READLN (Device_pointer);
    IF (Device_pointer < 1) OR (Device_pointer >3) THEN
        BEGIN
        WRITELN(' ');
        WRITELN ('You entered an invalid selection');
        END;

UNTIL (Device_pointer=1) OR (Device_pointer=2) OR (Device_pointer=3);

Fname := MSI_devices[Device_pointer] + Fname + '.ASC';


            {*** GET TODAYS DATE ***}
WRITELN('ENTER TODAYS DATE, FORMAT:  Month/Day/Year');
BEEP;
READLN(Date);

        {*** CREATE OR REWRITE ASCII FILE ***}
REWRITE(F,Fname);

        {*** WRITE ASCII Co2_dc_offset TO FILE ***}
STRWRITE(NSTRING,1,TEMP,Co2_dc_offset:4);
WRITELN(F,NSTRING);


        {*** WRITE ASCII O2_dc_offset TO FILE ***}
STRWRITE(NSTRING,1,TEMP,O2_dc_offset:4);
WRITELN(F,NSTRING);


            {*** WRITE ASCII Bin_zero_flow TO FILE ***}
STRWRITE(NSTRING,1,TEMP,Bin_zero_flow:4);
WRITELN(F,NSTRING);


            {*** WRITE ASCII Co2_cal TO FILE ***}
STRWRITE(RSTRING,1,TEMP,Co2_cal);
WRITELN(F,RSTRING);


            {*** WRITE O2_cal TO FILE ***}
STRWRITE(RSTRING,1,TEMP,O2_cal);
WRITELN(F,RSTRING);
```

```
            {*** WRITE Insp_flow_cal TO FILE ***}
 STRWRITE(RSTRING,1,TEMP,Non_C_I_flow_cal);
 WRITELN(F,RSTRING);


            {*** WRITE Expr_flow_cal TO FILE ***}
 STRWRITE(RSTRING,1,TEMP,Non_C_E_flow_cal);
 WRITELN(F,RSTRING);


              {*** WRITE Time_delay TO FILE ***}
 STRWRITE(RSTRING,1,TEMP,Time_delay);
 WRITELN(F,RSTRING);


          {*** WRITE SAMPLING FREQUENCY (S) TO FILE ***}
 STRWRITE(NSTRING,1,TEMP,S:4);
 WRITELN(F,NSTRING);


    {*** WRITE ACTUAL O2 CONCENTRATION FOR GAS MIXTURE TO FILE ***}
 STRWRITE(RSTRING,1,TEMP,O1);
 WRITELN(F,RSTRING);


 {*** WRITE 2ND ORDER TEMPERATURE COEFFICIENT (Ta) TO FILE ***}
 STRWRITE(RSTRING,1,TEMP,Ta);
 WRITELN(F,RSTRING);


  {*** WRITE 1ST ORDER TEMPERATURE COEFFICIENT (Tb) TO FILE ***}
 STRWRITE(RSTRING,1,TEMP,Tb);
 WRITELN(F,RSTRING);


     {*** WRITE CONSTANT TEMPERATURE COEFFICIENT (Tc) TO FILE ***}
 STRWRITE(RSTRING,1,TEMP,Tc);
 WRITELN(F,RSTRING);


              {*** WRITE TODAYS DATE TO FILE ***}
 WRITELN(F,Date);


 {This section writes theFlow calibrationn values which are a result of
  version 2.0 of the Calibration Aquisition Program (CAP2).  The vis-
  cosity corrected, STPD flow calibration values are written first
  followed by the barometric pressure and the relative humidity.}


  STRWRITE(RSTRING,1,TEMP,Corr_I_flow_cal);
  WRITELN (F,RSTRING);

  STRWRITE(RSTRING,1,TEMP,Corr_E_flow_cal);
  WRITELN (F,RSTRING);

  STRWRITE(RSTRING,1,TEMP,Pb);
  WRITELN (F,RSTRING);

  STRWRITE(RSTRING,1,TEMP,Rel_humid);
  WRITELN (F,RSTRING);

  STRWRITE(RSTRING,1,TEMP,Room_temp);
```

```
      WRITELN (F,RSTRING);

            {*** CLOSE AND COMPACT THE CALIBRATION FILE ***}
      CLOSE(F,'CRUNCH');

    END;
  WRITELN ('      CAP2 PROGRAM RUN COMPLETE');
  BEEP;
END. {CAP2 END}
```

A6.2 DAP2: a Pascal Routine for Collection of Respiratory Data

The Pascal program DAP2 (Data Acquisition Program, version 2) is used to collect respiratory data from an instrumented subject. A flowchart for the program is given in Chapter 6, Section 2. The method used by DAP2 to control the DAM is described by Riblett [13]. The contents of the data files has the same file structure specified by Riblett: four serial ASCII files. A description of each of the procedures used by DAP2 follows.

A6.2.1 BEEP Procedure

The procedure BEEP sounds an audible beep. This procedure is used to gain the operator's attention. This is the same procedure used by CAP2.

Variables passed to the procedure

None

Variables returned by the procedure

None

Variable definitions

No variables used.

A6.2.2 BIT_HI and BIT_TST Procedures

BIT_HI and BIT_TEST are external assembly language procedures. These procedures are used to control the DAM. BIT_HI and BIT_TEST procedures are documented by Riblett [13]. These procedures are also used by CAP2.

A6.2.3 CHECK_IF_DISK_FULL Procedure

The CHECK_IF_DISK_FULL procedure checks the storage device to determine if the disk contains uncompacted ASCII data. A status file, named 'DISK_STAT.ASC', created by the ASCI_CONF Pascal program is checked to determine if the data has been compacted.

Variables passed to the procedure

Msi_device - string, passed by reference.

Variables returned by the procedure

Disk_stat - string, passed by reference.

Variable definitions

Disk_stat - a string variable containing the status of the disk. Disk_stat is equal to 'EMPTY' if the disk does not contain ASCII data and 'FULL' if the disk does contain ASCII data.

Disk_stat_file - a text variable associated with the file specified by File_name.

File_name - a string containing the name of the file whose status is to be checked. File_name is set equal to the variable Msi_device concatenated with the string 'DISK_STAT.ASC'.

Msi_device - a string containing the mass storage device which is to be checked.

A6.2.4 CHECK_SATURATION Procedure

CHECK_SATURATION determines if a datum sample is saturated, equal to the maximum sample value of 4095 or equal to the minimum sample value of 0. If a saturated value is found, the appropriate saturated value counter is incremented. CHECK_SATURATION is nested within the MAXMIN procedure.

Variables passed to the procedure

Number_equal_4095 - integer, passed by reference.

Number_equal_0 - integer, passed by reference.

Sample - integer, passed by value.

Variables returned by the procedure

Number_equal_4095 - integer, passed by reference.

Number_equal_0 - integer, passed by reference.

Variable definitions

Number_equal_4095 - an integer value which is incremented if the sample is equal to 4095.

Number_equal_0 - an integer value which is incremented if the sample is equal to 0.

Sample - an integer which is equal to the sampled value to be checked for saturation.

A6.2.5 CLKSET Procedure

The procedure CLKSET sets the 8253 timer in the DAM with the desired sampling frequency. CLKSET is called by the main CAP2 procedure to set the DAM's sampling frequency. This procedure was designed by Riblett [13] and has no modifications for version 2 of the calibration program (CAP2). CLKSET is also used by CAP2.

Variables passed to the procedure

None

Variables returned by the procedure

S - integer, passed by reference.

Variable definitions

F1 - an integer used to set the least significant byte of the 8253 timer.

Fm - an integer used to set the most significant byte of the 8253 timer.

S - an integer equal to the user specified sampling frequency.

X - an integer variable used to set clock 1 of the 8253 timer.

A6.2.6 DAP2 Procedure

The DAP2 procedure provides the main control of the the data acquisition. DAP2 calls the DATA_COLLECTION procedure, the MAXMIN procedure, and the DATA_STORAGE procedure. DAP2 also calls the BEEP and HOLD_UP procedures.

Variables passed to the procedure

None

Variables returned by the procedure

None

Variable definitions

Co2max - an integer containing the maximum sampled datum point obtained from the CO_2 channel, Line1.

Co2min - an integer containing the minimum sampled datum point obtained from the CO_2 channel, Line1.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 1, the CO_2 channel, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 2, the O_2 channel, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 3, the flow channel, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 4, the temperature channel, of the DAM.

MAX - a constant value equal to 24,000, the maximum number of samples which may be collected.

O2max - an integer containing the maximum sampled datum point obtained from the O_2 channel, Line2.

O2min - an integer containing the minimum sampled datum point obtained from the O_2 channel, Line2.

S - an integer containing the sampling frequency.

Sam - an integer containing the number of samples to be collected.

Tmax - an integer containing the maximum sampled datum point obtained from the temperature channel, Line4.

Tmin - an integer containing the minimum sampled datum point obtained from the temperature channel, Line4.

Vmax - an integer containing the maximum sampled datum point obtained from the flow channel, Line3.

Vmin - an integer containing the minimum sampled datum point obtained from the flow channel, Line3.

A6.2.7 DATA_COLLECT Procedure

The procedure DATA_COLLECT provides the necessary control of the DAM to collect a set of data from four channels. DATA_COLLECT samples the analog signals. DATA_COLLECT calls the assembly language procedures BIT_HI and BIT_TST to test the status of the DAM. This procedure was designed by Riblett [13] and has only minor modifications for version 2 of the calibration program (CAP2). This is the same

procedure used by CAP2 to collect data.

Variables passed to the procedure

S - integer, passed by reference.

Sam - integer, passed by reference.

Variables returned by the procedure

Line1 - integer array, passed by reference.

Line2 - integer array, passed by reference.

Line3 - integer array, passed by reference.

Line4 - integer array, passed by reference.

Variable definitions

Chana - a constant value equal to 15359 which sets the DAM's analog multiplexer so that channel A is passed to the analog to digital converter.

Chanb - a constant value equal to 13311 which sets the DAM's analog multiplexer so that channel B is passed to the analog to digital converter.

Chanc - a constant value equal to 11263 which sets the DAM's analog multiplexer so that channel C is passed to the analog to digital converter.

Chand - a constant value equal to 9215 which sets the DAM's analog multiplexer so that channel D is passed to the analog to digital converter.

Del - an integer variable used in the delay loop that allows the sample and hold amplifiers to reach equilibrium once the hold command is given.

I - an integer variable used as a loop counter and a pointer into the four data arrays during collection of the specified number of datum points.

Line1 - an integer array containing the sampled data acquired from the CO_2 channel of the DAM.

Line2 - an integer array containing the sampled data acquired from the O_2 channel of the DAM.

Line3 - an integer array containing the sampled data acquired from the flow channel of the DAM.

Line4 - an integer array containing the sampled data acquired from the temperature channel of the DAM.

Mask - a constant value equal to 4095 which is used to mask of

the all but the 12 least significant data bits.

R4 - an integer variable which is loaded with the contents of register 4 of the GPIO.

R6 - an integer variable which is loaded with the contents of register 6 of the GPIO.

S - an integer equal to the user specified sampling frequency.

Sam - an integer variable equal to the number of samples to be collected form the DAM.

A6.2.8 DATA_STORAGE Procedure

The data storage procedure, DATA_STORAGE, stores the collected data and the maximum and minimum values from each channel. The file format is specified by Riblett [13]. The procedure CHECK_IF_DISK_FULL is used to determine if the disk contains data. The procedure SET_DISK_FULL_FLAG is used to set the disk status to full.

Variables passed to the procedure

Co2max - integer, passed by value.
Co2min - integer, passed by value.
Line1 - array, passed by reference.
Line2 - array, passed by reference.
Line3 - array, passed by reference.
Line4 - array, passed by reference.
O2max - integer, passed by value.
O2min - integer, passed by value.
Sam - integer, passed by value.
Tmax - integer, passed by value.
Tmin - integer, passed by value.
Vmax - integer, passed by value.
Vmin - integer, passed by value.

Variables returned by the procedure

None

Variable definitions

Co2max - an integer containing the maximum sampled datum point obtained from the CO_2 channel, Line1.

Co2min - an integer containing the minimum sampled datum point obtained from the CO_2 channel, Line1.

Device_pointer - an integer which is used to indicate which mass storage device is to contain the calibration data.

Disk_stat - a string variable containing the status of the disk. Disk_stat is equal to 'EMPTY' if the disk does not contain ASCII data and 'FULL' if the disk does contain ASCII data.

F - a text variable associated with the file specified by the string variable File_name.

File_name - a string containing the name of the file to be stored.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 1, the CO_2 channel.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 2, the O_2 channel.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 3, the flow channel.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 4, the temperature channel.

MSI_devices - a string array containing three elements. These elements correspond to the three mass storage devices which can be used for the storage of the calibration data. Array element 1 contains #12 corresponding to platter 1 of the hard disk, element 2 contains #13 corresponding to platter 2 of the hard disk, and element 3 contains #7 corresponding to the 8-inch disk drive.

NSTRING- a string variable used to represent the ASCII value of an integer number.

O2max - an integer containing the maximum sampled datum point obtained from the O_2 channel, Line2.

O2min - an integer containing the minimum sampled datum point obtained from the O_2 channel, Line2.

Sam - an integer containing the number of samples to be collected.

TEMP - an integer variable needed by the STRWRITE function to convert the calibration data to ASCII strings.

Tmax - an integer containing the maximum sampled datum point

obtained from the temperature channel, Line4.

Tmin - an integer containing the minimum sampled datum point obtained from the temperature channel, Line4.

Vmax - an integer containing the maximum sampled datum point obtained from the flow channel, Line3.

Vmin - an integer containing the minimum sampled datum point obtained from the flow channel, Line3.

A6.2.9 HOLD UP Procedure

The hold up procedure, HOLD_UP, pauses program execution until the operator is ready to proceed. To continue program execution the operator must press the 'ENTER' key. This procedure was designed by Riblett [13] and has no modifications for version 2 of the calibration program (CAP2). This procedure is also used by CAP2.

Variables passed to the procedure

None

Variables returned by the procedure

None

Variable definitions

No variables used.

A6.2.10 Line feed Procedure

The line feed procedure, Line_feed, sends a specified number of line feeds to the screen. This procedure is also used by CAP2.

Variables passed to the procedure

I - integer, passed by value.

Variables returned by the procedure

None

Variable definitions

I - an integer variable containing the number of line feeds to send to the screen.

J - an integer variable used as a loop counter. J is incremented for each line feed which is sent until it is equal to the number of line feeds to be sent, I.

A6.2.11 MAXMIN Procedure

MAXMIN computes the maximum sample value and minimum sample value obtained from each channel. The procedure calls CHECK_SATURATION to determine the number of samples that are saturated, equal to 4095 or 0, from each channel.

Variables passed to the procedure

Sam - integer, passed by reference.

Variables returned by the procedure

Co2max - integer, passed by value.
Co2min - integer, passed by value.
O2max - integer, passed by value.
O2min - integer, passed by value.
Tmax - integer, passed by value.
Tmin - integer, passed by value.
Vmax - integer, passed by value.
Vmin - integer, passed by value.

Variable definitions

Co2max - an integer containing the maximum sampled datum point obtained from the CO_2 channel, Line1.

Co2min - an integer containing the minimum sampled datum point obtained from the CO_2 channel, Line1.

I - an integer variable used as a pointer into the data arrays.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 1, the CO_2 channel.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 2, the O_2 channel.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 3, the flow channel.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from channel 4, the temperature channel.

Num_CO2_equal_4095 - an integer equal to the number of CO_2 samples which are saturated on the positive rail, 4095.

Num_CO2_equal_0 - an integer equal to the number of CO_2 samples

which are saturated on the negative rail, 0

Num_O2_equal_4095 - an integer equal to the number of O_2 samples which are saturated on the positive rail, 4095.

Num_O2_equal_0 - an integer equal to the number of O_2 samples which are saturated on the negative rail, 0

Num_T_equal_4095 - an integer equal to the number of temperature samples which are saturated on the positive rail, 4095.

Num_T_equal_0 - an integer equal to the number of temperature samples which are saturated on the negative rail, 0

Num_V_equal_4095 - an integer equal to the number of flow samples which are saturated on the positive rail, 4095.

Num_V_equal_0 - an integer equal to the number of flow samples which are saturated on the negative rail, 0

O2max - an integer containing the maximum sampled datum point obtained from the O_2 channel, Line2.

O2min - an integer containing the minimum sampled datum point obtained from the O_2 channel, Line2.

Sam - an integer containing the number of samples to be collected.

Tmax - an integer containing the maximum sampled datum point obtained from the temperature channel, Line4.

Tmin - an integer containing the minimum sampled datum point obtained from the temperature channel, Line4.

Vmax - an integer containing the maximum sampled datum point obtained from the flow channel, Line3.

Vmin - an integer containing the minimum sampled datum point obtained from the flow channel, Line3.

A6.2.12 SET_DISK_FULL_FLAG Procedure

The SET_DISK_FULL_FLAG procedure sets the status of the data storage disk to the full state indicating the disk contains uncompacted ASCII data. A status file, named 'DISK_STAT.ASC', created by the ASCI_CONF Pascal program is set to the 'FULL' state.

Variables passed to the procedure

Msi_device - string, passed by reference.

Variables returned by the procedure

None

Variable definitions

Disk_stat_file - a text variable associated with the file specified by the File_name.

File_name - a string containing the name of the file to be checked for the disk status. File_name is set equal to the variable Msi_device concatenated with the name 'DISK_STAT.ASC'.

Msi_device - a string containing the mass storage device which is to be checked.

A6.2.13 DAP2 Program Listing

```
$SYSPROG ON$
$TABLES$
$LINES 57$
PROGRAM DAP2(INPUT,OUTPUT);
{*****************************************************************************

        DATA ACQUISTION AND STORAGE ROUTINE

        PASCAL REV 2.1 SOURCE FILENAME:  DAP2.TEXT

        Department of Elecrtical and Computer Engineering
        Kansas State University

        REVISION        DATE                    PROGRAMMER
        --------        ----                    ----------
          1.0           JUNE 28, 1984           LOREN E. RIBLETT, JR.
          2.0           1-Feb-85                Michael Masters

*****************************************************************************

        PURPOSE

                THIS ROUTINE PERFORMS ALL THE NECESSARY ACTIONS TO ACQUIRE,
                CONVERT (TO ASCII), AND STORE FOUR CHANNELS (CO2, O2, FLOW,
                AND TEMPERATURE) OF 12-BIT BINARY DATA FROM THE DAM.

        ROUTINE(S) CALLED BY THIS ROUTINE

                BIT_TST - 68000 ASSEMBLY MODULE THAT WAITS UNTIL THE EOC BIT
                          ON THE DAM GOES HIGH, THEN LOW
                BIT_HI  - 68000 ASSEMBLY MODULE THAT WAITS UNTIL THE EOC BIT
                          ON THE DAM GOES LOW
                CLKSET  - INTERNAL PROCEDURE THAT SETS THE 8253 TIMER CHIP FOR
                          THE PROPER SAMPLING FREQUENCY
                MAXMIN  - INTERNAL PROCEDURE THAT DETERMINES THE MAXIMUM AND
                          MINIMUM VALUES FOR EACH OF THE FOUR DATA CHANNELS
                DATA_STORAGE - INTERNAL PROCEDURE THAT CONVERTS THE FOUR
                               CHANNELS OF 12-BIT VALUES TO ASCII AND STORES
                               THESE VALUES IN FOUR SEPERATE ASCII FILES
                HOLD_UP - INTERNAL PROCEDURE FOR TEMPORARY PAUSING OF PROGRAM
                          OPERATION
                DATA_COLLECT - INTERNAL PROCEDURE THAT CONTROLS THE DAM IN THE
                               PROPER FASHION TO COLLECT THE DESIRED NUMBER OF
                               DATA POINTS

                BEEP    - Sounds an audible beep to alert the operator.

                Line_feed - Sends a specified number of line feeds to the
                            terminal.

*****************************************************************************

        NOTE 1:  THE DAM SHOULD BE CONNECTED TO THE HP9826 COMPUTER VIA A
                 GPIO INTERFACE AT SELECT CODE #12.  THIS INSURES THE PROPER
                 DEVICE ADDRESS FOR SENDING AND RECEIVING INFORMATION BETWEEN
                 THE DAM AND THE HP9826.

        NOTE 2:  A MAXIMUM OF 24000 DATA POINTS PER CHANNEL IS ALLOWED WITH
                 THE CURRENT VERSION OF DAP2.  THIS VALUE MAY HAVE TO BE
                 REDUCED IF SUBSTANTIAL ADDITIONS TO DAP2'S PROGRAM LENGTH
                 IS REQUIRED.
```

NOTE 3: TWO EXTERNAL 68000 ASSEMBLY LANGUAGE ROUTINES ARE UTILIZED BY DAP2 TO HANDLE THE HIGH SPEED REQUIREMENTS NEEDED TO MONITOR THE End Of Conversion (EOC) SIGNAL FROM THE DAM.

NOTE 4: AT PRESENT, A MAXIMUM SAMPLING RATE OF 350 HZ IS RECOMMENDED. THIS VALUE MANY HAVE TO BE REDUCED IF SUBSTANTIAL ADDITIONS TO THE PROCEDURE DATA_COLLECT ARE MADE.

```
*****************************************************************************}
{
*** LOAD NECESSARY LIBRARY MODULES
}
IMPORT IODECLARATIONS,GENERAL_0,
       GENERAL_1,IOCOMASM;
{
*** DECLARE FOUR LARGE EXTERNAL DATA ARRAYS AND POINTERS
}
TYPE L1=ARRAY [1..24000] OF INTEGER;      {CO2 CHANNEL DATA ARRAY}
     PT1=^L1;      {POINTER TO ARRAY L1}
     L2=ARRAY [1..24000] OF INTEGER;      {O2 CHANNEL DATA ARRAY}
     PT2=^L2;      {POINTER TO ARRAY L2}
     L3=ARRAY [1..24000] OF INTEGER;      {FLOW CHANNEL DATA ARRAY}
     PT3=^L3;      {POINTER TO ARRAY L3}
     L4=ARRAY [1..24000] OF INTEGER;      {TEMPERATURE CHANNEL DATA ARRAY}
     PT4=^L4;      {POINTER TO ARRAY L4}
{
*** SET PROGRAM CONSTANTS
}
CONST MAX=24000;      {MAXIMUM NUMBER OF SAMPLES PER CHANNEL ALLOWED}
{
*** DECLARE PROGRAM VARIABLES
}
VAR S,Sam:INTEGER;
    Co2min,O2min,Vmin,Tmin,
    Co2max,O2max,Vmax,Tmax      :      INTEGER;
    Line1: PT1;
    Line2: PT2;
    Line3: PT3;
    Line4: PT4;


        {*** DECLARE EXTERNAL 68000 ASSEMBLY MODULES ***}
PROCEDURE BIT_TST;EXTERNAL;      {WAITS UNTIL EOC BIT GOES HIGH, THEN LOW}
PROCEDURE BIT_HI;EXTERNAL;      {WAITS UNTIL EOC BIT IS LOW}


PROCEDURE BEEP;
{*****************************************************************************
*****************************************************************************
*                                                                           *
*       Procedure Name:    BEEP                                             *
*                                                                           *
*       File Name:         DAP2                                             *
*                                                                           *
*       REVISION                DATE            PROGRAMMER                 *
*       --------                ---------       ----------                 *
*         1.0                   1-Feb-85        Michael Masters            *
*                                                                           *
*       Procedure discription:                                              *
*           This procedure sounds an audible beep to the operator.          *
*                                                                           *
*       Calling sequence:                                                   *
*            BEEP;                                                          *
*                                                                           *
```

```
*       Parameters supplied by the calling routine.                    *
*            None                                                      *
*                                                                      *
*       Parameters returned to the calling routine.                    *
*            None                                                      *
*                                                                      *
*======================================================================*
************************************************************************
                                                                       }
    BEGIN
    PROMPT (CHR(7));
    END;

{======================================================================}



PROCEDURE Line_feed(I    :       INTEGER);
{************************************************************************
************************************************************************
*                                                                      *
*       Procedure Name:   Line feed                                    *
*                                                                      *
*       File Name:        DAP2                                         *
*                                                                      *
*       REVISION              DATE            PROGRAMMER               *
*       --------              ---------       ----------               *
*         1.0                 1-Feb-85        Michael Masters          *
*                                                                      *
*       Procedure discription:                                         *
*           This procedure sends a specified number of line feeds to the *
*           terminal.                                                  *
*                                                                      *
*       Calling sequence:                                              *
*             Line_feed(1);                                            *
*                                                                      *
*       Parameters supplied by the calling routine.                    *
*                                                                      *
*               I - is an integer value parameter corresponding to the *
*                   number of line feeds to be sent to the terminal.   *
*                                                                      *
*       Parameters returned to the calling routine.                    *
*             None                                                     *
*                                                                      *
*======================================================================*
************************************************************************
                                                                       }
VAR     J       :       INTEGER;

    BEGIN
    FOR J := 1 TO I DO          {Send the specified number of line feeds}
        BEGIN
        PROMPT (CHR(10));   {Send the line feed}
        END;
    END;
{======================================================================}


            {*** DECLARE PROCEDURE FOR PAUSING PROGRAM OPERATION ***}
PROCEDURE HOLD_UP;
    BEGIN
    PROMPT ('PRESS ENTER TO CONTINUE.');      {DISPLAY PROMPT ON CRT}
```

```
    BEEP;
    READLN;      {WAIT UNTIL 'ENTER' IS PRESSED}
    END; {HOLD_UP END}



          {*** DECLARE PROCEDURE TO SET DAM CLOCK ****}
PROCEDURE CLKSET(VAR S:INTEGER);      {PASS SAMPLING FREQUENCY (S)}

VAR X,Fm,F1:INTEGER;
    BEGIN

              {*** HAVE USER ENTER THE SAMPLING FREQUENCY ***}
    PROMPT ('ENTER SAMPLING FREQUENCY:  ');
    BEEP;
    READLN(S);


      {*** DETERMINE 16-BIT COUNTER VALUE FOR CLK1 IN 8253 TIMER CHIP ***}
    X:=1000000 DIV 2 DIV S;
    IF X>=256 THEN
        BEGIN
        Fm:=X DIV 256;
        F1:=X-256*Fm;
        END
    ELSE
        BEGIN
        Fm:=0;
        F1:=X;
        END;

           {*** SET COUNTER 0 IN 8253 TIMER TO MODE 3 ***}
    IOCONTROL(12,3,15678); {11110100111110}
    IOCONTROL(12,3,15422); {11110000111110}
    IOCONTROL(12,3,15670); {11110100110110}


         {*** SET COUNTER 1 IN 8253 TIMER TO MODE 2 ***}
    IOCONTROL(12,3,15740); {11110101111100}
    IOCONTROL(12,3,15484); {11110001111100}
    IOCONTROL(12,3,15732); {11110101110100}


              {*** LOAD LSB OF COUNTER 0 ***}
    IOCONTROL(12,3,9474);  {10010100000010}
    IOCONTROL(12,3,9218);  {10010000000010}
    IOCONTROL(12,3,9474);  {10010100000010}


              {*** LOAD MSB OF COUNTER 0 ***}
    IOCONTROL(12,3,9472);  {10010100000000}
    IOCONTROL(12,3,9216);  {10010000000000}
    IOCONTROL(12,3,9472);  {10010100000000}


              {*** LOAD LSB OF COUNTER 1 ***}
    IOCONTROL(12,3,13568+F1); {11010100000000}
    IOCONTROL(12,3,13312+F1); {11010000000000}
    IOCONTROL(12,3,13568+F1); {11010100000000}

              {*** LOAD MSB OF COUNTER 1 ***}
    IOCONTROL(12,3,13568+Fm); {11010100000000}
    IOCONTROL(12,3,13312+Fm); {11010000000000}
```

```
    IOCONTROL(12,3,13568+Fm); {11010100000000}
    END; {CLKSET END}


PROCEDURE MAXMIN(VAR Sam,Co2max,Co2min,O2max,
                    O2min,Vmax,Vmin,Tmax,Tmin  :       INTEGER);
{*************************************************************************
**************************************************************************
*                                                                        *
*       Procedure Name:   MAXMIN                                         *
*                                                                        *
*       File Name:        DAP2                                           *
*                                                                        *
*       REVISION              DATE              PROGRAMMER               *
*       --------              ---------         ----------               *
*         1.0                 JUNE 28, 1984     LOREN E. RIBLETT, JR.    *
*         2.0                 1-Feb-85          Michael Masters          *
*                                                                        *
*       Procedure disoription:                                           *
*           This procedure calculates the maximum and minimum binary     *
*           values of CO2, O2, flow, and temperature.  The maximum and   *
*           mimimum values are displayed along with the number of sat-   *
*           uration points on each channel.                              *
*                                                                        *
*       Calling sequence:                                                *
*              MAXMIN (Sam,Co2max,Co2min,O2max,O2min,Vmax,Vmin,Tmax,Tmin, *
*                      Line1,Line2,Line3,Line4);                         *
*                                                                        *
*                                                                        *
*       Parameters supplied by the calling routine.                      *
*                    (all parameters are integers)                       *
*                                                                        *
*           Sam - A value parameter equal to the number of samples       *
*                 collected.                                             *
*                                                                        *
*           Line1  - A reference variable array containing the CO2 data. *
*                                                                        *
*           Line2  - A reference variable array containing the O2 data.  *
*                                                                        *
*           Line1  - A reference variable array containing the flow data.*
*                                                                        *
*           Line1  - A reference variable array containing the temp. data.*
*                                                                        *
*       Parameters returned to the calling routine.                      *
*                                                                        *
*           Co2max - A reference parameter equal to the maximum CO2 value.*
*                                                                        *
*           Co2min - A reference parameter equal to the minimum CO2 value.*
*                                                                        *
*           O2min  - A reference parameter equal to the minimum O2 value.*
*                                                                        *
*           Vmax   - A reference parameter equal to the maximum flow value.*
*                                                                        *
*           Vmin   - A reference parameter equal to the minimum flow value.*
*                                                                        *
*           Tmax   - A reference parameter equal to the maximum temp value*
*                                                                        *
*           Tmin   - A reference parameter equal to the minimum temp value*
*                                                                        *
*       Procedures called by this procedure.                             *
*            None                                                        *
*                                                                        *
*========================================================================*
```

```
*****************************************************************************
                                                                           }

VAR     I,Num_CO2_equal_0,Num_CO2_equal_4095,
        Num_O2_equal_0,Num_O2_equal_4095,
        Num_V_equal_0,Num_V_equal_4095,
        Num_T_equal_0,Num_T_equal_4095      :       INTEGER;

PROCEDURE CHECK_SATURATION(Sample : INTEGER;
                    VAR Number_equal_4095,Number_equal_0 : INTEGER);
    BEGIN

    IF Sample = 0 THEN Number_equal_0 := Number_equal_0 +1;
    IF Sample = 4095 THEN Number_equal_4095 := Number_equal_4095 +1;
    END;


    BEGIN {MAX/MIN ROUTINE}

                        {*** INITIALIZE THE DATA ***}
    Co2max := Line1^[1];
    Co2min := Line1^[1];
    O2max  := Line2^[1];
    O2min  := Line2^[1];
    Vmax   := Line3^[1];
    Vmin   := Line3^[1];
    Tmax   := Line4^[1];
    Tmin   := Line4^[1];

    Num_CO2_equal_4095 := 0;
    Num_CO2_equal_0    := 0;
    Num_O2_equal_4095  := 0;
    Num_O2_equal_0     := 0;
    Num_V_equal_4095   := 0;
    Num_V_equal_0      := 0;
    Num_T_equal_4095   := 0;
    Num_T_equal_0      := 0;

    {*** STEP THROUGH REMAINDER OF DATA TO FIND TRUE MAX/MIN VALUES ***}
    FOR I:=1 TO Sam DO
        BEGIN

                {*** CHECK FOR NEW MAX/MIN VALUES ***}
        IF Co2max<Line1^[I] THEN Co2max := Line1^[I];
        IF Co2min>Line1^[I] THEN Co2min := Line1^[I];
        IF O2max<Line2^[I]  THEN O2max  := Line2^[I];
        IF O2min>Line2^[I]  THEN O2min  := Line2^[I];
        IF Vmax<Line3^[I]   THEN Vmax   := Line3^[I];
        IF Vmin>Line3^[I]   THEN Vmin   := Line3^[I];
        IF Tmax<Line4^[I]   THEN Tmax   := Line4^[I];
        IF Tmin>Line4^[I]   THEN Tmin   := Line4^[I];


        CHECK_SATURATION(Line1^[I],Num_CO2_equal_4095,Num_CO2_equal_0);
        CHECK_SATURATION(Line2^[I],Num_O2_equal_4095,Num_O2_equal_0);
        CHECK_SATURATION(Line3^[I],Num_V_equal_4095,Num_V_equal_0);
        CHECK_SATURATION(Line4^[I],Num_T_equal_4095,Num_T_equal_0);

        END;  {Do Loop}


            {*** DISPLAY THESE MAX/MIN VALUES ON THE CRT ***}
    WRITELN ('Signal    Max   Min       #4095 s      #0 s');
```

```
    WRITELN (' CO2      ',Co2max:4,'  ',Co2min:4,
                        ',Num_CO2_equal_4095:5,'      ',Num_CO2_equal_0:5);
    WRITELN ('  O2      ',O2max:4,'  ',O2min:4,
                        ',Num_O2_equal_4095:5,'       ',Num_O2_equal_0:5);
    WRITELN (' FLOW     ',Vmax:4,'  ',Vmin:4,
                        ',Num_V_equal_4095:5,'        ',Num_V_equal_0:5);
    WRITELN (' Temp     ',Tmax:4,'  ',Tmin:4,
                        ',Num_T_equal_4095:5,'        ',Num_T_equal_0:5);
    END; {MAXMIN END}


PROCEDURE DATA_STORAGE(Sam,Co2max,Co2min,O2max,
                       O2min,Vmax,Vmin,Tmax,Tmin       :    INTEGER;
                    VAR Line1:PT1; VAR Line2:PT2; Line3:PT3; VAR Line4:PT4);
{*****************************************************************************
*****************************************************************************
*                                                                           *
*       Procedure Name:    BEEP                                           *
*                                                                           *
*       File Name:         DAP2                                           *
*                                                                           *
*       REVISION                DATE              PROGRAMMER              *
*       --------                ---------         ----------              *
*         1.0                   JUNE 28, 1984     LOREN E. RIBLETT, JR.   *
*         2.0                   1-Feb-85          Michael Masters         *
*                                                                           *
*                                                                           *
*       Procedure discription:                                              *
*           This procedure stores the respiratory data to a disk.           *
*                                                                           *
*       Calling sequence:                                                   *
*             DATA_STORAGE (Sam,Co2max,Co2min,O2max,O2min,Vmax,Vmin,        *
*                           Tmax,Tmin,Line1,Line2,Line3,Line4);             *
*                                                                           *
*       Parameters supplied by the calling routine.                         *
*                     (all parameters are integers)                         *
*                                                                           *
*           Sam - A value parameter equal to the number of samples          *
*                 collected.                                                *
*                                                                           *
*           Co2max - A value parameter equal to the maximum CO2 value.      *
*                                                                           *
*           Co2min - A value parameter equal to the minimum CO2 value.      *
*                                                                           *
*           O2min  - A value parameter equal to the minimum O2 value.       *
*                                                                           *
*           Vmax   - A value parameter equal to the maximum flow value.     *
*                                                                           *
*           Vmin   - A value parameter equal to the minimum flow value.     *
*                                                                           *
*           Tmax   - A value parameter equal to the maximum temp. value.    *
*                                                                           *
*           Tmin   - A value parameter equal to the minimum temp. value.    *
*                                                                           *
*           Line1  - A reference variable array containing the CO2 data.    *
*                                                                           *
*           Line2  - A reference variable array containing the O2 data.     *
*                                                                           *
*           Line1  - A reference variable array containing the flow data.   *
*                                                                           *
*           Line1  - A reference variable array containing the temp. data.  *
*                                                                           *
*                                                                           *
```

```
*       Parameters returned to the calling routine.                    *
*            None                                                      *
*                                                                      *
*       Procedures called by this procedure.                           *
*                                                                      *
*        CHECK_IF_DISK_FULL - Checks the selected mass storage medium to *
*                             see if it contins noncompacted ASCII data. *
*                                                                      *
*        SET_DISK_FULL_FLAG - Sets the ASCII disk to the full state.   *
*                                                                      *
*======================================================================*
************************************************************************
                                                                       }
VAR TEMP,I,Device_pointer   :       INTEGER;
    NSTRING                 :       STRING [4];
    Disk_stat               :       STRING[5];
    F                       :       TEXT;
    Msi_devices             :       ARRAY[1..3] OF STRING [4];
    File_name               :       STRING [16];

PROCEDURE  CHECK_IF_DISK_FULL( VAR Msi_device,Disk_stat :   STRING);
{************************************************************************
************************************************************************
*                                                                      *
*       Procedure Name:   Check if the disk is full                    *
*                                                                      *
*       File Name:        DAP2                                         *
*                                                                      *
*       REVISION              DATE            PROGRAMMER               *
*       --------              ---------       ----------               *
*         1.0                 1-Feb-85        Michael Masters          *
*                                                                      *
*                                                                      *
*      Procedure discription:                                          *
*          This procedure checks to see if the selected storage device *
*          contains ASCII data that has not been compacted.            *
*                                                                      *
*                                                                      *
*       Calling sequence:                                              *
*            CHECK_IF_DISK_FULL(Msi_device,Disk_stat);                 *
*                                                                      *
*       Parameters supplied by the calling routine.                    *
*                                                                      *
*            Msi_device - a string value parameter indicating which    *
*                 mass storage device is to be checked.                *
*                                                                      *
*       Parameters returned to the calling routine.                    *
*                                                                      *
*            Disk_stat - a string reference parameter that returns the *
*                 status of the disk.  Disk_stat is equal to 'FULL'    *
*                 if the disk contains un-compacted data and is equal  *
*                 'EMPTY' if the disk contains ascii data.             *
*                                                                      *
*======================================================================*
************************************************************************
                                                                       }
VAR  Disk_stat_file  :      TEXT;
     File_name       :      STRING[17];

     BEGIN
     File_name := Msi_device + 'DISK_STAT.ASC';
     RESET (Disk_stat_file,File_name);
     READLN (Disk_stat_file,Disk_stat);
     IF NOT (Disk_stat = 'FULL') THEN Disk_stat := 'EMPTY';
```

```
    END;
{=============================================================================}


PROCEDURE SET_DISK_FULL_FLAG( VAR Msi_device : STRING);
{*****************************************************************************
*****************************************************************************
*                                                                           *
*       Procedure Name:   Set the disk full flag                            *
*                                                                           *
*       File Name:        DAP2                                              *
*                                                                           *
*       REVISION              DATE            PROGRAMMER                    *
*       --------              ---------       ----------                    *
*         1.0                 1-Feb-85        Michael Masters               *
*                                                                           *
*       Procedure discription:                                              *
*           This procedure alters the disk status file and sets the flag    *
*           to FULL.                                                        *
*                                                                           *
*       Calling sequence:                                                   *
*            CHECK_IF_DISK_FULL(Msi_device,Disk_stat);                      *
*                                                                           *
*       Parameters supplied by the calling routine.                         *
*                                                                           *
*            Msi_device - a string value parameter indicating which         *
*                  mass storage device is to be set to the FULL state.      *
*                                                                           *
*       Parameters returned to the calling routine.                         *
*            None                                                           *
*                                                                           *
*===========================================================================*
*****************************************************************************}

VAR  Disk_stat_file  :       TEXT;
     File_name       :       STRING[17];

    BEGIN
    File_name := Msi_device + 'DISK_STAT.ASC';
    REWRITE (Disk_stat_file,File_name);
    WRITELN (Disk_stat_file,'FULL');
    CLOSE(Disk_stat_file,'CRUNCH');
    END;
{=============================================================================}


    BEGIN {DATA_STORAGE}

       {*** Determine where to store the 'MONSTER' ASCII files. ***}
    Msi_devices[1] := '#12:';    {The hard disk, platter 2}
    Msi_devices[2] := '#13:';    {The hard disk, platter 3}
    Msi_devices[3] := '#7:';     {The 8 in. floppy disk.}
    REPEAT
        Line_feed(3);
        WRITELN('You may store the ASCII files in one of the');
        WRITELN('locations below.');
        WRITELN(' ');
        WRITELN('      1.   The Hard disk, Platter 1.');
        WRITELN('      2.   The Hard disk, Platter 2.');
        WRITELN('      3.   The 8 in. floppy disk.');
```

```
    WRITELN(' ');

    REPEAT
        PROMPT ('Which device do you want to use? ');
        BEEP;
        READLN (Device_pointer);
        IF (Device_pointer < 1) OR (Device_pointer > 3)  THEN
            BEGIN
            Line_feed(1);
            WRITELN ('You entered an invalid device option.');
            END;
    UNTIL (Device_pointer>=1) AND (Device_pointer<=3);

    CHECK_IF_DISK_FULL(Msi_devices[Device_pointer],Disk_stat);
    IF Disk_stat='FULL' THEN
        BEGIN
        Line_feed(20);
        WRITELN ('The device you selected contains ASCII data.');
        WRITELN ('Either exchange the medium or select another');
        WRITELN ('device.');
        END;
UNTIL (Disk_stat='EMPTY');


      {*** BEGIN CREATING ASCII CO2 FILE, FILENAME = MONSTER1.ASC ***
            CREATE OR REWRITE FILE ON THE SPECIFIED UNIT}
WRITELN ('Storing the CO2 ASCII data.');
File_name := Msi_devices[Device_pointer] + 'MONSTER1.ASC';
REWRITE(F,File_name);
STRWRITE(NSTRING,1,TEMP,Co2max:4);      {CONVERT CO2 MAXIMUM TO ASCII}
WRITELN(F,NSTRING);                     {WRITE CO2 MAXIMUM TO FILE}
STRWRITE(NSTRING,1,TEMP,Co2min:4);      {CONVERT CO2 MINIMUM TO ASCII}
WRITELN(F,NSTRING);                     {WRITE CO2 MINIMUM TO FILE}


{*** MAIN LOOP FOR CONVERTING AND STORING CO2 DATA TO ASCII FILE ***}
FOR I:=1 TO Sam DO

    BEGIN
    STRWRITE(NSTRING,1,TEMP,Line1^[I]:4);     {CONVERT CO2 VALUE TO ASCII}
    WRITELN(F,NSTRING);       {WRITE CO2 VALUE TO FILE}
    END;

CLOSE(F,'CRUNCH');      {CLOSE AND COMPACT ASCII CO2 FILE}


    {*** BEGIN CREATING ASCII O2 FILE, FILENAME = MONSTER2.ASC ***}
WRITELN ('Storing the O2 ASCII data.');
File_name := Msi_devices[Device_pointer] + 'MONSTER2.ASC';
REWRITE(F,File_name);                   {CREATE OR REWRITE FILE }
STRWRITE(NSTRING,1,TEMP,O2max:4);       {CONVERT O2 MAXIMUM TO ASCII}
WRITELN(F,NSTRING);                     {WRITE O2 MAXIMUM TO FILE}
STRWRITE(NSTRING,1,TEMP,O2min:4);       {CONVERT O2 MINIMUM TO ASCII}
WRITELN(F,NSTRING);                     {WRITE O2 MINIMUM TO FILE}


{*** MAIN LOOP FOR CONVERTING AND STORING O2 DATA TO ASCII FILE ***}
FOR I:=1 TO Sam DO
    BEGIN
    STRWRITE(NSTRING,1,TEMP,Line2^[I]:4);     {CONVERT O2 VALUE TO ASCII}
    WRITELN(F,NSTRING);       {WRITE O2 VALUE TO FILE}
    END;

CLOSE(F,'CRUNCH');      {CLOSE AND COMPACT ASCII O2 FILE}
```

```
    {*** BEGIN CREATING ASCII FLOW FILE, FILENAME = MONSTER3.ASC ***}

    WRITELN ('Storing the Flow ASCII data.');
    File_name := Msi_devices[Device_pointer] + 'MONSTER3.ASC';
    REWRITE(F,File_name);                    {CREATE OR REWRITE FILE }
    STRWRITE(NSTRING,1,TEMP,Vmax:4);         {CONVERT FLOW MAXIMUM TO ASCII}
    WRITELN(F,NSTRING);                      {WRITE FLOW MAXIMUM TO FILE}
    STRWRITE(NSTRING,1,TEMP,Vmin:4);         {CONVERT FLOW MINIMUM TO ASCII}
    WRITELN(F,NSTRING);                      {WRITE FLOW MINIMUM TO FILE}


    {*** MAIN LOOP FOR CONVERTING AND STORING FLOW DATA TO ASCII FILE ***}
    FOR I:=1 TO Sam DO
        BEGIN
        STRWRITE(NSTRING,1,TEMP,Line3^[I]:4);    {CONVERT FLOW VALUE TO ASCII}
        WRITELN(F,NSTRING);                       {WRITE FLOW VALUE TO FILE}
      . END;

    CLOSE(F,'CRUNCH');      {CLOSE AND COMPACT ASCII FLOW FILE}


    {*** BEGIN CREATING ASCII TEMPERATURE FILE, FILENAME = MONSTER4.ASC ***}
    WRITELN('Storing the Temperature ASCII data');
    File_name := Msi_devices[Device_pointer] + 'MONSTER4.ASC';
    REWRITE(F,File_name);                    {CREATE OR REWRITE FILE }

    STRWRITE(NSTRING,1,TEMP,Tmax:4);         {CONVERT TEMPERATURE MAXIMUM TO ASCII}
    WRITELN(F,NSTRING);                      {WRITE TEMPERATURE MAXIMUM TO FILE}
    STRWRITE(NSTRING,1,TEMP,Tmin:4);         {CONVERT TEMPERATURE MINIMUM TO ASCII}
    WRITELN(F,NSTRING);                      {WRITE TEMPERATURE MINIMUM TO FILE}

    {*** MAIN LOOP FOR CONVERTING AND STORING FLOW DATA TO ASCII FILE ***}
    FOR I:=1 TO Sam DO
        BEGIN
        STRWRITE(NSTRING,1,TEMP,Line4^[I]:4); {CONVERT TEMP. VALUE TO ASCII}
        WRITELN(F,NSTRING);                   {WRITE TEMP. VALUE TO FILE}
        END;

  CLOSE(F,'CRUNCH');      {CLOSE AND COMPACT ASCII TEMP. FILE}

          {Set the disk is full flag.}
  SET_DISK_FULL_FLAG(Msi_devices[Device_pointer]);
  END; {DATA_STORAGE}
{=============================================================================}



{*** DECLARE PROCEDURE FOR COLLECTING FOUR CHANNELS OF DAM DATA ***}
PROCEDURE DATA_COLLECT(VAR Sam,S:INTEGER;
                       VAR LINE1:PT1;VAR LINE2:PT2;
                       VAR LINE3:PT3;VAR LINE4:PT4);
  VAR I,R6,De1,R4:INTEGER;
  CONST Chna=15359;Chnb=13311;      {BIT PATTERNS NECESSARY TO SET MEB CHANNEL
  CONST Chnc=11263;Chnd=9215;                                    MULTIPLIER
  CONST Mask=4095;      {MASKS OFF ALL BUT 12 DATA BITS IN STATUS WORD}
  {
  *** MAIN LOOP FOR FOUR CHANNEL DATA ACQUISITION
  }

  BEGIN {DATA_COLLECT}
      WRITELN('NOW COLLECTING DATA... please wait patiently.');
      WRITELN('Collecting ',Sam:5,' samples. This will take',Sam DIV S:4,' Sec');
```

```
FOR I:=1 TO Sam DO
   BEGIN
      {
      *** PUT S/H AMPS IN TRACKING MODE AND SELECT CHANNEL A
      }
      R6:=BINAND(15360,Chna);
      R6:=BINCMP(R6);
      IOCONTROL(12,3,R6);
      {
      *** GIVE S/H AMPS TIME TO TRACK INPUT SIGNALS
      }
      De1:=15;
      WHILE De1>0 DO
         BEGIN
            De1:=De1-1;
         END;
      {
      *** SELECT CHANNEL A ON MULTIPLEXER AND CONVERT SIGNAL HIGH
      }
      R6:=BINAND(7168,Chna);
      R6:=BINCMP(R6);
      IOCONTROL(12,3,R6);
      {
      *** SELECT CHANNEL A ON MULTIPLEXER AND CONVERT SIGNAL LOW
      }
      R6:=BINAND(7680,Chna);  {SEND CONVERT...}
      R6:=BINCMP(R6);         {...PULSE}
      IOCONTROL(12,3,R6);
      {
      *** SELECT CHANNEL A ON MULTIPLEXER AND CONVERT SIGNAL HIGH
      }
      R6:=BINAND(7168,Chna);  {RETURN TO...}
      R6:=BINCMP(R6);         {...NORMAL}
      IOCONTROL(12,3,R6);
      {
      *** WAIT FOR EOC LINE TO GO LOW
      }
      BIT_HI;
      {
      *** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
      }
      R4:=IOSTATUS(12,3);
      R4:=R4 DIV 4;
      R4:=BINAND(MASK,R4);
      Line1^[1]:=R4;
      {
      *** SELECT CHANNEL B ON MULTIPLEXER AND CONVERT SIGNAL HIGH
      }
      R6:=BINAND(7168,Chnb);
      R6:=BINCMP(R6);
      IOCONTROL(12,3,R6);
      {
      *** SELECT CHANNEL B ON MULTIPLEXER AND CONVERT SIGNAL LOW
      }
      R6:=BINAND(7680,Chnb);
      R6:=BINCMP(R6);
      IOCONTROL(12,3,R6);
      {
      *** SELECT CHANNEL B ON MULTIPLEXER AND CONVERT SIGNAL HIGH
      }
      R6:=BINAND(7168,Chnb);
      R6:=BINCMP(R6);
      IOCONTROL(12,3,R6);
      {
```

```
*** WAIT FOR EOC SIGNAL TO GO LOW
}
BIT_HI;
{
*** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
}
R4:=IOSTATUS(12,3);
R4:=R4 DIV 4;
R4:=BINAND(Mask,R4);
Line2^[I]:=R4;
{
*** SELECT CHANNEL C ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnc);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL C ON MULTIPLEXER AND CONVERT SIGNAL LOW
}
R6:=BINAND(7680,Chnc);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL C ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnc);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** WAIT FOR EOC SIGNAL TO GO LOW
}
BIT_HI;
{
*** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
}
R4:=IOSTATUS(12,3);
R4:=R4 DIV 4;
R4:=BINAND(Mask,R4);
Line3^[I]:=R4;
{
*** SELECT CHANNEL D ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnd);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL D ON MULTIPLEXER AND CONVERT SIGNAL LOW
}
R6:=BINAND(7680,Chnd);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL D ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chnd);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** WAIT FOR EOC SIGNAL TO GO LOW
}
BIT_HI;
{
*** READ GPIO STATUS REGISTER AND KEEP ONLY 12 BITS
}
R4:=IOSTATUS(12,3);
```

```
            R4:=R4 DIV 4;
            R4:=BINAND(Mask,R4);
            Line4^[I]:=R4;
            {
            *** WAIT FOR EOC SIGNAL TO GO HIGH, THEN LOW
            }
            BIT_TST;
          {
          *** LOOP BACK UNTIL ALL POINTS ARE COLLECTED
          }
          END;
  END; {DATA_COLLECT}


{*** BEGIN MAIN DATA ACQUISITION PROGRAM (DAP2). ***}
BEGIN {DAP2 START}
    NEW(Line1);       {CREATE DYNAMIC VARIABLE Line1}
    NEW(Line2);       {CREATE DYNAMIC VARIABLE Line2}
    NEW(Line3);       {CREATE DYNAMIC VARIABLE Line3}
    NEW(Line4);       {CREATE DYNAMIC VARIABLE Line4}


        {*** GO SET DAM CLOCK AT DESIRED FREQUENCY ***}
    CLKSET(S);


        {*** DETERMINE NUMBER OF DATA POINTS TO ACQUIRE ***}
    REPEAT
        PROMPT ('ENTER NUMBER OF SAMPLES [24000 MAX]:  ');
        BEEP;
        READLN(Sam);
    UNTIL (Sam<24001) AND (Sam>0);


       {*** GO COLLECT THE FOUR CHANNELS WORTH OF DATA ***}
    HOLD_UP;
    WRITELN ('Now collecting the data');
    DATA_COLLECT (Sam,S,LINE1,LINE2,LINE3,LINE4);
    BEEP;
    WRITELN ('DATA COLLECTION COMPLETE');

        {*** DETERMINE THE MAX/MIN VALUES FOR EACH CHANNEL ***}
    WRITELN ('CALCULATING THE MAXIMUM AND MINIMUM VALUES');
    MAXMIN(Sam,Co2max,Co2min,O2max,O2min,Vmax,Vmin,Tmax,Tmin);


     {*** GO STORE THE FOUR CHANNELS OF DATA AS ASCII FILES ***}
    HOLD_UP;
    DATA_STORAGE(Sam,Co2max,Co2min,O2max,O2min,
                 Vmax,Vmin,Tmax,Tmin,Line1,Line2,
                 Line3,Line4);
    WRITELN (' ');
    WRITELN ('***** PROGRAM RUN COMPLETE *****');
    BEEP;
END. {DAP2 END}
```

A6.3 ASCI_CONF: a Pascal Routine to Configure the ASCII Data Disks to Accept Data

The Pascal program ASCI_CONF (ASCII CONFiguration) is used to configure the respiratory data disk. DAP2 checks the status file, 'DISK_STAT.ASC', to determine if the data disk contains uncompacted ASCII data. This status file prevents DAP2 from overwriting previously collected data. The configuration program, ASCI_CONF, must be executed to create the status file prior to using a disk to store ASCII data. If the file is not present the data acquisition program, DAP2, will 'crash'.

The ASCI_CONF program can also be used to set a disk to the empty status so that DAP2 can store data on it. This is helpful in the event data is collected and stored but is not desired to be retained.

A6.3.1 ASCI_CONF Procedures

The ASCI_CONF procedure provides the main control for the configuration program. ASCI_CONF creates the status file and sets the status to the 'EMPTY' state. ASCI_CONF calls the BEEP and Line_feed procedures.

Variables passed to the procedures

None

Variables returned by the procedures

None

Variable definitions

Disk_stat_file - a text variable associated with the file specified by File_name.

Device_pointer - an integer which is used to indicate which mass storage device is to be configured.

File_name - a string containing the name of the file to be checked for the disk status. File_name is set equal to the variable Msi_device concatenated with the name 'DISK_STAT.ASC'.

MSI_devices - a string array containing three elements. These elements correspond to the three mass storage devices which can be used for the storage of the calibration data. Array element 1 contains #12 corresponding to platter 1 of

the hard disk, element 2 contains #13 corresponding to platter 2 of the hard disk, and element 3 contains #7 corresponding to the 8-inch disk drive.

A6.3.2 BEEP Procedures

The procedure BEEP sounds an audible beep. This procedure is used to gain the operator's attention. This is the same procedure used by CAP2 and DAP2.

Variables passed to the procedures

None

Variables returned by the procedures

None

Variable definitions

No variables used.

A6.3.3 Line feed Procedures

The line feed procedure, Line_feed, sends a specified number of line feeds to the screen. This is the same Line_feed procedure used by CAP2 and DAP2.

Variables passed to the procedures

I - integer, passed by value.

Variables returned by the procedures

None

Variable definitions

I - an integer variable containing the number of line feeds to send to the screen.

J - an integer variable used as a loop counter. J is incremented for each line feed which is sent until it is equal to the number of line feeds to be sent, I.

A6.3.4 ASCII_CONF Program Listing

```
$SYSPROG ON$
$LINES 57$
$TABLES$
$REF 25$
PROGRAM ASCII_CONF(INPUT,OUTPUT);
{*****************************************************************************
*****************************************************************************
*                                                                           *
*       Program Name:           ASCI_CONF                                   *
*                                                                           *
*       Department of Electrical and Computer Engineering                   *
*       Kansas State University                                             *
*                                                                           *
*       Source code location:   ASCI_CONF                                   *
*                                                                           *
*       REVISION                DATE            PROGRAMMER                  *
*       --------                ---------       ----------                  *
*         1.0                   20-Feb-85       Michael Masters.            *
*                                                                           *
*                                                                           *
*       Procedure discription:                                              *
*           This program sets up disks so that the respiratory data         *
*           collection program can store ASCII data on them.  The program   *
*           alters, or creates in the case of new disks, the disk status    *
*           file so that the disk can be written to by the data collection  *
*           program.                                                        *
*                                                                           *
*       Calling sequence:                                                   *
*           From the Main Command Level of the Pascal operating system      *
*           press the R key (R for RUN) and type ASCI_CONF in response      *
*           to the system prompt followed by the execute key.               *
*                                                                           *
*       Procedures called                                                   *
*           BEEP - sounds an audible beep to gain the attention of the      *
*                  operator.                                                *
*                                                                           *
*           Line_feed - sends a specified number of line feeds to the       *
*                  terminal                                                 *
*                                                                           *
*===========================================================================*
*****************************************************************************
                                                                          }
VAR    Device_pointer          :       INTEGER;
       Disk_stat_file          :       TEXT;
       Msi_devices             :       ARRAY[1..3] OF STRING [4];
       File_name               :       STRING [17];


PROCEDURE BEEP;
{*****************************************************************************
*****************************************************************************
*                                                                           *
*       Procedure Name:   BEEP                                              *
*                                                                           *
*       File Name:        ASCI_CONF                                         *
*                                                                           *
*       REVISION                DATE            PROGRAMMER                  *
*       --------                ---------       ----------                  *
*         1.0                   22-Sept-84      Michael Masters.            *
*                                                                           *
*                                                                           *
```

```
*       Procedure discription:                                          *
*           This procedure sounds an audible beep to the operator.      *
*                                                                       *
*                                                                       *
*                                                                       *
*       Calling sequence:                                               *
*             BEEP;                                                     *
*                                                                       *
*                                                                       *
*       Parameters supplied by the calling routine.                     *
*             None                                                      *
*                                                                       *
*       Parameters returned to the calling routine.                     *
*             None                                                      *
*                                                                       *
*=======================================================================*
*************************************************************************
                                                                       }
    BEGIN
    PROMPT (CHR(7));
    END;

{=============================================================================}


PROCEDURE Line_feed(I    :       INTEGER);
{*************************************************************************
*************************************************************************
*                                                                       *
*       Procedure Name:   Line feed                                     *
*                                                                       *
*       File Name:        ASCI_CONF                                     *
*                                                                       *
*       REVISION          DATE            PROGRAMMER                    *
*       --------          ---------       ----------                    *
*         1.0             22-Sept-84      Michael Masters.              *
*                                                                       *
*                                                                       *
*       Procedure discription:                                          *
*           This procedure sends a specified number of line feeds to the*
*           terminal.                                                   *
*                                                                       *
*                                                                       *
*       Calling sequence:                                               *
*             Line_feed(I);                                             *
*                                                                       *
*       Parameters supplied by the calling routine.                     *
*                                                                       *
*                I - is an integer value parameter corresponding to the *
*                    number of line feeds to be sent to the terminal.   *
*                                                                       *
*       Parameters returned to the calling routine.                     *
*             None                                                      *
*                                                                       *
*=======================================================================*
*************************************************************************
                                                                       }
    VAR     J       :       INTEGER;
    BEGIN
    FOR J := 1 TO I DO          {Send the specified number of line feeds}
        BEGIN
        PROMPT (CHR(10));   {Send the line feed}
        END;
```

```
    END;
{=============================================================================}



BEGIN {ASCI_CONF}
    Msi_devices[1] := '#12:';    {The hard disk, platter 2}
    Msi_devices[2] := '#13:';    {The hard disk, platter 3}
    Msi_devices[3] := '#7:';     {The 8 in. floppy disk.}
    Line_feed(3);

    WRITELN('ASCII_CONF will configure a disk so that it');
    WRITELN('will except the ASCII data created by the resp-');
    WRITELN('iratory data collection program DAP2.  The mass');
    WRITELN('storage devices listed below are valid locations');
    WRITELN('for storage of the ASCII respiratory data.');

    Line_feed(1);
    WRITELN('     1.   The Hard disk, Platter 1.');
    WRITELN('     2.   The Hard disk, Platter 2.');
    WRITELN('     3.   The 8 in. floppy disk.');
    WRITELN(' ');

    REPEAT


        PROMPT ('Which device do you want to configure? ');
        BEEP;
        READLN (Device_pointer);
        IF (Device_pointer < 1) OR (Device_pointer > 3)  THEN
            BEGIN
            WRITELN (' ');
            WRITELN ('You entered an invalid device option.');
            END;
    UNTIL (Device_pointer>=1) AND (Device_pointer<=3);

       {Set up the file name.}
    File_name := Msi_devices[Device_pointer];
    STRAPPEND (File_name,'DISK_STAT.ASC');

       {open the status file.}
    REWRITE (Disk_stat_file,File_name);

       {write an epmty status and close the file.}
    WRITELN (Disk_stat_file,'EMPTY');
    CLOSE(Disk_stat_file,'CRUNCH');

    BEEP;
    Line_feed(3);
    WRITELN ('      **** Configuration complete ****');

END.
```

APPENDIX VII. BASIC PROGRAM DOCUMENTATION AND LISTING

This appendix gives the details of the Basic programs used by the CBRMS. The documentation is similar to that of the Pascal programs. Each program is broken down into descriptions of the individual Basic subroutines which make up the program. The variables for a particular subroutine will be defined in that subroutine's section. The subroutines within each program are documented in alphabetical order. The variables within each subroutine are also listed in alphabetical order.

It should be noted that subroutine parameters can not be specified as value parameter or as a reference parameter, as is the case with Pascal procedures. Therefore, the parameter specifications will not state whether a parameter is passed by value or by reference. The manner in which a subroutine is called determines if a value is passed by reference or value. The following example points out how values are passed to a subroutine by reference or by value.

```
CALL Calc_sig_val((A),(Z),("INSP"),Incr_vol,Fo2,Fco2)
```

The first three parameters are passed by value and the last three are passed by reference.

Two additional sections have been added to the documentation of the Basic programs, definition of the COMmon blocks (A7.1) and a discussion of the respiratory variable array (Resp_var). The COMmon blocks serve as a link between the AUTOSTART, CAPCRUNCH, DAPCRUNCH, and ANALYSIS programs. This link allows variables to be passed between the programs. Also, several COMmon blocks are used to pass variables from a calling routine to a subroutine. The COMmon blocks and the contained variables are defined in Section A7.1. The documentation for the individual subroutines will include a listing of the COMmon blocks used in that subroutine. The respiratory variable array, Resp_var, is a special array containing the calculated breath-by-breath data.

A7.1 Definition of the COMmon Blocks

The COMmon blocks used by AUTOSTART, CAPCRUNCH, DAPCRUNCH, and ANALYSIS are defined here. The variables within each COMmon block are also defined.

The Anal_data COMmon blocks contain several variables which are used by the subroutines within the ANALYSIS program.

Anal_data1

Data_available - an integer variable used as a Boolean value which indicates if respiratory data is present in memory.

Anal_data2

False - an integer used as the constant Boolean value of false.

True - an integer used as the constant Boolean value of true.

Anal_data3

Co2_prod - an integer used as a constant value which points to the computed rate of CO_2 production data in the respiratory variable array.

Insp_min_vent - an integer used as a constant value which points to the computed inspiratory minute ventilation data in the respiratory variable array.

O2_cons - an integer used as a constant value which points to the computed rate of O_2 consumption data in the respiratory variable array.

Time_of_brth - an integer used as a constant value which points to the beginning time of each breath in the respiratory variable array.

Anal_data4

Expr_min_vent - an integer used as a constant value which points to the computed expiratory minute ventilation data in the respiratory variable array.

Resp_freq - an integer used as a constant value which points to the computed respiratory frequency data in the respiratory variable array.

V_tid_expr - an integer used as a constant value which points to the computed expiratory tidal volume data in the respiratory variable array.

V_tid_insp - an integer used as a constant value which points to the computed inspiratory tidal volume data in the respiratory variable array.

Anal_data5

Breath_good - an integer used as a constant value which points to the good or bad breath data in the respiratory variable array.

Formfd$ - a character used as the constant value of the ASCII form feed character (12 decimal, 0C hex).

Resp_quotient - an integer used as a constant value which points to the computed respiratory quotient data in the respiratory variable array.

Anal_data5a

Frc_co2_prod - an integer used as a constant value which points to the computed alveolar CO_2 production data in the respiratory variable array.

Frc_o2_cons - an integer used as a constant value which points to the computed alveolar O_2 consumption data in the respiratory variable array.

Volume_of_lung - an integer used as a constant value which points to the computed lung volume data in the respiratory variable array.

Anal_data6

Corr_flag - an integer variable used as a Boolean value which indicates if the temperature corrections are used in computing the respiratory flow.

Room_fh2o - a real variable equal to the relative concentration of water vapor in room air.

T - a real variable equal to the sampling frequency used to collect the respiratory data.

Anal_data7

Fh2o_sat - a real valued, 250 element array used to hold the saturated relative concentrations of water vapor in room air between 20°C and 45°C.

Vap - a real valued, 250 element array used to hold the saturated water vapor pressure between 20°C and 45°C.

Anal_data8

Resp_var - a real valued array containing the calculated breath-by-breath respiratory variables. Resp_var is a two dimensional array, the first dimension is the respiratory variable and the second dimension is the breath number. A more detailed description of the respiratory variable array is given in Section A7.2 of this appendix.

Anal_data9

Comment$ - a string variable containing a user specified comment.

Name$ - a string variable containing a user specified subject name.

The Cal_data COMmon blocks contain the calibration data generated by CAP2 and converted to numeric format by CAPCRUNCH.

Cal_data1

Cal$ - a string variable containing the name of the calibration file.

Date$ - a string variable containing the date data were collected.

Cal_data2

Bin_zero_flow - an integer variable containing the sampled value equal to zero flow through the PTM.

Co2_dc_offset - an integer variable containing the sampled CO_2 value corresponding to room air.

O2_dc_offset - an integer variable containing the sampled O_2 value corresponding to the calibration gas.

Cal_data3

Co2_cal - a real variable equal to the CO_2 calibration factor.

O2_cal - a real variable equal to the O_2 calibration factor.

Cal_data4

Non_c_e_flowcal - a real variable containing the PTG calibration factor for expiratory flows determined without the use of temperature corrections.

Non_c_i_flowcal - a real variable containing the PTG calibration factor for inspiratory flows determined without the use of temperature corrections.

Cal_data5

Corr_e_flowcal - a real variable containing the PTG calibration factor for expiratory flows determined using temperature corrections.

Corr_i_flowcal - a real variable containing the PTG calibration factor for inspiratory flows determined using

temperature corrections.

Cal_data6

Ta - a real variable equal to the second order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tb - a real variable equal to the first order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tc - a real variable equal to the zero order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Cal_data7

O1 - a real variable equal to the fractional concentration of O_2 in the calibration gas.

Pb - a real variable equal to the barometric pressure at the time the system was calibrated.

Rel_humid - a real variable equal to the relative humidity at the time the system was calibrated.

Room_temp - a real variable equal to the room temperature at the time the system was calibrated.

Time_delay - an integer variable equal to the GMS time delay in milli-seconds.

The Plot_resp_data COMmon blocks contain information used by the respiratory parameter display subroutines.

Plot_resp_data1

No_b_avg - an integer variable equal to the number of breath-by-breath values to be averaged.

Num_breaths - a real variable equal to the number of breaths that were integrated.

Omit_bad - an integer variable used as a Boolean flag which indicates if the 'bad' breaths are not to be included in the average.

Plot_resp_data2

End_time - a real variable equal to the ending time of the respiratory results display.

Start_time - a real variable equal to the starting time of the

respiratory results display.

W_period - a real variable equal time of the center of the window.

W_wdth - a real variable equal to the time width of the window.

The Resp_data COMmon blocks contain the respiratory data collected by DAP2 and converted to numeric format by DAPCRUNCH.

Resp_data1

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from the CO_2 channel, channel 1 of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from the O_2 channel, channel 2 of the DAM.

Resp_data2

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from the flow channel, channel 2 of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from the temperature channel, channel 4 of the DAM.

Resp_data3

Cmax - an integer variable containing the maximum sampled datum point obtained from the CO_2 channel, Line1.

Cmin - an integer variable containing the minimum sampled datum point obtained from the CO_2 channel, Line1.

Fmax - an integer variable containing the maximum sampled datum point obtained from the flow channel, Line3.

Fmin - an integer variable containing the minimum sampled datum point obtained from the flow channel, Line3.

Omax - an integer variable containing the maximum sampled datum point obtained from the O_2 channel, Line2.

Omin - an integer variable containing the minimum sampled datum point obtained from the O_2 channel, Line2.

Tmax - an integer variable containing the maximum sampled datum point obtained from the temperature channel, Line4.

Tmin - an integer variable containing the minimum sampled datum point obtained from the temperature channel, Line4.

Resp_data4

No_points - an integer variablecontaining the number of samples collected.

Resp_data5

C1$ - a string variable containing the name of the CO_2 data file.

O$ - a string variable containing the name of the O_2 data file.

T$ - a string variable containing the name of the temperature data file.

V$ - a string variable containing the name of the flow data file.

A7.2 Description of the Respiratory Variable Array

The respiratory variable array, Resp_var, contains the calculated breath-by-breath respiratory parameters. The respiratory variable array is a two dimensional array. The first dimension of the respiratory variable array is the respiratory parameter and the second dimension is the breath number. The array used in this work contains 13 respiratory parameters and can hold up to 500 breaths. The construction of the respiratory variable array is illustrated in Figure A7.2.1. The pointers to the array are shown on the left of the figure along with the values currently assigned to them. Access to a certain parameter for a particular breath is accomplished by placing the parameter of interest in the variable dimension and the breath number in the breath dimension. For example, X can be equated to the O_2 consumed for breath 5 by the statement.

```
X = Resp_var(O2_cons,5)
```

Use of this type of array to contain the calculated results allows for easy addition of other respiratory parameters. Additional variables can be stored in the array by increasing the respiratory variable dimension by one and defining a pointer to that variable. Also, it is an easy matter to add another dimension to the array.

		Breath Number 1	2	3	4		498	499	500
Time_of_brth	(0)								
O2_cons	(1)					.....			
Co2_prod	(2)					.....			
Frc_o2_cons	(3)					.....			
Frc_co2_prod	(4)					.....			
Insp_min_vent	(5)					.....			
Expr_min_vent	(6)					.....			
V_tid_insp	(7)					.....			
V_tid_expr	(8)					.....			
Volume_of_lung	(9)					.....			
Resp_freq	(10)					.....			
Resp_quotient	(11)					.....			
Breath_good	(12)					.....			

Figure A7.2.1 Construction of the respiratory variable array.

A7.3 AUTOST: a Basic Program Linking all Basic Programs

The auto start program, AUTOST, links the other Basic programs of the CBRMS. The operator can load any other program by pressing a function key while AUTOST is executing.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data1	Data_available
Cal_data1	Cal$, Date$
Cal_data2	Bin_zero_flow, Co2_dc_offset, O2_dc_offset
Cal_data3	Co2_cal, O2_cal
Cal_data4	Non_c_e_flowcal, Non_c_i_flowcal
Cal_data5	Corr_e_flowcal, Corr_i_flowcal
Cal_data6	Ta, Tb, Tc
Cal_data7	O1, Pb, Rel_humid, Room_temp, Time_delay
Resp_data1	Line1, Line2
Resp_data2	Line3, Line4
Resp_data3	Cmax, Cmin, Fmax, Fmin, Omax, Omin, Tmax,
Resp_data4	No_points

Resp_data5 - C1$, O$, T$, V$

Variable definitions

A - a real variable used as a pointer into the mass storage device array, Dev$.

Dev$ - a string array of five elements containing the five mass storage devices. This array is used to allow the operator to easily change the default mass storage device.

File_to_load$ - a string variable which contains an operator specified program file name. This program is loaded and begins execution.

I - a real variable used as a pointer into the Dev$ array.

A7.3.1 AUTOST Program Listing

```
10    !************************************************************************
20    !
30    !       AUTO-START ROUTINE
40    !
50    !       HP BASIC FILENAME:  AUTOST
60    !
70    !       Department of Electrical and Computer Engineering
80    !       Kansas State University
90    !
100   !       REVISION        DATE                    PROGRAMMER
110   !       --------        ----                    ----------
120   !         1.0           JUNE 1, 1984            LOREN E. RIBLETT
130   !
140   !************************************************************************
150   !
160   !       PURPOSE
170   !
180   !               THIS ROUTINE ALLOWS THE USER TO ACCESS THE VARIOUS HP
190   !               BASIC ROUTINES THAT CURRENTLY EXIST FOR THE HUMAN
200   !               RESPIRATORY RESEARCH STUDIES.
210   !
220   !       ROUTINE(S) CALLED BY THIS ROUTINE
230   !
240   !               CAPCRUNCH - CALIBRATION ASCII TO BINARY FILE CONVERSION
250   !               DAPCRUNCH - DAM DATA ASCII TO BINARY FILE CONVERSION
260   !               ANALYSIS - BREATH-BY-BREATH ANALYSIS ROUTINE
270   !               STOREHR  - Stores the heart rate values in a format
280   !                          compatible with Sprick's PLOT program.
290   !
300   !************************************************************************
310   !
320   !       NOTE 1:  Because this is an AUTOST routine, it should appear
330   !                only on the 5.25" floppy used in the mass storage unit
340   !                ":INTERNAL".
350   !
360   !       NOTE 2:  The program is initiated by placing the floppy in the
370   !                mentioned drive and applying power to the HP9826.
380   !
390   !       NOTE 3:  This program is recalled upon completion of the called
400   !                routines.
410   !
420   !************************************************************************
430   COM /Resp_data1/ INTEGER Line1(1:24000),Line2(1:24000)
440   COM /Resp_data2/ INTEGER Line3(1:24000),Line4(1:24000)
450   COM /Resp_data3/ INTEGER Cmin,Cmax,Omin,Omax,Fmin,Fmax,Tmin,Tmax
460   COM /Resp_data4/ INTEGER No_points
470   COM /Resp_data5/ C1$[10],O$[10],V$[10],T$[10]
480   COM /Cal_data1/ Cal$[10],Date$[25]
490   COM /Cal_data2/ INTEGER Co2_dc_offset,O2_dc_offset,Bin_zero_flow,S
500   COM /Cal_data3/ REAL Co2_cal,O2_cal
510   COM /Cal_data4/ REAL Non_c_i_flowcal,Non_c_e_flowcal
520   COM /Cal_data5/ REAL Corr_i_flowcal,Corr_e_flowcal
530   COM /Cal_data6/ REAL Ta,Tb,Tc
540   COM /Cal_data7/ REAL Time_delay,O1,Pb,Rel_humid,Room_temp
550   COM /Anal_data1/ INTEGER Data_available
560   !
570   !*** DECLARATIONS
580   !
590   DIM Dev$(5)[15],File_to_load$[25]
600   !
610   !*** ASSIGN SPECIAL FUNCTION KEYS
620   !
```

```
630   ON KEY 0 LABEL "CCRUNCH" GOTO Ccrunch
640   ON KEY 1 LABEL "DCRUNCH" GOTO Dcrunch
650   ON KEY 2 LABEL "ANALYSIS" GOTO Analysis
660   ON KEY 4 LABEL "STOREHR" GOTO Heartrate
670   ON KEY 5 LABEL "New MSI" GOTO Msi
680   ON KEY 6 LABEL "   LOAD" GOTO Load
690   ON KEY 7 LABEL "   CAT" GOTO Cat
700   ON KEY 9 LABEL "   EXIT" GOTO Exit
710   GOTO 630
720 !
730 !*** CALL TO DAPCRUNCH
740 !
750 Dcrunch:!
760   OFF KEY
770   LOAD "DAPCRUNCH:INTERNAL",1
780 !
790 !*** CALL TO CAPCRUNCH
800 !
810 Ccrunch:!
820   OFF KEY
830   LOAD "CAPCRUNCH:INTERNAL",1
840 !
850 !*** CALL TO ANALYSIS
860 !
870 Analysis:!
880   OFF KEY
890   LOAD "ANALYSIS:INTERNAL",1
900 Heartrate: !
910 !
920 !       Store heart rate routine
930 !
940   OFF KEY
950   LOAD "STOREHR:INTERNAL",1
960 !
970 !*** CATALOG REQUEST
980 !
990 Cat:!
1000  OFF KEY
1010  CAT
1020  GOTO 630
1030!
1040!*** NEW MASS STORAGE REQUEST
1050!
1060 Msi:!
1070  OFF KEY
1080  PRINT CHR$(12)
1090  PRINT "Select the device you wish to use"
1100  Dev$(1)=":INTERNAL"
1110  Dev$(2)=":HP9895,700,0"
1120  Dev$(3)=":HP9895,702,1"
1130  Dev$(4)=":HP9895,702,2"
1140  Dev$(5)=":HP9895,702,3"
1150  FOR I=1 TO 5
1160     PRINT USING "3X,D,29A,15A";I," -is the mass storage device ",Dev$(I)
1170  NEXT I
1180  INPUT "ENTER the desired device's corresponding number.",A
1190  IF A<1 OR A>5 THEN Msi
1200  MASS STORAGE IS Dev$(A)
1210  PRINT CHR$(12)
1220  GOTO 630
1230!
1240!*** LOAD PROGRAM REQUEST
1250!
1260 Load:!
1270  OFF KEY
```

```
1280  BEEP
1290  LINPUT "ENTER THE NAME OF THE PROGRAM YOU WISH TO LOAD",File_to_load$
1300  LOAD File_to_load$
1310  GOTO 630
1320!
1330!*** PROGRAM TERMINATION REQUEST
1340!
1350 Exit:!
1360  OFF KEY
1370  END
```

A7.4 CAPCRUNCH: a Basic Program Which Compacts the ASCII Data Collected by CAP2 to Numeric Format

CAPCRUNCH performs a conversion operation on the calibration data generated by the Pascal program CAP2. CAP2 stores the calibration data in an ASCII file. In order to minimize the required disk space and file access time, CAPCRUNCH converts this ASCII file to the numeric format of the Basic operating system.

A7.4.1 The Main Routine of CAPCRUNCH

The main routine of CAPCRUNCH performs all of the data conversion operations. The main routine calls the File_locations subroutine to determine the storage locations of the ASCII format calibration data file and where the numeric format calibration data is to be stored.

Common Blocks Used by the Subroutine

Common Block	Variables in the block
Anal_data1	Data_available
Cal_data1	Cal$, Date$
Cal_data2	Bin_zero_flow, Co2_dc_offset, O2_dc_offset
Cal_data3	Co2_ca, O2_cal
Cal_data4	Non_c_e_flowcal, Non_c_i_flowcal
Cal_data5	Corr_e_flowcal, Corr_i_flowcal
Cal_data6	Ta, Tb, Tc
Cal_data7	O1, Pb, Rel_humid, Room_temp, Time_delay
Resp_data1	Line1, Line2
Resp_data2	Line3, Line4
Resp_data3	Cmax, Cmin, Fmax, Fmin, Omax, Omin, Tmax,
Resp_data4	No_points
Resp_data5	C1$, O$, T$, V$

Variable definitions

Ascii_loc$ - a string variable equal to the operator specified storage location of the ASCII format calibration data.

Bin_zero_flow - an integer variable containing the sampled value equal to zero flow through the PTM.

Bin_zero_flow$ - a string variable equal to the ASCII representation of Bin_zero_flow read from the calibration file.

Cal$ - a string variable containing the name of the calibration

file.

Char_length - a real variable containing the number of total number of characters to be stored in the numeric format calibration file.

Co2_cal - a real variable equal to the CO_2 calibration factor.

Co2_cal$ - a string variable equal to the ASCII representation of Co2_cal read from the calibration file.

Co2_dc_offset - an integer variable containing the sampled CO_2 value corresponding to room air.

Co2_dc_offset$ - a string variable equal to the ASCII representation of Co2_dc_offset read from the calibration file.

Corr_e_flowcal - a real variable containing the PTG calibration factor for expiratory flows determined using temperature corrections.

Corr_e_flowcal$ - a string variable equal to the ASCII representation of Corr_e_flowcal read from the calibration file.

Corr_i_flowcal - a real variable containing the PTG calibration factor for inspiratory flows determined using temperature corrections.

Corr_i_flowcal$ - a string variable equal to the ASCII representation of Corr_i_flowcal read from the calibration file.

Data_available - an integer variable used as a Boolean value which indicates if respiratory data is present in memory.

Data_loc$ - a string variable set equal to the device specification of the 8-inch disk drive (:HP9895,700,0) which will be used to store the numeric format calibration data.

Date$ - a string variable containing the date data were collected.

Formfd$ - a single character string used as the constant value of the ASCII form feed character (12 decimal, 0C hex).

N1$ - a string variable equal to the name of the calibration file created by CAP2 as it appears to the Basic operating system.

No_integers - a real variable equal to the number of integer

values to be stored in the calibration file.

No_reals - a real variable equal to the number of real values to be stored in the calibration file.

Non_c_e_flowcal - a real variable containing the PTG calibration factor for expiratory flows determined without the use of temperature corrections.

Non_c_e_flowcal$ - a string variable equal to the ASCII representation of Non_c_e_flowcal read from the calibration file.

Non_c_i_flowcal - a real variable containing the PTG calibration factor for inspiratory flows determined without the use of temperature corrections.

Non_c_i_flowcal$ - a string variable equal to the ASCII representation of Non_c_i_flowcal read from the calibration file.

O2_cal - a real variable equal to the O_2 calibration factor.

O2_cal$ - a string variable equal to the ASCII representation of O2_cal read from the calibration file.

O2_dc_offset - an integer variable containing the sampled O_2 value corresponding to the calibration gas.

O2_dc_offset$ - a string variable equal to the ASCII representation of O2_dc_offset read from the calibration file.

O1 - a real variable equal to the fractional concentration of O_2 in the calibration gas.

O1$ - a string variable equal to the ASCII representation of O1 read from the calibration file.

Pb - a real variable equal to the barometric pressure at the time the system was calibrated.

Pb$ - a string variable equal to the ASCII representation of Pb read from the calibration file.

Rel_humid - a real variable equal to the relative humidity at the time the system was calibrated.

Rel_humid$ - a string variable equal to the ASCII representation of Rel_humid read from the calibration file.

Room_temp - a real variable equal to the room temperature at the time the system was calibrated.

Room_temp$ - a string variable equal to the ASCII representation

of Room_temp read from the calibration file.

S - a real variable equal to the sampling frequency used by CAP2.

S$ - a string variable equal to the ASCII representation of S read from the calibration file.

Ta - a real variable equal to the second order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Ta$ - a string variable equal to the ASCII representation of Ta read from the calibration file.

Tb - a real variable equal to the first order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tb$ - a string variable equal to the ASCII representation of Tb read from the calibration file.

Tc - a real variable equal to the zero order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tc$ - a string variable equal to the ASCII representation of Tc read from the calibration file.

Time_delay - an integer variable equal to the GMS time delay in milli-seconds.

Time_delay$ - a string variable equal to the ASCII representation of Time_delay read from the calibration file.

True - an integer used as the constant Boolean value of true.

A7.4.2 File_locations Subroutine

The File_locations subroutine prompts the operator for the storage location of the ASCII format calibration data. The file location where the numeric format calibration data is to be stored is always set equal to the 8-inch disk drive (:HP9895,700,0). This could be easily changed to prompt the operator for a specific storage location. This subroutine is also used by DAPCRUNCH.

Common Blocks Used by the Subroutine

None

Variables passed to the procedures

None

Variables returned by the procedures

Ascii_loc$ - string.

Data_loc$ - string.

Variable definitions

A - a real variable used as a pointer into the Msi_devices$ array.

Ascii_loc$ - a string variable equal to the operator specified storage location of the ASCII format calibration data.

Data_loc$ - a string variable set equal to the 8-inch disk drive device specification (:HP9895,700,0) which will be used to store the numeric format calibration data.

Msi_devices$ - a string array containing the device specifications of the disk drives containing the data. Element 1 specifies platter 1 of the hard disk (:HP9895,702,1), element 2 specifies platter 2 of the hard disk (:HP9895,702,2), and element 3 specifies the 8-inch disk drive (:HP9895,700,0).

A7.4.3 CAPCRUNCH Program Listing

```
10    !************************************************************************
20    !
30    !       CALIBRATION FILE, ASCII TO BINARY CONVERSION ROUTINE
40    !
50    !       HP BASIC FILENAME:  CAPCRUNCH
60    !
70    !       Department of Electrical and Computer Engineering
80    !       Kansas State University.
90    !
100   !       REVISION        DATE                    PROGRAMMER
110   !       --------        ----                    ----------
120   !         1.0           JUNE 1, 1984            LOREN E. RIBLETT
130   !         1.1           OCT  1, 1985            Michael Masters
140   !
150   !************************************************************************
160   !
170   !       PURPOSE
180   !
190   !               THIS ROUTINE CONVERTS THE CALIBRATION FILE CREATED BY
200   !               THE PASCAL ROUTINE "CAP.CODE" FROM ASCII TO BINARY,
210   !               ALLOWING FOR MORE EFFICIENT DATA STORAGE.
220   !
230   !       ROUTINE(S) CALLED BY THIS ROUTINE
240   !
250   !               AUTOST - USER PROGRAM ACCESS ROUTINE
260   !
270   !************************************************************************
280   !
290   !
300   !     NOTE 1: Converted files (binary) will be stored on the 8" floppy
310   !             ":HP9895,700,0".
320   !
330   !     NOTE 3: ASCII files are purged following the conversion process.
340   !
350   !     NOTE 4: AUTOST is called following completion of CAPCRUNCH.
360   !
370   !************************************************************************
380   COM /Resp_data1/ INTEGER Line1(1:24000),Line2(1:24000)
390   COM /Resp_data2/ INTEGER Line3(1:24000),Line4(1:24000)
400   COM /Resp_data3/ INTEGER Cmin,Cmax,Omin,Omax,Fmin,Fmax,Tmin,Tmax
410   COM /Resp_data4/ INTEGER No_points
420   COM /Resp_data5/ C1$[10],O$[10],V$[10],T$[10]
430   COM /Cal_data1/ Cal$[10],Date$[25]
440   COM /Cal_data2/ INTEGER Co2_dc_offset,O2_dc_offset,Bin_zero_flow,S
450   COM /Cal_data3/ REAL Co2_cal,O2_cal
460   COM /Cal_data4/ REAL Non_c_i_flowcal,Non_c_e_flowcal
470   COM /Cal_data5/ REAL Corr_i_flowcal,Corr_e_flowcal
480   COM /Cal_data6/ REAL Ta,Tb,Tc
490   COM /Cal_data7/ REAL Time_delay,O1,Pb,Rel_humid,Room_temp
500   COM /Anal_data1/ INTEGER Data_available
510   True=(1=1)
520   Formfd$=CHR$(12)
530   !
540   !
550   !
560   !*** ASSIGN SPECIAL FUNCTION KEYS
570   !
580   ON KEY 0 LABEL "CCRUNCH" GOTO 660
590   ON KEY 9 LABEL "  EXIT" GOTO Done
600   GOTO 580
610   !
620   !
```

```
630   !
640   !*** MAKE DECLARATIONS
650   !
660   OFF KEY
670   DIM Co2_dc_offset$[4],O2_dc_offset$[4],Bin_zero_flow$[4]
680   DIM Co2_cal$[25],O2_cal$[25],Insp_flow_cal$[25]
690   DIM Expr_flow_cal$[25],Time_delay$[25],S$[4]
700   DIM O1$[25],Ta$[25],Tb$[25],Tc$[25]
710   DIM N$[10],N1$[10],Data_loc$[13],Ascii_loc$[13]
720   !
730   !
740   !
750   !*** GET FILE NAME TO CRUNCH
760   !
770   BEEP
780   INPUT "ENTER NAME OF CALIBRATION FILE TO CRUNCH:  ",Cal$
790   File_locations(Ascii_loc$,Data_loc$)
800   !
810   IF Ascii_loc$=Data_loc$ THEN
820      PRINT TABXY(1,18);"Insert the ASCII data disk."
830      PRINT "Press CONTINUE when you are ready to proceed."
840      PAUSE
850      PRINT Formfd$
860   END IF
870   !
880   !
890   !*** ALTER FILE NAME TO REFLECT PASCAL TO BASIC NAME CHANGES
900   !
910   N1$=Cal$&"A"
920   WHILE LEN(N1$)<10
930      N1$=N1$&"_"
940   END WHILE
950 !
960 !
970 !
980 !*** READ IN ASCII VALUES FROM CALIBRATION FILE
990 !
1000  ASSIGN @Path1 TO N1$&Ascii_loc$
1010  ENTER @Path1;Co2_dc_offset$,O2_dc_offset$
1020  ENTER @Path1;Bin_zero_flow$,Co2_cal$
1030  ENTER @Path1;O2_cal$,Non_c_i_flowcal$
1040  ENTER @Path1;Non_c_e_flowcal$,Time_delay$,S$
1050  ENTER @Path1;O1$,Ta$,Tb$,Tc$,Date$
1060  ENTER @Path1;Corr_i_flowcal$,Corr_e_flowcal$
1070  ENTER @Path1;Pb$,Rel_humid$,Room_temp$
1080  ASSIGN @Path1 TO *
1090 !
1100 !
1110 !
1120 !
1130 !              Convert the ASCII calibration values to number values.
1140 !
1150  Co2_dc_offset=VAL(Co2_dc_offset$)
1160  O2_dc_offset=VAL(O2_dc_offset$)
1170  Bin_zero_flow=VAL(Bin_zero_flow$)
1180  Co2_cal=VAL(Co2_cal$)
1190  O2_cal=VAL(O2_cal$)
1200  Non_c_i_flowcal=VAL(Non_c_i_flowcal$)
1210  Non_c_e_flowcal=VAL(Non_c_e_flowcal$)
1220  Time_delay=VAL(Time_delay$)
1230  S=VAL(S$)
1240  O1=VAL(O1$)
1250  Ta=VAL(Ta$)
1260  Tb=VAL(Tb$)
1270  Tc=VAL(Tc$)
```

```
1280  Corr_i_flowcal=VAL(Corr_i_flowcal$)
1290  Corr_e_flowcal=VAL(Corr_e_flowcal$)
1300  Pb=VAL(Pb$)
1310  Rel_humid=VAL(Rel_humid$)
1320  Room_temp=VAL(Room_temp$)
1330        !
1340        !
1350        !*** CREATE BINARY FILE ON THE DATA DISK
1360        !
1370  IF Data_loc$=Ascii_loc$ THEN
1380     PRINT TABXY(1,18);"Remove the ASCII data disk and put in"
1390     PRINT "the BINARY data disk."
1400     PRINT
1410     PRINT "Press CONTINUE when you are ready to proceed."
1420     BEEP
1430     PAUSE
1440     PRINT Formfd$
1450  END IF
1460  No_integers=4    !The number of calibration factors which are integers.
1470  No_reals=13      !The number of calibration factors which are reals.
1480  Char_length=25   !The total length of the char strngs in the cal factors
1490  !
1500  !
1510  !                    Create the file
1520  !
1530  CREATE BDAT Cal$&Data_loc$,1,2*No_integers+8*No_reals+Char_length
1540        !
1550        !*** WRITE BINARY CALIBRATION CONSTANTS TO FILE
1560        !
1570  ASSIGN @Path1 TO Cal$&Data_loc$
1580  OUTPUT @Path1;Co2_dc_offset
1590  OUTPUT @Path1;O2_dc_offset
1600  OUTPUT @Path1;Bin_zero_flow
1610  OUTPUT @Path1;Co2_cal
1620  OUTPUT @Path1;O2_cal
1630  OUTPUT @Path1;Non_c_i_flowcal
1640  OUTPUT @Path1;Non_c_e_flowcal
1650  OUTPUT @Path1;Time_delay
1660  OUTPUT @Path1;S,O1
1670  OUTPUT @Path1;Ta,Tb,Tc
1680  OUTPUT @Path1;Date$
1690  OUTPUT @Path1;Corr_i_flowcal
1700  OUTPUT @Path1;Corr_e_flowcal
1710  OUTPUT @Path1;Pb,Rel_humid,Room_temp
1720  ASSIGN @Path1 TO *
1730  !
1740  !
1750  !
1760  !
1770        !*** DELETE OLD ASCII CALIBRATION FILE
1780  !
1790  IF Data_loc$=Ascii_loc$ THEN
1800     PRINT TABXY(1,18);"Remove the BINARY data disk and put in"
1810     PRINT "the ASCII data disk."
1820     PRINT
1830     PRINT "Press CONTINUE when you are ready to proceed."
1840     BEEP
1850     PAUSE
1860     PRINT Formfd$
1870  END IF
1880  PURGE N1$&Ascii_loc$
1890  PRINT "CALIBRATION FILE COMPACTION COMPLETE."
1900!
1910!*** RETURN TO AUTOST
1920!
```

```
1930  Data_available=True
1940 Done:OFF KEY
1950  MASS STORAGE IS ":INTERNAL"
1960  LOAD "AUTOST",1
1970  END
1980  SUB File_locations(Ascii_loc$,Data_loc$)
1990  !
2000     DIM Msi_devices$(1:3)[13]
2010     Msi_devices$(1)=":HP9895,702,1"
2020     Msi_devices$(2)=":HP9895,702,2"
2030     Msi_devices$(3)=":HP9895,700,0"
2040     Formfd$=CHR$(12)
2050     PRINT Formfd$
2060     PRINT
2070     PRINT "Mass Storage devices"
2080     PRINT "     1.   The hard disk, platter 1"
2090     PRINT "     2.   The hard disk, platter 2"
2100     PRINT "     3.   The 8 inch floppy disk."
2110     BEEP
2120     INPUT "Which storage location contains the ASCII data",A
2130     Ascii_loc$=Msi_devices$(A)
2140     Data_loc$=Msi_devices$(3)
2150     PRINT Formfd$
2160  SUBEND
```

A7.5 DAPCRUNCH: a Basic Program Which Compacts the ASCII Data Collected by DAP2 to Numeric Format

DAPCRUNCH performs a conversion operation on the respiratory data collected by the Pascal program DAP2. DAP2 stores the data in four ASCII files. In order to minimize the disk space and file access time, DAPCRUNCH converts these ASCII files to the numeric format of the Basic operating system.

A7.5.1 The Main Routine of DAPCRUNCH

The main routine of DAPCRUNCH performs the main control of the program. The main routine calls the File_locations, Rd_and_crunch, and Write_data subroutines. File_locations prompts the operator for the mass storage device containing the ASCII data. Rd_and_crunch reads each ASCII file and converts the data from the ASCII format to the numeric format. Write_data writes the numeric format data to disk.

Common Blocks Used by the Subroutine

Common Block	Variables in the block
Anal_data1	Data_available
Cal_data1	Cal$, Date$
Cal_data2	Bin_zero_flow, Co2_dc_offset, O2_dc_offset
Cal_data3	Co2_ca, O2_cal
Cal_data4	Non_c_e_flowcal, Non_c_i_flowcal
Cal_data5	Corr_e_flowcal, Corr_i_flowcal
Cal_data6	Ta, Tb, Tc
Cal_data7	Ol, Pb, Rel_humid, Room_temp, Time_delay
Resp_data1	Line1, Line2
Resp_data2	Line3, Line4
Resp_data3	Cmax, Cmin, Fmax, Fmin, Omax, Omin, Tmax,
Resp_data4	No_points
Resp_data5	C1$, O$, T$, V$

Variable definitions

Ascii_loc$ - a string variable equal to the operator specified storage location of the ASCII format data.

C1$ - a string variable containing the name of the CO_2 data file.

Cmax - an integer variable containing the maximum sampled datum

point obtained from the CO_2 channel, Line1.

Cmin - an integer variable containing the minimum sampled datum point obtained from the CO_2 channel, Line1.

Data_available - an integer variable used as a Boolean value which indicates if respiratory data is present in memory.

Data_loc$ - a string variable set equal to the device specification of the 8-inch disk drive (:HP9895,700,0) which will be used to store the numeric format data.

File_nm$ - a string variable which is not used within the main routine of DAPCRUNCH. The declaration of File_nm$ within the main procedure (line 630) may be deleted. The variable is used within the Rd_and_crunch subroutine.

Fmax - an integer variable containing the maximum sampled datum point obtained from the flow channel, Line3.

Fmin - an integer variable containing the minimum sampled datum point obtained from the flow channel, Line3.

Formfd$ - a single character string used as the constant value of the ASCII form feed character (12 decimal, 0C hex).

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from the CO_2 channel, channel 1, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from the O_2 channel, channel 2, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from the flow channel, channel 3, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from the temperature channel, channel 4, of the DAM.

No_points - an integer variable containing the number of samples collected.

O$ - a string variable containing the name of the O_2 data file.

Omax - an integer variable containing the maximum sampled datum point obtained from the O_2 channel, Line2.

Omin - an integer variable containing the minimum sampled datum point obtained from the O_2 channel, Line2.

T$ - a string variable containing the name of the temperature data file.

Tmax - an integer variable containing the maximum sampled datum point obtained from the temperature channel, Line4.

Tmin - an integer variable containing the minimum sampled datum point obtained from the temperature channel, Line4.

True - an integer used as the constant Boolean value of true.

V$ - a string variable containing the name of the flow data file.

A7.5.2 File locations Subroutine

The File_locations subroutine prompts the operator for the storage location of the ASCII format data. The file location where the numeric format data is to be stored is always set equal to the 8-inch disk drive (:HP9895,700,0). This could be easily changed to prompt the operator for a specific storage location. This subroutine is also used by CAPCRUNCH.

Common Blocks Used by the Subroutine

None

Variables passed to the procedures

None

Variables returned by the procedures

Ascii_loc$ - string.

Data_loc$ - string.

Variable definitions

A - a real variable used as a pointer into the Msi_devices$ array.

Ascii_loc$ - a string variable equal to the operator specified storage location of the ASCII format data.

Data_loc$ - a string variable set equal to the 8-inch disk drive device specification (:HP9895,700,0) which will be used to store the numeric format data.

Msi_devices$ - a string array containing the device specifications of the disk drives containing the data. Element 1 specifies platter 1 of the hard disk (:HP9895,702,1), element 2 specifies platter 2 of the hard

disk (:HP9895,702,2), and element 3 specifies the 8-inch disk drive (:HP9895,700,0).

A7.5.3 Rd_and_crunch Subroutine

The Rd_and_crunch (Read and crunch) subroutine performs the file conversion operation on the ASCII format data. The subroutine reads one of the ASCII format files from the disk. The data read is then converted to numeric data. The numeric data is then returned to the calling routine.

Common Blocks Used by the Subroutine

None

Variables passed to the procedures

Ascii_loc$ - string.
File_nm$ - string.
Num - integer.

Variables returned by the procedures

Array - array of integer.
Max - integer.
Min - integer.

Variable definitions

A$ - an array of string elements. The size of the array is equal to the Num parameter. The ASCII data are read into this array.

Array - an array of integers. The ASCII data are converted to numeric data and stored in Array. Array is returned to the calling routine.

Ascii_loc$ - a string variable equal to the operator specified storage location of the ASCII format data.

File_nm$ - a string variable equal to ASCII file which is to be read from disk.

Formfd$ - a single character string used as the constant value of the ASCII form feed character (12 decimal, 0C hex).

I - an integer variable used as a loop counter.

Max - an integer variable containing the maximum sampled datum point obtained from the channel being read and crunched.

Max$ - a string variable equal to the ASCII value of Max.

Min - an integer variable containing the minimum sampled datum point obtained from the channel being read and crunched.

Min$ - a string variable equal to the ASCII value of Min.

Num - an integer variable equal to the number of samples collected.

A7.5.4 Write_data Subroutine

The Write_data subroutine stores a channel of the sampled data in numeric format files. The subroutine creates the file and stores the data in the file. The file format is specified by Riblett [13].

Common Blocks Used by the Subroutine

None

Variables passed to the procedures

Data_loc$ - string.

File_nm$ - string.

Num - integer.

Variables returned by the procedures

Array - array of integer.

Max - integer.

Min - integer.

Variable definitions

Array - an array of integers. The ASCII data are converted to numeric data and stored in Array. Array is returned to the calling routine.

Data_loc$ - a string variable equal to the operator specified mass storage specification of device which will contain the numeric data.

File_nm$ - a string variable equal to the data file to be stored.

Max - an integer variable containing the maximum sampled datum point obtained from the channel being read and crunched.

Min - an integer variable containing the minimum sampled datum point obtained from the channel being read and crunched.

Num - an integer variable equal to the number of samples collected.

A7.5.5 DAPCRUNCH Program Listing

```
10    !************************************************************************
20    !
30    !       DAM DATA FILE, ASCII TO BINARY CONVERSION ROUTINE
40    !
50    !       HP BASIC FILENAME:  DAPCRUNCH
60    !
70    !       Department of Electrical and Computer Engineering
80    !       Kansas State University.
90    !
100   !       REVISION        DATE                    PROGRAMMER
110   !       --------        ----                    ----------
120   !         1.0           JUNE 1, 1984            LOREN E. RIBLETT
130   !         1.1           AUGUST 7, 1984          MICHAEL MASTERS
140   !
150   !************************************************************************
160   !
170   !       PURPOSE
180   !            THIS ROUTINE CONVERTS FOUR ASCII FILES OF DAM DATA (NAMELY
190   !            THE CO2, O2, FLOW, AND TEMPERATURE DATA FILES) CREATED
200   !            BY "DAP.CODE" TO FOUR FILES OF BINARY DATA, ALLOWING FOR
210   !            MORE EFFICIENT DATA STORAGE.
220   !
230   !       ROUTINE(S) CALLED BY THIS ROUTINE
240   !
250   !               AUTOST - USER PROGRAM ACCESS ROUTINE
260   !
270   !************************************************************************
280   !
290   !     NOTE 1:  The files to be converted must be named MONSTER1A_,
300   !              MONSTER2A_, MONSTER3A_, and MONSTER4A_.
310   !              (These files would be named MONSTER1.ASC, MONSTER2.ASC,
320   !                etc. in the PASCAL operating system.)
330   !
340   !   NOTE 2:  Converted files (binary) will be stored on the 8" floppy
350   !              ":HP9895,700,0".  The data from MONSTER1A_ is assumed to
360   !            be CO2 data, MONSTER2A_ is O2 data, MONSTER3A_ is flow
370   !            data, and MONSTER4A_ is temperature data.
380   !
390   !     NOTE 3:  Binary data file names in excess of 10 characters should
400   !              not be used.
410   !
420   !     NOTE 4:  ASCII files are purged following the conversion process.
430   !
440   !     NOTE 5:  AUTOST is called following completion of DAPCRUNCH.
450   !
460   !
470   !************************************************************************
480   OPTION BASE 1
490   COM /Resp_data1/ INTEGER Line1(1:24000),Line2(1:24000)
500   COM /Resp_data2/ INTEGER Line3(1:24000),Line4(1:24000)
510   COM /Resp_data3/ INTEGER Cmin,Cmax,Omin,Omax,Fmin,Fmax,Tmin,Tmax
520   COM /Resp_data4/ INTEGER No_points
530   COM /Resp_data5/ C1$[10],O$[10],V$[10],T$[10]
540   COM /Cal_data1/ Cal$[10],Date$[25]
550   COM /Cal_data2/ INTEGER Co2_dc_offset,O2_dc_offset,Bin_zero_flow,S
560   COM /Cal_data3/ REAL Co2_cal,O2_cal
570   COM /Cal_data4/ REAL Non_c_i_flowcal,Non_c_e_flowcal
580   COM /Cal_data5/ REAL Corr_i_flowcal,Corr_e_flowcal
590   COM /Cal_data6/ REAL Ta,Tb,Tc
600   COM /Cal_data7/ REAL Time_delay,O1,Pb,Rel_humid,Room_temp
610   COM /Anal_data1/ INTEGER Data_available
620   INTEGER True
```

```
630   DIM Ascii_loc$[13],Data_loc$[13],File_nm$[10]
640   True=(1=1)
650   Formfd$=CHR$(12)
660   !
670   !*** SPECIAL FUNCTION KEY DEFINITIONS
680   !
690   ON KEY 1 LABEL "DCRUNCH" GOTO Crunch_data
700   ON KEY 9 LABEL "  EXIT" GOTO Done
710 Idle:GOTO Idle
720 Crunch_data:OFF KEY
730   Formfd$=CHR$(12)
740   !
750   !              Determine where the ASCII data is stored and where the
760   !              BINARY data will be stored.
770   File_locations(Ascii_loc$,Data_loc$)
780   !
790   !*** GET NUMBER OF DATA POINTS AND MAKE APPROPRIATE DIMENSIONS
800   !
810   BEEP
820   INPUT "ENTER NUMBER OF DATA POINTS TO CRUNCH:  ",No_points
830   !
840   !
850   !** Prompt the user for the file names in which the data will be stored
860   BEEP
870   INPUT "ENTER NAME OF CO2 BINARY DATA FILE",C1$
880   BEEP
890   O$="O"&C1$[2,LEN(C1$)]
900   INPUT "ENTER NAME OF O2 BINARY DATA FILE:  ",O$
910   BEEP
920   V$="V"&C1$[2,LEN(C1$)]
930   INPUT "ENTER NAME OF FLOW BINARY DATA FILE:  ",V$
940   BEEP
950   T$="T"&C1$[2,LEN(C1$)]
960   INPUT "ENTER NAME OF TEMPERATURE BINARY DATA FILE:  ",T$
970   !
980   !
990   IF Data_loc$=Ascii_loc$ THEN
1000     PRINT TABXY(1,18);"Install the ASCII data disk in the 8 in. drive"
1010     PRINT
1020     PRINT "Press CONTINUE when you are ready to proceed."
1030     BEEP
1040     PAUSE
1050     PRINT Formfd$
1060  END IF
1070  !
1080  !
1090  !
1100  !
1110  !*** READ AND CRUNCH ASCII DATA MONSTER FILES.
1120  !
1130  Rd_and_crunch(("MONSTER1A_"),Ascii_loc$,No_points,Cmax,Cmin,Line1(*))
1140  Rd_and_crunch(("MONSTER2A_"),Ascii_loc$,No_points,Omax,Omin,Line2(*))
1150  Rd_and_crunch(("MONSTER3A_"),Ascii_loc$,No_points,Fmax,Fmin,Line3(*))
1160  Rd_and_crunch(("MONSTER4A_"),Ascii_loc$,No_points,Tmax,Tmin,Line4(*))
1170  !
1180  !
1190  !
1200  !                    Store the data.
1210  !
1220  IF Data_loc$=Ascii_loc$ THEN
1230     PRINT TABXY(1,18);"Take out the ASCII data disk and put in the"
1240     PRINT "BINARY data disk."
1250     PRINT
1260     PRINT "Press CONTINUE when you are ready to proceed."
1270     BEEP
```

```
1280      PAUSE
1290      PRINT Formfd$
1300    END IF
1310    Write_data(C1$,Data_loc$,No_points,Cmax,Cmin,Line1(*))
1320    Write_data(O$,Data_loc$,No_points,Cmax,Cmin,Line2(*))
1330    Write_data(V$,Data_loc$,No_points,Fmax,Fmin,Line3(*))
1340    Write_data(T$,Data_loc$,No_points,Tmax,Tmin,Line4(*))
1350    !
1360    !
1370    !                       Purge the monster files.
1380    !
1390    IF Data_loc$=Ascii_loc$ THEN
1400       PRINT TABXY(1,18);"Exchange the BINARY data disk and insert"
1410       PRINT "the ASCII data disk."
1420       PRINT
1430       PRINT "Press CONTINUE when you are ready to proceed."
1440       BEEP
1450       PAUSE
1460    END IF
1470    PURGE "MONSTER1A_"&Ascii_loc$
1480    PURGE "MONSTER2A_"&Ascii_loc$
1490    PURGE "MONSTER3A_"&Ascii_loc$
1500    PURGE "MONSTER4A_"&Ascii_loc$
1510    ASSIGN @Path TO "DISK_STATA"&Ascii_loc$
1520    OUTPUT @Path;"EMPTY"
1530    ASSIGN @Path TO *
1540    !
1550    !*** RETURN TO AUTOST
1560    !
1570    BEEP 1200,2
1580    Data_available=True
1590    PRINT Formfd$
1600    PRINT "DATA FILE COMPACTION COMPLETE. RETURNING TO AUTOST"
1610  Done:OFF KEY
1620    MASS STORAGE IS ":INTERNAL"
1630    LOAD "AUTOST",1
1640    END
1650    !
1660    !
1670    !
1680    SUB Rd_and_crunch(File_nm$,Ascii_loc$,INTEGER Num,Max,Min,Array(*))
1690       ALLOCATE A$(1:Num)[4],Max$[4],Min$[4]
1700       INTEGER I
1710       Formfd$=CHR$(12)
1720    !
1730    !
1740    !                          Read the ASCII file
1750    !
1760       DISP "Now reading in the ASCII file ";File_nm$
1770       ASSIGN @Path TO File_nm$&Ascii_loc$
1780       ENTER @Path;Max$,Min$
1790       ENTER @Path;A$(*)
1800    !
1810    !     *** CONVERT ASCII DATA TO BINARY
1820    !
1830       DISP "Converting the "&File_nm$&" file to Int. values."
1840       Max=VAL(Max$)
1850       Min=VAL(Min$)
1860       FOR I=1 TO Num
1870          Array(I)=VAL(A$(I))
1880       NEXT I
1890       ASSIGN @Path TO *
1900       DISP
1910    SUBEND
1920    !
```

```
1930  !
1940  !
1950  !
1960  SUB Write_data(File_nm$,Data_loc$,INTEGER Num,Max,Min,Array(*))
1970  !
1980  !*** CREATE BINARY DATA FILE FOR DATA
1990  !
2000      DISP "Now storing the Integer data file "&File_nm$
2010      CREATE BDAT File_nm$&Data_loc$,1,2.0*(Num+2)
2020      ASSIGN @Path TO File_nm$&Data_loc$
2030      ON END @Path GOTO Exit_sub
2040  !
2050  !
2060  !*** WRITE BINARY DATA
2070  !
2080      OUTPUT @Path;Max,Min
2090      OUTPUT @Path;Array(*)
2100  Exit_sub:ASSIGN @Path TO *
2110      DISP
2120  SUBEND
2130  !
2140  !
2150  !
2160  SUB File_locations(Ascii_loc$,Data_loc$)
2170  !
2180      DIM Msi_devices$(1:4)[13]
2190      Msi_devices$(1)=":HP9895,702,1"
2200      Msi_devices$(2)=":HP9895,702,2"
2210      Msi_devices$(3)=":HP9895,700,0"
2220      Formfd$=CHR$(12)
2230      PRINT Formfd$
2240      PRINT
2250      PRINT "Mass Storage devices"
2260      PRINT "     1.   The hard disk, platter 1"
2270      PRINT "     2.   The hard disk, platter 2"
2280      PRINT "     3.   The 8 inch floppy disk."
2290      BEEP
2300      INPUT "Which storage location contains the ASCII data",A
2310      Ascii_loc$=Msi_devices$(A)
2320      Data_loc$=Msi_devices$(3)
2330      PRINT Formfd$
2340  SUBEND
```

A7.6 ANALYSIS: a Basic Program Which Computes the Respiratory Parameters

The ANALYSIS program computes the breath-by-breath respiratory parameters from the sampled respiratory signals. ANALYSIS uses the respiratory data and performs the calculations described in Chapter IV and Appendix III. The program has the ability to display the calculated respiratory parameters in a variety of formats. The parameters may be plotted, two parameters per plot, or printed in tabular format. The parameters calculated for individual breaths may be displayed or the parameters for several breaths may be combined and averaged. Averaging is done by one of two methods, average the parameter values of a specified number of breaths or average the parameter values of breaths appearing in a window of time.

A7.6.1 The Main Routine of ANALYSIS

The main routine of ANALYSIS calculates the respiratory parameters. The routine relies on the Calc_sig_val subroutine to compute the fractional concentration of CO_2 and O_2 and the incremental volume for each sample. The Expiration and Inspiration functions are used to determine if a flow value is an inspiratory or an expiratory flow. The calculated respiratory parameters are displayed using the Disp_results subroutine. The main routine provides the plotting of the sampled data, displays the average respiratory values for the analyzed window, calculates the GMS time delay, and reads the respiratory data and calibration data from disk files.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data1	Data_available
Anal_data2	False, True
Anal_data3	Co2_prod, Insp_min_vent, O2_cons, Time_of_brth
Anal_data4	Expr_min_vent, Resp_freq, V_tid_expr, V_tid_insp
Anal_data5	Breath_good, Formfd$, Resp_quotient
Anal_data5a	Frc_co2_prod, Frc_o2_cons, Volume_of_lung
Anal_data6	Corr_flag, Room_fh2o, T
Anal_data7	Fh2o_sat, Vap

Anal_data8 - Resp_var

Anal_data9 - Comment$, Name$

Cal_data1 - Cal$, Date$

Cal_data2 - Bin_zero_flow, Co2_dc_offset, O2_dc_offset

Cal_data3 - Co2_cal, O2_cal

Cal_data4 - Non_c_e_flowcal, Non_c_i_flowcal

Cal_data5 - Corr_e_flowcal, Corr_i_flowcal

Cal_data6 - Ta, Tb, Tc

Cal_data7 - Ol, Pb, Rel_humid, Room_temp, Time_delay

Resp_data1 - Line1, Line2

Resp_data2 - Line3, Line4

Resp_data3 - Cmax, Cmin, Fmax, Fmin, Omax, Omin, Tmax,

Resp_data4 - No_points

Resp_data5 - C1$, O$, T$, V$

Variable definitions

A - an integer variable used as a pointer into the flow and temperature data arrays during integration.

Adiff - a real variable equal to the difference between Asum and Bsum.

Air_expr - a real variable equal to the volume of air expired for the current breath being analyzed.

Air_insp - a real variable equal to the volume of air inspired for the current breath being analyzed.

Asum - a real variable containing the area above the CO_2 signal based on the integration limits Beg_pnt and Beg_intg. Used in the calculation of the breath-by-breath time delay.

Avco2prod - a real variable equal to the average volume of CO_2 produced per breath

Avg_air_expr - a real variable equal to the average expiratory tidal volume of the subject.

Avg_air_insp - a real variable equal to the average expiratory tidal volume of the subject.

Avg_co2_expr - a real variable equal to the average volume of CO_2 expired per breath.

Avg_co2_insp - a real variable equal to the average volume of CO_2 inspired per breath.

Avg_o2_expr - a real variable equal to the average volume of O_2

expired per breath.

Avg_o2_insp - a real variable equal to the average volume of O_2 inspired per breath.

Avg_time_delay - a real variable equal to the average of the valid time delays computed on a breath-by-breath basis.

Avo2cons - a real variable equal to the average volume of O_2 consumed per breath.

A$ - a string constant equal to 'Air' and used in printing the analyzed results.

Begtime_colct - a real variable equal to the experimental time at which collection of the breath-by-breath data began.

Beg_intg - a real variable used as an integration marker in the determination of the breath-by-breath time delay. The area above the curve is found be integrating from the point of maximum CO_2 to Beg_intg. The area below the curve is found by integrating from Beg_intg to the ending point. Beg_intg is slid from the point of maximum CO_2 to the ending point.

Beg_pt - a real variable used as a pointer for the CO_2 signal which points to where integration above the CO_2 signal begins (used in the calculation of the breath-by-breath time delay).

Best_index - a real variable used as a pointer which points to the location in CO_2 and O_2 signal arrays corresponding to the beginning of inspiration (used in the calculation of the breath-by-breath time delay).

Best_match - a real variable containing the smallest difference in the area above and the area below the fractional CO_2 signal (used in the calculation of the breath-by-breath time delay).

Bin_zero_flow - an integer variable containing the flow sample value equal to zero flow through the PTM.

Body_temp - a real variable equal to the subject's body temperature.

Breath_count - an integer variable used as a breath counter and a pointer into the Resp_var array.

Breath_good - an integer used as a constant value which points to the good or bad breath parameter in the respiratory

variable array.

Bsum - a real variable containing the area below the CO_2 signal based on the integration limits Beg_intg and End_pt (used in the calculation of the breath-by-breath time delay)

Btps_to_stpd - a real variable equal to the BTPS to STPD conversion factor.

B_by_b_output - an integer variable used as a Boolean value. The variable is true if the breath-by-breath output is not to be omitted and is false if the output is to be omitted.

B$ - a string constant equal to 'O2' and used in printing the analyzed results.

C - a real variable equal to the Co2_dc_offset. C is used to shorten the length of the statements involved in the calculation of the breath-by-breath time delay.

Cal$ - a string variable containing the name of the calibration file.

Chng_fco2 - a real variable equal to the change in the end expirate concentration of CO_2 between the current breath being analyzed and the previous breath.

Chng_fo2 - a real variable equal to the change in the end expirate concentration of O_2 between the current breath being analyzed and the previous breath.

Chng_lung_vol - a real variable equal to the change in the lung volume between the current breath being analyzed and the previous breath.

Cmax - an integer variable containing the maximum sampled datum point obtained from the CO_2 channel, Line1.

Cmin - an integer variable containing the minimum sampled datum point obtained from the CO_2 channel, Line1.

Comment$ - a string variable containing an operator specified comment.

Corr_e_flowcal - a real variable containing the PTG calibration factor for expiratory flows determined using temperature corrections.

Corr_flag - an integer variable used as a Boolean value which indicates if the temperature corrections are to be used in computing the respiratory flow.

Corr_i_flowcal - a real variable containing the PTG calibration

factor for inspiratory flows determined using temperature corrections.

Co2max - the maximum CO_2 value for the breath (used in the computation of the breath-by-breath time delay).

Co2min - the minimum CO_2 value for the breath (used in the computation of the breath-by-breath time delay).

Co2prod - a real variable equal to the volume of CO_2 produced for the current breath being analyzed.

Co2_cal - a real variable equal to the CO_2 calibration factor.

Co2_dc_offset - an integer variable containing the sampled CO_2 value corresponding to room air.

Co2_expr - a real variable equal to the volume of CO_2 expired for the current breath being analyzed.

Co2_insp - a real variable equal to the volume of CO_2 inspired for the current breath being analyzed.

Co2_prod - an integer used as a constant value which points to the computed rate of CO_2 production parameter in the respiratory variable array.

Ctmax - a real variable equal to the temperature corresponding to Tmax.

Ctmin - a real variable equal to the temperature corresponding to Tmin.

C1$ - a string variable containing the name of the CO_2 data file.

C$ - a string constant equal to 'CO2' and used in printing the analyzed results.

Data_available - an integer variable used as a Boolean value which indicates if respiratory data are present in memory.

Date$ - a string variable containing the date data were collected.

D$ - a string constant equal to 'Inspired' and used in printing the analyzed results.

End - a real variable equal to the ending point to analyze or plot.

End_intg - a real variable equal to End_pt.

End_pt - a real variable used as a pointer to the CO_2 signal which points to where integration ends (used in the calculation of the breath-by-breath time delay).

Expr_begin - an integer variable equal to the first expiratory flow data point of the current breath being analyzed.

Expr_flowcal - a real variable set equal to Non_c_e_flowcal if temperature corrections are not used and set equal to Corr_e_flowcal if temperature corrections are used.

Expr_min_vent - an integer used as a constant value which points to the computed expiratory minute ventilation parameter in the respiratory variable array.

Expr_time - a real variable equal to the duration of the expiratory cycle of the current breath being analyzed.

E$ - a string constant equal to 'Expired' and used in printing the analyzed results.

False - an integer variable used as a constant Boolean value of False.

Fco2 - a real variable equal to the fractional concentration of CO_2 in the sample being integrated.

Fh2o_sat - a real valued, 250 element array used to hold the saturated relative concentrations of water vapor in room air between $20^{o}C$ and $45^{o}C$.

Final_index - a real variable equal to the final point in the window of data which has been analyzed.

Final_insp - an integer variable equal which points to the beginning point of the last inspiration analyzed.

Fmax - an integer variable containing the maximum sampled datum point obtained from the flow channel, Line3.

Fmin - an integer variable containing the minimum sampled datum point obtained from the flow channel, Line3.

Formfd$ - a single character string used as the constant value of an ASCII form feed character (12 decimal, 0C hex).

Found - an integer variable used as a Boolean value. Found is used in the search of the first point to be integrated. Found is false until the first inspiratory flow is found.

Fo2 - a real variable equal to the fractional concentration of O_2 in the sample being integrated.

Frac_h2o_body - the fractional concentration of water vapor at body temperature.

Frac_co2_1 - a real variable equal to the end expirate fractional concentration of CO_2 of the current breath being analyzed.

Frac_co2_2 - a real variable equal to the end expirate fractional concentration of CO_2 of the breath previous to the current breath being analyzed.

Frac_o2_1 - a real variable equal to the end expirate fractional concentration of O_2 of the current breath being analyzed.

Frac_o2_2 - a real variable equal to the end expirate fractional concentration of O_2 of the breath previous to the current breath being analyzed.

Frc_co2prod - a real variable equal to the CO_2 produced during the current breath, taking into account the change in lung volume.

Frc_co2_prod - an integer used as a constant value which points to the computed alveolar CO_2 production parameter in the respiratory variable array.

Frc_flag - a real variable used as a Boolean value which is true if FRC computations are used and is false otherwise.

Frc_o2cons - a real variable equal to the O_2 consumed during the current breath, taking into account the change in lung volume.

Frc_o2_cons - an integer used as a constant value which points to the computed alveolar O_2 consumption parameter in the respiratory variable array.

F$ - a string constant equal to '(liters)' and used in printing the analyzed results.

Good_exp_count - an integer variable used as a counter for the number of good expirations.

Good_insp_count - an integer variable used as a counter for the number of good inspirations.

G$ - a string constant equal to 'BTPS' and used in printing the analyzed results.

H$ - a string constant equal to 'STPD' and used in printing the analyzed results.

I - an integer variable used as a loop counter.

Incr - a real variable equal to an increment value which is used as the increment parameter of the FOR loops used to label the X axes in the data plotting routine.

Incr_vol - a real variable equal to the incremental volume of air computed from the flow sample.

Init_index - a real variable equal to the first point in the window of data which is analyzed.

Insp_count - an integer variable equal to the first inspiratory point of the current breath.

Insp_flowcal - a real variable set equal to Non_c_i_flowcal if temperature corrections are not used and set equal to Corr_i_flowcal if temperature corrections are used.

Insp_min_vent - an integer used as a constant value which points to the computed inspiratory minute ventilation parameter in the respiratory variable array.

Insp_time - a real variable equal to the duration of the inspiratory cycle of the current breath being analyzed.

Left - a real variable equal to the X value of the left hand side of the data plots.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from the CO_2 channel, channel 1, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from the O_2 channel, channel 2, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from the flow channel, channel 3, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from the temperature channel, channel 4, of the DAM.

Lung_vol - a real variable equal to the calculated lung volume.

Max_index - a real variable used as a pointer into the CO_2 data array where the maximum CO_2 sample for the current breath is observed. (used in the calculation of the breath-by-breath time delay).

Max_pnt - a real variable equal to the sample corresponding to the maximum fractional concentration of CO_2 of the current breath being analyzed.

Mid_index - a real variable used as a pointer into the CO_2 data array where one-half of the maximum fractional concentration of CO_2 is observed in the current breath (used in the calculation of the breath-by-breath time

delay).

Minvole - a real variable equal to the average expiratory minute ventilation of the subject.

Minvoli - a real variable equal to the average inspiratory minute ventilation of the subject.

Min_index - a real variable used as a pointer into the CO_2 data array where the minimum CO_2 sample for the current breath is observed. (used in the calculation of the breath-by-breath time delay).

Name$ - a string variable containing an operator specified subject name.

Net_lung_chng - a real variable equal to the net change in lung volume for the analyzed window of data.

Non_c_e_flowcal - a real variable containing the PTG calibration factor for expiratory flows determined without the use of temperature corrections.

Non_c_i_flowcal - a real variable containing the PTG calibration factor for inspiratory flows determined without the use of temperature corrections.

No_points - an integer variable containing the number of samples collected.

Num_breaths - a real variable equal to the number of breaths that were analyzed.

Offset - a real variable added to the CO_2 and O_2 array pointers for the plotting of time aligned signals.

O1 - a real variable equal to the fractional concentration of O_2 in the calibration gas.

Omax - an integer variable containing the maximum sampled datum point obtained from the O_2 channel, Line2.

Omin - an integer variable containing the minimum sampled datum point obtained from the O_2 channel, Line2.

O2cons - a real variable equal to the volume of O_2 consumed for the current breath being analyzed.

O2_cal - a real variable equal to the O_2 calibration factor.

O2_cons - an integer used as a constant value which points to the computed rate of O_2 consumption parameter in the respiratory variable array.

O2_dc_offset - an integer variable containing the sampled O_2

value corresponding to the calibration gas.

O2_expr - a real variable equal to the volume of O_2 expired for the current breath being analyzed.

O2_insp - a real variable equal to the volume of O_2 inspired for the current breath being analyzed.

O$ - a string variable containing the name of the O_2 data file.

P - a real variable equal to the number of data points to be plotted.

Pb - a real variable equal to the barometric pressure at the time the system was calibrated.

Ph2o_body - a real variable equal to the saturated partial pressure of water vapor at body temperature.

Q$ - a string variable used to obtain the operator's response to a question.

R - a real variable equal to the respiratory quotient.

Rel_humid - a real variable equal to the relative humidity at the time the system was calibrated.

Respf - a real variable equal to the average respiratory frequency.

Resp_freq - an integer used as a constant value which points to the computed respiratory frequency parameter in the respiratory variable array.

Resp_quotient - an integer used as a constant value which points to the computed respiratory quotient parameter in the respiratory variable array.

Resp_var - a real valued array containing the calculated breath-by-breath respiratory parameters. Resp_var is a two dimensional array, the first dimension is the respiratory variable and the second dimension is the breath number. A more detailed description of the respiratory variable array is given in Section A7.2 of this appendix.

Right - a real variable equal to the X value on the right hand side of the data plots.

Room_fh2o - a real variable equal to the concentration of water vapor in room air.

Room_temp - a real variable equal to the room temperature at the time the system was calibrated.

S - a real variable equal to the sampling frequency.

Start - a real variable equal to the beginning point to be analyzed or plotted.

Still_calc - a real variable used as a Boolean value which indicates if data remains in the window to be analyzed.

Stpd_to_btps - a real variable equal to the STPD to BTPS volume conversion factor.

T - a real variable equal to the sampling period used to collect the respiratory data.

Ta - a real variable equal to the second order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tb - a real variable equal to the first order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tc - a real variable equal to the zero order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Temp - a real variable equal to the temperature corresponding to a sample of the temperature channel.

Temporary - a real variable used as an intermediate variable. Temporary is used when the average results are printed.

Temp_a - a real variable used by the breath-by-breath time delay routine to preserve the variable 'A', the flow and temperature signal pointer.

Temp_z - a real variable used by the breath-by-breath time delay routine to preserve the variable 'Z', the CO_2 and O_2 signal pointer.

Time_delay - an integer variable equal to the GMS time delay in milli-seconds.

Time_delay_cnt - a real variable equal to the number of valid time delays computed on a breath-by-breath basis.

Time_delay_flag - a real variable which to indicate the type of time delay used. The flag is equal to zero for a fixed time delay, equal to one for a variable time delay, and equal to two for a variable time delay with an invalid computed time delay.

Time_delay_sum - a real variable equal to the sum of the valid

time delays and is used with Time_delay_cnt to compute the Avg_time_delay.

Time_of_brth - an integer used as a constant value which points to the time of breath parameter in the respiratory variable array.

Tmax - an integer variable containing the maximum sampled datum point obtained from the temperature channel, Line4.

Tmin - an integer variable containing the minimum sampled datum point obtained from the temperature channel, Line4.

Tot_co2_exp - a real variable equal to the total volume of CO_2 expired in the analyzed window of data.

Tot_co2_insp - a real variable equal to the total volume of CO_2 inspired for all breaths in the analyzed window of data.

Tot_co2_prod - a real variable equal to the total volume of CO_2 produced for all breaths in the analyzed window of data.

Tot_o2_cons - a real variable equal to the total volume of O_2 consumed for all breaths in the analyzed window of data.

Tot_o2_exp - a real variable equal to the total volume of O_2 expired in the analyzed window of data.

Tot_o2_insp - a real variable equal to the total volume of O_2 inspired for all breaths in the analyzed window of data.

Tot_time - a real variable equal to the duration of the respiratory cycle currently being analyzed.

Tot_time_expr - a real variable equal to the total time of expiration during the analyzed window of data.

Tot_time_insp - a real variable equal to the total time of inspiration during the analyzed window of data.

Tot_time_resp - a real variable equal to the total time of respiration.

Tot_vol_exp - a real variable equal to the total volume expired in the analyzed window of data.

Tot_vol_insp - a real variable equal to the total volume inspired for all breaths in the analyzed window of data.

True - an integer used as the constant Boolean value of true.

TS - a string variable containing the name of the temperature data file.

Vap - a real valued, 250 element array used to hold the saturated water vapor pressure between 20°C and 45°C.

Volume_of_lung - an integer used as a constant value which points to the computed lung volume parameter in the respiratory variable array.

V_dot_co2 - a real variable equal to the average rate of CO_2 production.

V_dot_o2 - a real variable equal to the average rate of O_2 consumption.

V_tid_expr - an integer used as a constant value which points to the computed expiratory tidal volume parameter in the respiratory variable array.

V_tid_insp - an integer used as a constant value which points to the computed inspiratory tidal volume parameter in the respiratory variable array.

V$ - a string variable containing the name of the flow data file.

Wye - a real variable used as a loop counter in the breath-by-breath time delay calculation.

Z - an integer variable used as a pointer into the CO_2 and O_2 data arrays during integration.

A7.6.2 Avg_breath Subroutine

The Avg_breath subroutine is used to compute the average of a specified number of breath-by-breath values of any respiratory parameter in the respiratory variable array. Those breaths specified as 'bad' breaths may be omitted in the computation of the average. The subroutine is used by the Max_min, Plot_value, and Print_results subroutines.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data3	Co2_prod, Insp_min_vent, O2_cons, Time_of_brth
Anal_data5	Breath_good, Formfd$, Resp_quotient
Anal_data5a	Frc_co2_prod, Frc_o2_cons, Volume_of_lung
Anal_data8	Resp_var
Plot_resp_data1	Omit_bad, Num_breaths, No_b_avg

Variables passed to the subroutine

Br_pntr - integer.

Variable - integer.

Variables returned by the subroutine

Average - real.

Num_bad_breaths - integer.

Time - real.

Variable definitions

Average - a real variable equal to the average of the specified number of breath-by-breath values of the specified respiratory parameter.

Breath_good - an integer used as a constant value which points to the good or bad breath parameter in the respiratory variable array.

Br_pntr - an integer variable used as a pointer into the breath dimension of the respiratory variable array. Br_pntr points to the first breath to be included in the average.

J - an integer variable used as a loop counter.

No_b_avg - an integer variable equal to the number of breath-by-breath values to be averaged.

Num_bad_breaths - an integer variable equal to the number of bad breaths that are not included in the average.

Num_breaths - a real variable equal to the number of breaths to be analyzed.

Omit_bad - an integer variable used as a Boolean flag which indicates if the 'bad' breaths are not to be included in the average.

Pointer - an integer variable used as a pointer into the breath dimension of the respiratory variable array.

Resp_var - a real valued array containing the calculated breath-by-breath respiratory parameters.

Time - a real variable equal to the time that is midway between the first breath included in the average and the last breath included in the average.

Time_of_brth - an integer used as a constant value which points to the time of breath parameter in the respiratory variable array.

Variable - an integer variable used as a pointer into the respiratory variable dimension of the respirator variable array. Variable is equal to the respiratory parameter to be averaged.

A7.6.3 Avg_window Subroutine

The Avg_window subroutine is used to compute the average of several breath-by-breath values for all breaths occurring in a specified window in time. Any respiratory parameter in the respiratory variable array may be averaged. Those breaths specified as 'bad' breaths may be omitted in the computation of the average. The subroutine is used by the Max_min, Plot_value, and Print_results subroutines.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data3	Co2_prod, Insp_min_vent, O2_cons, Time_of_brth
Anal_data5	Breath_good, Formfd$, Resp_quotient
Anal_data5a	Frc_co2_prod, Frc_o2_cons, Volume_of_lung
Anal_data8	Resp_var
Plot_resp_data1	Omit_bad, Num_breaths, No_b_avg
Plot_resp_data2	Start_time, End_time, W_period, W_wdth

Variables passed to the subroutine

Br_pntr - integer.

Variable - integer.

W_cntr - real.

Variables returned by the subroutine

Average - real.

Br_in_w_flag - integer.

Br_pntr - integer.

Variable definitions

Average - a real variable equal to the average of all breaths occuring in the time window.

Br_in_w_flag - an integer variable used as a Boolean value which indicates if breaths were found in the specified time

window.

Breath_good - an integer used as a constant value which points to the good or bad breath parameter in the respiratory variable array.

Br_pntr - an integer variable used as a pointer into the breath dimension of the respiratory variable array. Br_pntr points to the first breath to be included in the average.

Fstbr_in_w_fnd - an integer variable used as a Boolean value which indicates when the first breath in the specified window is found.

Left_edge_w - a real variable equal to the time of the left side of the window.

Num_breaths - a real variable equal to the number of breaths that were analyzed.

Num_b_in_window - an integer variable equal to the number of breaths in the window which were averaged.

Omit_bad - an integer variable used as a Boolean flag which indicates if the 'bad' breaths are not to be included in the average.

Resp_var - a real valued array containing the calculated breath-by-breath respiratory parameters.

Time_of_brth - an integer used as a constant value which points to the time of breath parameter in the respiratory variable array.

True - an integer used as the constant Boolean value of true.

Variable - an integer variable used as a pointer into the respiratory variable dimension of the respirator variable array. Variable is equal to the respiratory parameter to be averaged.

W_cntr - a real variable equal to the time of the center of the window.

W_wdth - a real variable equal to the time width of the window.

A7.6.4 Calc sig val Subroutine

The Calc_sig_val subroutine calculates the physical signal values from one set of samples of the four channels. The subroutine converts the CO_2 sample to the fractional concentration of CO_2 and the O_2 sample to the fractional concentration of O_2. The incremental volume for the

sample from the flow sample is calculated. If the temperature and viscosity corrections are to be performed the incremental volume is converted to STPD conditions. This conversion is accomplished using the temperature sample, ambient conditions, the fractional gas concentrations, and the relative viscosity function, Rel_viscos (A7.6.14). The Calc_sig_val subroutine is similar to the Calc_Incr_Vol procedure (A6.1.4) used by the calibration program, CAP2 (A6.1).

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data6	Corr_flag, Room_fh2o, T
Anal_data7	Fh2o_sat, Vap
Cal_data2	Bin_zero_flow, Co2_dc_offset, O2_dc_offset
Cal_data3	Co2_cal, O2_cal
Cal_data6	Ta, Tb, Tc
Cal_data7	O1, Pb, Rel_humid, Room_temp, Time_delay
Resp_data1	Line1, Line2
Resp_data2	Line3, Line4

Variables passed to the procedure

A - integer.

Insp_or_expr$ - string.

Z - integer.

Variables returned by the procedure

Fco2 - real.

Fo2 - real.

Incr_vol - real.

Variable definitions

A - an integer variable used as a pointer into the flow and temperature data arrays during integration.

Bin_flow - a real variable proportional to the flow. The value is computed by subtracting the value of zero flow and dividing by the relative viscosity.

Bin_zero_flow - an integer variable containing the flow sample value corresponding to zero flow through the PTM.

Corr_flag - an integer variable used as a Boolean value which indicates if the temperature corrections are used in

computing the respiratory flow.

Co2_cal - a real variable equal to the CO_2 calibration factor.

Co2_dc_offset - an integer variable containing the sampled CO_2 value corresponding to room air.

Fco2 - a real variable equal to the fractional concentration of CO_2 calculated from the CO_2 sample. This concentration value neglects the amount of water vapor in the respiratory flow gas.

Fco2_wet - a real variable equal to the fractional concentration of CO_2 in the wet respiratory gas. This concentration value takes into account the amount of water vapor in the respiratory flow gas.

Fh2o - a real variable equal to the fractional concentration of water vapor in the respiratory flow gas.

Fh2o_sat - a real valued, 250 element array used to hold the saturated relative concentrations of water vapor in room air between $20^{o}C$ and $45^{o}C$.

Fn2_wet - a real variable equal to the fractional concentration of N_2 in the wet respiratory gas. This concentration value takes into account the amount of water vapor in the respiratory flow gas.

Fo2 - a real variable equal to the fractional concentration of O_2 calculated from the O_2 sample. This concentration value neglects the amount of water vapor in the respiratory flow gas.

Fo2_wet - a real variable equal to the fractional concentration of O_2 in the wet respiratory gas. This concentration value takes into account the amount of water vapor in the respiratory flow gas.

Incr_vol - a real variable proportional to the incremental volume of gas computed from the flow sample. Multiplying Incr_vol by the flow calibration factor yields the incremental volume in liters. However, the multiplication of the sample by the flow calibration factor is done after the entire respiratory cycle is integrated in order to reduce computation time in the integration loop.

Insp_or_expr$ - a string variable used to indicate if the sample being analyzed is an inspiratory or an expiratory flow.

Line1 - an integer array, of 24,000 elements, containing the sampled values obtained from the CO_2 channel, channel 1, of the DAM.

Line2 - an integer array, of 24,000 elements, containing the sampled values obtained from the O_2 channel, channel 2, of the DAM.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from the flow channel, channel 3, of the DAM.

Line4 - an integer array, of 24,000 elements, containing the sampled values obtained from the temperature channel, channel 4, of the DAM.

O1 - a real variable equal to the fractional concentration of O_2 in the calibration gas.

O2_cal - a real variable equal to the O_2 calibration factor.

O2_dc_offset - an integer variable containing the sampled O_2 value corresponding to the calibration gas.

Pb - a real variable equal to the barometric pressure at the time the system was calibrated.

Rel_Viscosity - a real variable equal to the viscosity of the respiratory gas relative to the viscosity of room air under STPD conditions.

Room_fh2o - a real variable equal to the relative concentration of water vapor in room air.

T - a real variable equal to the sampling period used to collect the respiratory data.

Ta - a real variable equal to the second order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tb - a real variable equal to the first order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Tc - a real variable equal to the zero order calibration coefficient for converting the collected temperature data to units of ^{o}C.

Temperature - a real variable equal to the temperature of the flow gas.

Z - an integer variable used as a pointer into the CO_2 and

O_2 data arrays during integration.

A7.6.5 Disp_results Subroutine

The Disp_results subroutine provides the main control for the display of the respiratory parameters. The subroutine provides the ability to generate two types of respiratory parameter output, a plot of any two respiratory parameters or a printed copy of all respiratory parameters. The values of a respiratory parameter from several breaths may be averaged by two techniques: breath averaging or window averaging.

Common Block	Variables in the block
Anal_data2	False, True
Anal_data3	Co2_prod, Insp_min_vent, O2_cons, Time_of_brth
Anal_data4	Expr_min_vent, Resp_freq, V_tid_expr, V_tid_insp
Anal_data5	Breath_good, Formfd$, Resp_quotient
Anal_data5a	Frc_co2_cons, Frc_o2_cons, Volume_of_lung
Anal_data8	Resp_var
Anal_data9	Comment$, Name$
Cal_data1	Cal$, Date$
Plot_resp_data1	Omit_bad, Num_breaths, No_b_avg
Plot_resp_data2	Start_time, End_time, W_period, W_wdth

Variables passed to the procedure

Breaths - integer.

Corr_flag - integer.

Variables returned by the procedure

None

Variable definitions

Avg_bad_comnt$ - a string variable which is used to label the plot with the type of averaging technique used and indicate if bad breaths are omitted.

Avg_type$ - a string variable which indicates the type of averaging technique used to average the respiratory parameters.

Breaths - an integer variable equal to the number of breaths contained in the respiratory parameter array.

Corr_flag - an integer variable used as a Boolean value which indicates if the temperature corrections are used in computing the respiratory flow.

Date$ - a string variable containing the date the data were collected.

End_time - a real variable equal to the ending time of the respiratory results display.

Event1$ - a string variable used as an operator defined event corresponding to the time defined by Time1$. The event is labeled on plots of respiratory data.

Event2$ - a string variable used as an operator defined event corresponding to the time defined by Time2$. The event is labeled on plots of respiratory data.

False - an integer variable used as a constant Boolean value of False.

Fig1 - an integer variable equal to the first respiratory parameter to be plotted.

Fig2 - an integer variable equal to the second respiratory parameter to be plotted.

Formfd$ - a single character string used as the constant value of an ASCII form feed character (12 decimal, 0C hex).

Max1 - a real variable equal to the maximum value to be plotted for the first respiratory variable.

Max2 - a real variable equal to the maximum value to be plotted for the second respiratory variable.

Min1 - a real variable equal to the minimum value to be plotted for the first respiratory variable.

Min2 - a real variable equal to the minimum value to be plotted for the second respiratory variable.

Name$ - a string variable containing the subject name.

No_b_avg - an integer variable equal to the number of breaths to average when performing breath averaging.

Num_breaths - a real variable equal to the number of breaths in the respiratory variable array.

Omit_bad - an integer variable used as a Boolean flag which indicates if the 'bad' breaths are not to be included in the average.

Plot_dev$ - a string variable containing the plotting device to

be used to plot the respiratory parameters.

Q$ - a string variable used to obtain the operator's response to a question.

Resp_quotient - an integer used as a constant value which points to the computed respiratory quotient parameter in the respiratory variable array.

Resp_var - a real valued array containing the calculated breath-by-breath respiratory variables. Resp_var is a two dimensional array, the first dimension is the respiratory variable and the second dimension is the breath number. A more detailed description of the respiratory variable array is given in Section A7.2 of this appendix.

Result_type$ - a string variable used to indicate the type of output used to display the respiratory parameters.

Start_time - a real variable equal to the starting time of the respiratory results display.

Time1$ - a string variable used as an operator defined time mark. The time is labeled on plots of respiratory data.

Time2$ - a string variable used as an operator defined time mark. The time is labeled on plots of respiratory data.

Title$ - a string variable used as an operator defined title to be placed on the plot.

Time_of_brth - an integer used as a constant value which points to the beginning time of each breath in the respiratory variable array.

True - an integer used as the constant Boolean value of true.

W_period - a real variable equal to the period of which the window is slid.

W_wdth - a real variable equal to the time width of the window.

A7.6.6 Expand dump Subroutine

The Expanded_dump subroutine generates a full page graphics dump of the HP9826's screen on the thermal printer. Expanded_dump is not called within the program but may be called by the operator. While the program is paused the operator must type the command below and press the 'EXECUTE' key.

```
CALL Expanded_dump
```

Common blocks used by the subroutine

None

Variables passed to the subroutine

None

Variables returned by the subroutine

None

Variable definitions

None

A7.6.7 Expiration Function

The Expiration function tests a flow sample to determine if the flow is an expiratory flow. The function returns an integer evalue which acts as a Boolean variable. The returned value is equal to true if the flow is an expiratory flow and is equal to false if the flow is not an expiratory flow. An expiratory flow sample must meet one of the following two conditions.

1. The flow sample is greater than the sample corresponding to zero flow, or
2. one of the next five flow samples are greater than or equal to zero flow.

These conditions were defined by Riblett [13].

Common Blocks Used by the function

Common Block	Variables in the block
Cal_data2	Bin_zero_flow, Co2_dc_offset, O2_dc_offset
Resp_data2	Line3, Line4

Variables passed to the function

Pntr - integer.

Variable definitions

Bin_zero_flow - an integer variable containing the flow sample corresponding to zero flow through the PTM.

Expiration - an integer variable used as a Boolean value which

indicates if the sample is an expiratory flow.

I - an integer variable used as a loop counter.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from the flow channel, channel 3, of the DAM.

Pntr - an integer variable equal to the flow sample being tested.

A7.6.8 Inspiration Function

The Inspiration function tests a flow sample to determine if the flow is an inspiratory flow. The function returns an integer value which acts as a Boolean variable. The returned value is equal to true if the flow is an inspiratory flow and is equal to false if the flow is not an inspiratory flow. An inspiratory flow sample must meet one of the following two conditions.

1. The flow sample is less than the sample corresponding to zero flow, or
2. one of the next five flow samples are less than or equal to zero flow.

These conditions were defined by Riblett [13].

Common Blocks Used by the function

Common Block	Variables in the block
Cal_data2	Bin_zero_flow, Co2_dc_offset, O2_dc_offset
Resp_data2	Line3, Line4

Variables passed to the function

Pntr - integer.

Variable definitions

Bin_zero_flow - an integer variable containing the flow sample corresponding to zero flow through the PTM.

Inspiration - an integer variable used as a Boolean value which indicates if the sample is an inspiratory flow.

I - an integer variable used as a loop counter.

Line3 - an integer array, of 24,000 elements, containing the sampled values obtained from the flow channel, channel 3,

of the DAM.

Pntr - an integer variable equal to the flow sample being tested.

A7.6.9 Make_axes Subroutine

The Make_axes subroutine generates a set of axes for plots of the respiratory parameters. The scale of the plot is declared from the maximum and minimum values to be plotted and the starting and ending times. The X-axis is labeled with the units of time and the Y-axis is labeled with the respiratory parameter to be plotted.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data3	Co2_prod, Insp_min_vent, O2_cons, Time_of_brth
Anal_data4	Expr_min_vent, Resp_freq, V_tid_expr, V_tid_insp
Anal_data5	Breath_good, Formfd$, Resp_quotient
Anal_data5a	Frc_co2_cons, Frc_o2_cons, Volume_of_lung
Plot_resp_data2	Start_time, End_time, W_period, W_wdth

Variables passed to the subroutine

Corr_flag - integer.

Max - real.

Min - real.

Variable - integer.

Variables returned by the subroutine

None

Variable definitions

Bottom - a real variable equal to the bottom of the plot in terms of the X dimension. The value is slightly less than the minimum value to be plotted.

Co2_prod - an integer used as a constant value which points to the computed rate of CO_2 production parameter in the respiratory variable array.

Corr_flag - an integer variable used as a Boolean value which indicates if the temperature corrections are used in computing the respiratory flow.

End_time - a real variable equal to the ending time of the

respiratory results display.

Expr_min_vent - an integer used as a constant value which points to the computed expiratory minute ventilation parameter in the respiratory variable array.

Frc_co2_prod - an integer used as a constant value which points to the computed alveolar CO_2 production parameter in the respiratory variable array.

Frc_o2_cons - an integer used as a constant value which points to the computed alveolar O_2 consumption parameter in the respiratory variable array.

Headings$ - a string array of eleven elements. The elements contain heading information of the respiratory parameters. The element specified by Variable is used to label the Y axes of the plot.

Insp_min_vent - an integer used as a constant value which points to the computed inspiratory minute ventilation parameter in the respiratory variable array.

Left - a real variable equal to the X value of the left hand side of the data plots.

Max - a real variable equal to the maximum value to be plotted.

Min - a real variable equal to the minimum value to be plotted.

O2_cons - an integer used as a constant value which points to the computed rate of O_2 consumption parameter in the respiratory variable array.

Resp_freq - an integer used as a constant value which points to the computed respiratory frequency parameter in the respiratory variable array.

Resp_quotient - an integer used as a constant value which points to the computed respiratory quotient parameter in the respiratory variable array.

Right - a real variable equal to the X value on the right hand side of the data plots.

Start_time - a real variable equal to the starting time of the respiratory results display.

Top - a real variable equal to the top of the plot in terms of the X dimension. The value is slightly greater than the

maximum value to be plotted.

Variable - an integer variable which indicates which respiratory parameter is to be plotted.

Volume_of_lung - an integer used as a constant value which points to the computed lung volume parameter in the respiratory variable array.

V_tid_expr - an integer used as a constant value which points to the computed expiratory tidal volume parameter in the respiratory variable array.

V_tid_insp - an integer used as a constant value which points to the computed inspiratory tidal volume parameter in the respiratory variable array.

X - a real variable used as a loop counter to label the time marks on the X axes.

Xtick - the time mark spacing on the X axes.

Y - a real variable used as a loop counter to label values on the Y axes.

Ytick - the spacing of tick marks on the Y axes.

A7.6.10 Max_co2_pntr Function

The Max_co2_pntr function is used to determine the end expirate CO_2 concentration. The function determines a pointer to the maximum fractional concentration of CO_2 during the final stages of expiration. The CO_2 concentration corresponding to this sample is assumed to be the end expirate sample. This subroutine was developed by Pieschl [12] for use with FRC calculations.

Common blocks used by the function

None

Variables passed to the function

A - integer.

Line1 - integer array.

Variable definitions

A - an integer variable used as a pointer into the flow and temperature data arrays during integration.

I - an integer variable used as a loop counter.

Line1 - an integer array, of 24,000 elements, containing the

sampled values obtained from the CO_2 channel, channel 1, of the DAM.

Max - an integer variable equal to the element in the CO_2 array determined to be the maximum sample.

A7.6.11 Max_min Subroutine

The Max_min subroutine is used to determine the maximum and minimum values of a respiratory parameter. The subroutine uses the specified averaging technique (breath averaging or window averaging) to average the values of the parameter. The maximum and minimum values are used to determine the scale for a plot of a respiratory parameter.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data8	Resp_var
Plot_resp_data1	Omit_bad, Num_breaths, No_b_avg
Plot_resp_data2	Start_time, End_time, W_period, W_wdth

Variables passed to the subroutine

Avg_type$ - string.

Variable - integer.

Variables returned by the subroutine

Max - real.

Min - real.

Variable definitions

Average - a real variable equal to the average value obtained from the averaging subroutines.

Avg_type$ - a string variable which indicates the type of averaging technique used to average the respiratory parameters.

Br_in_w_flag - an integer variable used as a Boolean value which indicates if breaths were found in the specified time window. This flag is used when performing window averaging.

Br_pntr - an integer variable used as a pointer into the breath dimension of the respiratory variable array. Br_pntr points to the first breath to be included in the average.

End_time - a real variable equal to the ending time of the

respiratory results display.

Max - a real variable equal to the maximum value of the respiratory parameter.

Min - a real variable equal to the minimum value of the respiratory parameter.

No_b_avg - an integer variable equal to the number of breaths to average when performing breath averaging.

Num_bad_breaths - an integer variable equal to the number of bad breaths that are found in a window of time when using the window averaging technique.

Num_breaths - a real variable equal to the number of breaths in the respiratory variable array.

Start_time - a real variable equal to the starting time of the respiratory results display.

Time - a real variable equal to the time that is midway between the first breath included in the average and the last breath included in the average. The Time variable is used only as a dummy parameter when calling the Avg_breath subroutine.

Variable - an integer variable which indicates which respiratory parameter is to be plotted.

W_cntr - a real variable equal to the time of the center of the window.

W_wdth - a real variable equal to the time width of the window.

A7.6.12 Plot values Subroutine

The Plot_values subroutine generates a plot of the respiratory parameter. The subroutine uses the specified averaging technique to determine the average of the respiratory parameter. The subroutine plots the values on the scale generated by the Make_axes subroutine.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data8	Resp_var
Plot_resp_data1	Omit_bad, Num_breaths, No_b_avg
Plot_resp_data2	Start_time, End_time, W_period, W_wdth

Variables passed to the subroutine

Avg_type$ - string.

Variable - integer.

Variables returned by the subroutine

None

Variable definitions

Average - a real variable equal to the average value obtained from the averaging subroutines.

Avg_type$ - a string variable which indicates the type of averaging technique used to average the respiratory parameters.

Br_in_w_flag - an integer variable used as a Boolean value which indicates if breaths were found in the specified time window. This flag is used when performing window averaging.

Br_pntr - an integer variable used as a pointer into the breath dimension of the respiratory variable array. Br_pntr points to the first breath to be included in the average.

End_time - a real variable equal to the ending time of the respiratory results display.

No_b_avg - an integer variable equal to the number of breaths to average when performing breath averaging.

Num_bad_breaths - an integer variable equal to the number of bad breaths that are found in a window of time when using the window averaging technique.

Num_breaths - a real variable equal to the number of breaths in the respiratory variable array.

Start_time - a real variable equal to the starting time of the respiratory results display.

Time - a real variable equal to the time that is midway between the first breath included in the average and the last breath included in the average. This value is used as the time value of the data point when using the breath averaging technique.

Variable - an integer variable which indicates which respiratory parameter is to be plotted.

W_cntr - a real variable equal to the time of the center of the

window.

W_period - a real variable equal to the period of which the window is slid.

A7.6.13 Print values Subroutine

The Print_values subroutine generates a printout of all respiratory parameters. The subroutine uses the specified averaging technique to determine the average of the respiratory parameters.

Common blocks used by the subroutine

Common Block	Variables in the block
Anal_data3	Co2_prod, Insp_min_vent, O2_cons, Time_of_brth
Anal_data4	Expr_min_vent, Resp_freq, V_tid_expr, V_tid_insp
Anal_data5	Breath_good, Formfd$, Resp_quotient
Anal_data5a	Frc_co2_cons, Frc_o2_cons, Volume_of_lung
Anal_data6	Corr_flag, Room_fh2o, T
Anal_data8	Resp_var
Anal_data9	Comment$, Name$
Cal_data1	Cal$, Date$
Plot_resp_data1	Omit_bad, Num_breaths, No_b_avg
Plot_resp_data2	Start_time, End_time, W_period, W_wdth

Variables passed to the subroutine

Avg_type$ - string.

Variables returned by the subroutine

None

Variable definitions

Average - a real variable equal to the average value obtained from the averaging subroutines.

Avg_type$ - a string variable which indicates the type of averaging technique used to average the respiratory parameters.

Br_in_w_flag - an integer variable used as a Boolean value which indicates if breaths were found in the specified time window. This flag is used when performing window averaging.

Br_index - an integer variable which is set equal to the value of Br_pntr. Br_index is used when the Avg_window subroutine

is called so that the value of Br_pntr is not changed.

Br_pntr - an integer variable used as a pointer into the breath dimension of the respiratory variable array. Br_pntr points to the first breath to be included in the average.

Co2_prod - an integer used as a constant value which points to the computed rate of CO_2 production parameter in the respiratory variable array.

Comment$ - a string variable containing an operator specified comment.

Corr_flag - an integer variable used as a Boolean value which indicates if the temperature corrections are used in computing the respiratory flow.

Date$ - a string variable containing the date data were collected.

End_time - a real variable equal to the ending time of the respiratory results display.

Expr_min_vent - an integer used as a constant value which points to the computed expiratory minute ventilation parameter in the respiratory variable array.

Frc_co2_prod - an integer used as a constant value which points to the computed alveolar CO_2 production parameter in the respiratory variable array.

Frc_o2_cons - an integer used as a constant value which points to the computed alveolar O_2 consumption parameter in the respiratory variable array.

I - an integer variable used as a loop counter.

Insp_min_vent - an integer used as a constant value which points to the computed inspiratory minute ventilation parameter in the respiratory variable array.

Name$ - a string variable containing the subject name.

No_b_avg - an integer variable equal to the number of breath to average when performing breath averaging.

Num_bad_breaths - an integer variable equal to the number of bad breaths that are found in a window of time when using the window averaging technique.

Num_breaths - a real variable equal to the number of breaths in the respiratory variable array.

Omit_bad - an integer variable used as a Boolean flag which

indicates if the 'bad' breaths are not to be included in the average.

O2_cons - an integer used as a constant value which points to the computed rate of O_2 consumption parameter in the respiratory variable array.

Resp_freq - an integer used as a constant value which points to the computed respiratory frequency parameter in the respiratory variable array.

Resp_quotient - an integer used as a constant value which points to the computed respiratory quotient parameter in the respiratory variable array.

Some_rs_to_big - an integer variable used as a Boolean value which indicates that some values of the respiratory quotient average are too large to fit the print format.

Start_time - a real variable equal to the starting time of the respiratory results display.

Time - a real variable equal to the time that is midway between the first breath included in the average and the last breath included in the average.

Volume_of_lung - an integer used as a constant value which points to the computed lung volume parameter in the respiratory variable array.

V_tid_expr - an integer used as a constant value which points to the computed expiratory tidal volume parameter in the respiratory variable array.

V_tid_insp - an integer used as a constant value which points to the computed inspiratory tidal volume parameter in the respiratory variable array.

W_cntr - a real variable equal to the time of the center of the window.

W_period - a real variable equal to the period of which the window is slid.

W_wdth - a real variable equal to the time width of the window.

A7.6.14 Rel_viscos Function

The Rel_viscos function calculates the viscosity of the flow gas relative to the viscosity of room air under STPD conditions. The concentrations of the flow gases and the flow gas temperature are used to determine the relative viscosity.

Common blocks used by the subroutine

None

Variables passed to the procedure

Fco2 - real.

Fh2o - real.

Fn2 - real.

Fo2 - real.

Temperature - real.

Variable definitions

Fh2o - a real variable equal to the fractional concentration of water vapor in the respiratory flow gas.

Fco2 - a real variable equal to the fractional concentration of CO_2 in the wet respiratory gas. This concentration value takes into account the amount of water vapor in the respiratory flow gas.

Fn2 - a real variable equal to the fractional concentration of N_2 in the wet respiratory gas. This concentration value takes into account the amount of water vapor in the respiratory flow gas.

Fo2 - a real variable equal to the fractional concentration of O_2 in the wet respiratory gas. This concentration value takes into account the amount of water vapor in the respiratory flow gas.

Rel_Viscosity - a real variable equal to the viscosity of the respiratory gas relative to the viscosity of room air under STPD conditions.

Temperature - a real variable equal to the temperature of the flow gas.

Vis_co2 - a real variable equal to the viscosity of pure CO_2 at the temperature of the flow gas.

Vis_h2o - a real variable equal to the viscosity of pure H_2O at the temperature of the flow gas.

Vis_n2 - a real variable equal to the viscosity of pure N_2 at the temperature of the flow gas.

Vis_o2 - a real variable equal to the viscosity of pure O_2 at the temperature of the flow gas.

A7.6.15 View Subroutine

The View subroutine defines the area of the plotting device that will contain a plot. View considers the plotting device to be an area of 100 by 100. The subroutine operates similar to the VIEWPORT command. However, using the View subroutine allows plotting areas to be defined independent of which axis is is physically longer.

Common blocks used by the subroutine

None

Variables passed to the subroutine

Bottom - real.

Left - real.

Right - real.

Top - real.

Variables returned by the subroutine

None

Variable definitions

Bottom - a real variable defining the bottom of the plotting area on the 100 by 100 plotting device.

Left - a real variable defining the left side the plotting area on the 100 by 100 plotting device.

Right - a real variable defining the right side of the plotting area on the 100 by 100 plotting device.

Top - a real variable defining the top of the plotting area on the 100 by 100 plotting device.

A7.6.16 ANALYSIS Program Listing

```
5     !*************************************************************************
10    !*************************************************************************
15    !
20    !       BREATH-BY-BREATH RESPIRATORY ANALYSIS/PLOTTING ROUTINE
25    !
30    !       HP BASIC FILENAME:  ANALYSIS
35    !
40    !       Department of Electrical and Computer Engineering,
45    !       Kansas State University
50    !
55    !       REVISION        DATE                    PROGRAMMER
60    !       --------        ----                    ----------
65    !         1.0           JUNE 1, 1984            LOREN E. RIBLETT
70    !         2.0           January 1, 1985         Michael Masters
75    !
80    !*************************************************************************
85    !
90    !       PURPOSE
95    !               THIS ROUTINE PERFORMS ALL ANALYSIS THAT IS CURRENTLY
100   !               DONE ON THE RESPIRATORY DATA.  RESULTS OF THIS ANALYSIS
105   !               ARE PRESENTED IN BOTH TABULAR AND GRAPHICAL FORMS.
110   !
115   !*************************************************************************
120   !
125   !     NOTE 1:   THIS ROUTINE ASSUMES THAT THE BINARY CALIBRATION FILES
130   !               CREATED BY "CAPCRUNCH" ARE STORED ON THE HP9895A 8"
135   !               FLEXIBLE DISK ":HP9895,700,0" (VOLUME #7 IN THE PASCAL
140   !               OPERATING SYSTEM).
145   !
150   !     NOTE 2:   ANALYSIS ASSUMES THAT THE FOUR BINARY DATA FILES CREATED
155   !               BY "DAPCRUNCH" ARE ALSO STORED ON THE HP9895A 8" FLEXIBLE
160   !               DISK ":HP9895,700,0" (VOLUME #7 IN THE PASCAL OPERATING
165   !               SYSTEM).
170   !
175   !     NOTE 3:   GAS MASS SPECTROMETER TIME DELAY VALUES CAN BE DETERMINED
180   !               ON A BREATH-BY-BREATH BASIS OR A FIXED TIME DELAY CAN BE
185   !               SELECTED FOR ANALYSIS OF THE ACQUIRED DATA.  SHOULD AN
190   !               EXTREME BREATH-BY-BREATH TIME BE CALCULATED, AN AVERAGE
195   !               TIME DELAY IS SUBSTITUTED FOR THE COMPUTED TIME DELAY..
200   !
205   !     NOTE 4:   RESPIRATORY VOLUMES CAN BE CORRECTED TO STPD/BTPS CON-
210   !               DITIONS PROVIDED THE BAROMETRIC PRESSURE (TORR), RELATIVE
215   !               HUMIDITY (%), AND BODY TEMPERATURE (DEG C) IS SUPPLIED.
220   !               USING THIS INFORMATION ALONG WITH POINT-BY-POINT TEMP-
225   !               ERATURE CORRECTION (USING CHANNEL 4, THE RESPIRATORY
230   !               TEMPERATURE CHANNEL), THE INSPIRATORY AND EXPIRATORY
235   !               GAS VOLUMES ARE CORRECTED.
240   !
245   !     NOTE 5:   ANY WINDOW OF DATA FROM THE DAM DATA MAY BE PLOTTED
250   !               EITHER ON THE HP9826 CRT OR HP9872C PLOTTER.  ALL FOUR
255   !               CHANNELS OF DATA ARE PLOTTED, THE CO2 AND O2 CHANNELS
260   !               BEING PLOTTED WITH A TIME DELAY EQUAL TO THE AVERAGE
265   !               TIME DELAY ENTERED BY THE USER.  DATA PLOTTED ON THE
270   !               HP9826 CRT MAY ALSO BE DUMPED TO THE HP2673A THERMAL
275   !               PRINTER.
280   !
285   !     NOTE 6:   ONCE BREATH-BY-BREATH ANALYSIS OF THE DAM BEGINS,
290   !               INFORMATION CONCERNING EACH BREATH IS PRINTED ON THE
295   !               DECwriter PRINTER.
300   !
305   !     NOTE 7:   ONCE THE DATA ARRAYS ARE EXHAUSTED SUMMARY DATA FOR THE
310   !               ENTIRE RUN IS COMPUTED AND PRINTED.  FILE NAMES AS WELL
```

```
!              AS CRITICAL CALIBRATION PARAMETERS ARE ALSO PRINTED.
!*************************************************************************
!
!                  Analysis enhancements for version 2.0
!
!       Note 1: The breath-by-breath results as analysis is proceding
!               may be omitted.  The breath-by-breath variables may be
!               plotted or printed post-analysis.
!
!       Note 2: The respiratory flow may be corrected for variations in
!               gas viscosity and temperature.
!
!       Note 3: The breath-by-breath variables of interest may be
!               displayed, printed or plotted, post-analysis.  The
!               individual breath values may be combined by one of two
!               methods.
!                   (1) Breath Averaging.  The individual breath values
!                       are averaged for a specified number of breaths
!                       (including averaging one breath).
!                   (2) Time Window averaging.  The breaths occurring in
!                       a specified window are averaged.
!
!       Note 4: The data arrays, calibration values are retained in a
!               COMMON area of memory.  This allows the data to stay
!               resident in memory as various respiratory analysis prog-
!               rams are executed.  It is also a very useful tool during
!               program debugging.
!
!       Note 5: Many of the functional portions of the program have been
!               transfered to true subroutines.  This enhances the read-
!               ability by building a structured format.
!
!*************************************************************************
!
!
!                    Declare the common area of memory.
!
!                               The respiratory data
 COM /Resp_data1/ INTEGER Line1(1:24000),Line2(1:24000)
 COM /Resp_data2/ INTEGER Line3(1:24000),Line4(1:24000)
 COM /Resp_data3/ INTEGER Cmin,Cmax,Qmin,Qmax,Pmin,Pmax,Tmin,Tmax
 COM /Resp_data4/ INTEGER No_points
 COM /Resp_data5/ C1$[10],O$[10],V$[10],T$[10]
!
!                               The calibration data
 COM /Cal_data1/ Cal$[10],Date$[25]
 COM /Cal_data2/ INTEGER Co2_dc_offset,O2_dc_offset,Bin_zero_flow,S
 COM /Cal_data3/ REAL Co2_cal,O2_cal
 COM /Cal_data4/ REAL Non_c_i_flowcal,Non_c_e_flowcal
 COM /Cal_data5/ REAL Corr_i_flowcal,Corr_e_flowcal
 COM /Cal_data6/ REAL Ta,Tb,Tc
 COM /Cal_data7/ REAL Time_delay,Ol,Pb,Rel_humid,Room_temp
!
!                               The data used by the Analysis routines.
 COM /Anal_data1/ INTEGER Data_available
 COM /Anal_data2/ INTEGER True,False
 COM /Anal_data3/ INTEGER Time_of_brth,O2_cons,Co2_prod,Insp_min_vent
 COM /Anal_data4/ INTEGER Expr_min_vent,V_tid_insp,V_tid_expr,Resp_freq
 COM /Anal_data5/ INTEGER Breath_good,Resp_quotient,Formfd$[1]
 COM /Anal_data5a/ INTEGER Frc_o2_cons,Frc_co2_prod,Volume_of_lung
 COM /Anal_data6/ INTEGER Corr_flag,REAL Room_fh2o,T
 COM /Anal_data7/ Vap(0:250),Fh2o_sat(0:250)
 COM /Anal_data8/ Resp_var(0:12,1:500)
 COM /Anal_data9/ Name$[25],Comment$[46]
!
```

```
640   !*** SPECIAL FUNCTION KEY DECLARATION
645   !
650 Define_keys:!
655    ON KEY 2 LABEL "ANALYSIS" GOSUB Anal_routine
660    ON KEY 9 LABEL "  EXIT" GOTO Done
665    GOTO Define_keys
670   !
675   !*** SIGNAL END OF ROUTINE AND RETURN TO AUTOST
680   !
685 Done:OFF KEY
690    DISP "PROGRAM RUN COMPLETE"
695    MASS STORAGE IS ":INTERNAL"
700    LOAD "AUTOST",1
705   !
710   !
715 Anal_routine: !
720 !
725 !*** BEGINNING OF ANALYSIS SUBROUTINE
730    OFF KEY                    ! Turn the SOFT KEYS off.
735    INTEGER Good_insp_count,Good_exp_count,Insp_begin,Expr_begin
740    INTEGER A,Z,Breath_count,Final_insp
745    INTEGER Found,B_by_b_output                    ! Flags
750   !
755   !
760   !                  Declare all the constant values and variable pointers
765   !
770    True=(1=1)                 ! Corresponds to a logical True value.
775    False=(1=0)                ! Logical False value.
780    Formfd$=CHR$(12)           ! The ASCII character for a form feed.
785   !
790   !                    Declare the value pointers of the respiriratory
795   !                    variable array.
800    Time_of_brth=0
805    O2_cons=Time_of_brth+1
810    Co2_prod=O2_cons+1
815    Frc_o2_cons=Co2_prod+1
820    Frc_co2_prod=Frc_o2_cons+1
825    Insp_min_vent=Frc_co2_prod+1
830    Expr_min_vent=Insp_min_vent+1
835    V_tid_insp=Expr_min_vent+1
840    V_tid_expr=V_tid_insp+1
845    Volume_of_lung=V_tid_expr+1
850    Resp_freq=Volume_of_lung+1
855    Resp_quotient=Resp_freq+1
860    Breath_good=Resp_quotient+1
865    !
870    !                    *** GET SUBJECT INFORMATION FOR PRINTOUTS
875    BEEP
880    INPUT "ENTER THE SUBJECT'S NAME OR IDENTIFIER",Name$
885   !
890   !                              If data is resident in memory, inform the
895   !                              user.
900    IF Data_available THEN
905       PRINT Formfd$
910       PRINT "Data is resident in memory."
915       PRINT "      data file names     Max     Min"
920       PRINT "       CO2:   "&C1$&"     "&VAL$(Cmax)&"    "&VAL$(Cmin)
925       PRINT "       O2:    "&O$&"      "&VAL$(Omax)&"    "&VAL$(Omin)
930       PRINT "       FLOW:  "&V$&"      "&VAL$(Fmax)&"    "&VAL$(Fmin)
935       PRINT "       TEMP:  "&T$&"      "&VAL$(Tmax)&"    "&VAL$(Tmin)
940       PRINT "There are "&VAL$(No_points)&" data points"
945       PRINT "Calibration values"
950       IF Cal$="" THEN
955          PRINT "   No calibration data is resident in memory"
960       ELSE
```

```
965         PRINT "   The CALIBRATION file name is "&Cal$
970       END IF
975       PRINT "   Offsets:      CO2: ";Co2_dc_offset
980       PRINT "                  O2: ";O2_dc_offset
985       PRINT "   Binary zero flow: ";Bin_zero_flow
990       PRINT "   CO2 cal: ";Co2_cal;"    O2 cal: ";O2_cal
995       PRINT "   Non Corrected flow cals"
1000      PRINT "        Insp. ";Non_c_i_flowcal;"   Expr. ";Non_c_e_flowcal
1005      PRINT "   Corrected flow cals"
1010      PRINT "        Insp. ";Corr_i_flowcal;"   Expr. ";Corr_e_flowcal
1015 !
1020 !                          See if the user wishes to keep the present
1025 !                          calibration data.  If not, enter the new data.
1030      Q$="N"                !Set the default to No (get new data)
1035      BEEP
1040      INPUT "DO YOU WISH TO USE THESE CALIBRATION DATA? (Y/N)",Q$
1045      IF Q$="N" OR Q$="n" THEN
1050         GOSUB Rtcal
1055         PRINT "New cal data loaded. B_by_B data is the same"
1060      END IF
1065 !
1070 !                          See if the user wishes to keep the present
1075 !                          respiratory data.  If not, enter the new data.
1080      Q$="N"                !Set the default to No (get new data)
1085      BEEP
1090      INPUT "DO YOU WISH TO USE THESE B_by_B DATA? (Y/N)",Q$
1095      IF Q$="N" OR Q$="n" THEN
1100         BEEP
1105         INPUT "ENTER THE TOTAL NUMBER OF POINTS TO BE ANALYZED",No_points
1110         GOSUB Rtdata
1115      END IF
1120      PRINT Formfd$         !Clear the screen
1125   !
1130   ELSE
1135                  !  No data is resident.  Read in the data.
1140   !
1145   !                 *** LOAD BINARY CALIBRATION FILE IF REQUESTED
1150      BEEP
1155      Q$="N"          ! Set the default to No.
1160      INPUT "LOAD CALIBRATION FACTORS FROM DISK ? (Y/N)",Q$
1165      IF Q$="Y" OR Q$="y" THEN GOSUB Rtcal
1170      BEEP
1175      INPUT "ENTER THE TOTAL NUMBER OF POINTS TO BE ANALYZED",No_points
1180   !
1185   !                 *** LOAD FOUR CHANNELS OF BINARY DATA
1190      GOSUB Rtdata
1195      Data_available=True         ! Set the data resident in memory flag.
1200   END IF
1205   BEEP
1210   !
1215   !               Set experimental time which data collection began.
1220   INPUT "Enter the begining time of data collection. (sec)",Begtime_colct
1225   !
1230 Analyze:!
1235   !        Enter a comment to be printed on the hard copy output.
1240   BEEP
1245   INPUT "ENTER A COMNT TO BE PRNTED ON THE HRD CPY OUTPUT.",Comment$
1250   !              *** CHOOSE APPROPRIATE SAMPLING FREQUENCY
1255   S=50
1260   Q$="N"            ! Set the default to No.
1265   BEEP
1270   INPUT "CHANGE SAMPLING FREQUENCY FROM 50 HZ.? (Y/N)",Q$
1275   IF Q$="Y" OR Q$="y" THEN
1280      INPUT "ENTER DESIRED SAMPLING FREQUENCY (HZ.)",S
1285   END IF
```

```
1290  Q$=""
1295  !
1300  !*** CHOOSE B-BY-B TIME DELAYS OR A FIXED GMS TIME DELAY
1305  !
1310  BEEP
1315  Q$="F"            !Set the default to fixed.
1320  INPUT "B-BY-B TIME DELAY OR FIXED TIME DELAY (B/F) ?",Q$
1325  IF Q$="B" OR Q$="b" THEN
1330  !
1335  !*** FOR B-BY-B DELAYS, CHOOSE STARTING VALUE FOR AVERAGE DELAY
1340  !
1345     BEEP
1350     INPUT "AVERAGE TIME DELAY FOR BAD BREATH PROBLEMS (msec)?",Time_dela
y_sum
1355     Avg_time_delay=Time_delay_sum
1360     Time_delay_flag=1      !Time_delay_flag=1 FOR B-BY-B TIME DELAYS
1365  ELSE
1370  !
1375  !*** FOR FIXED TIME DELAY, ENTER DESIRED TIME DELAY
1380  !
1385     Time_delay_flag=0           !Time_delay_flag=0 FOR FIXED TIME DELAYS
1390     PRINT "CURRENT TIME DELAY IS ";Time_delay;" msec"
1395     Q$=""
1400     BEEP
1405     INPUT "CHANGE TIME DELAY? (Y/N)",Q$
1410     IF Q$="Y" OR Q$="y" THEN
1415        INPUT "ENTER DESIRED TIME DELAY (msec.)",Time_delay
1420     END IF
1425  END IF
1430  !
1435  !        See if viscosity and temperature corrections are desired.
1440  BEEP
1445  Q$=""
1450  INPUT "USE TEMP. AND VISCOSITY CORRECTIONS? (Y/N)",Q$
1455  IF Q$="Y" OR Q$="y" THEN
1460  !
1465  !*** IF STPD/BTPS DESIRED, LOAD IN WATER VAPOR PRESSURE TABLE
1470  !           Bypass if the vapor pressures are already present.
1475     IF Vap(0)=0 THEN
1480        DISP "READING WATER VAPOR PRESSURES"
1485        ASSIGN @Vap_path TO "VAP_BASIC:HP9895,702,3"
1490        ON END @Vap_path GOTO 1520
1495        ENTER @Vap_path;Vap(*)
1500        ASSIGN @Vap_path TO *
1505     END IF
1510     Corr_flag=True        ! Set the temperatur/viscosity correction flag.
1515     IF Body_temp=0 THEN Body_temp=(98.6-32)*5/9
1520     Body_temp=Body_temp*9/5+32        ! Convert body temp to deg F.
1525     BEEP
1530     INPUT "ENTER THE BODY TEMPERATURE (deg F)",Body_temp
1535     Body_temp=(Body_temp-32)*5/9       ! Convert body temp to deg C.
1540  !                 Display the ambient conditions.
1545     IF Room_temp=0 THEN Room_temp=(70-32)*5/9
1550     PRINT Formfd$,TABXY(1,15);"      Barometric pressure = ";Pb/25.4
1555     PRINT "      Relative humidity   = ";Rel_humid*100
1560     PRINT USING "K,DDD.DD";"      Room temperature = ";Room_temp*9/5+32
1565     BEEP
1570     Q$="N"
1575     INPUT "Do you wish to use the ambient cond. above? (Y/N)",Q$
1580     IF NOT (Q$="y" OR Q$="Y") THEN
1585        Pb=Pb/25.4         ! Convert to inches Hg.
1590        BEEP
1595        INPUT "Enter the new barometric pressure. (inches)",Pb
1600        Pb=Pb*25.4         ! Convert to torr.
1605        Rel_humid=Rel_humid*100
```

```
1610        BEEP
1615        INPUT "Enter the new percent relative humidty",Rel_humid
1620        Rel_humid=Rel_humid/100
1625        Room_temp=Room_temp*9/5+32        ! Convert to deg F
1630        BEEP
1635        INPUT "Enter the new room temperature",Room_temp
1640        Room_temp=(Room_temp-32)*5/9       ! Convert to deg C.
1645     END IF
1650     PRINT Formfd$
1655  !
1660  !              Generate the Fractional water vapor concentration of the
1665  !              saturated gas.
1670     FOR I=0 TO 250
1675        Fh2o_sat(I)=Vap(I)/Pb
1680     NEXT I
1685  !
1690  !
1695  !           Get the water fraction under room conditions and the vapor
1700  !           pressure at body temperature, calculate STPD to BTPS
1705  !           conversion factor and use the temperature corrected flow
1710  !           calibration factors.
1715     Room_fh2o=Rel_humid*Fh2o_sat((Room_temp-20)*10)
1720     Ph2o_body=Vap((Body_temp-20)*10)
1725     Stpd_to_btps=(273.15+Body_temp)/273.15*760/(Pb-Ph2o_body)
1730     Btps_to_stpd=1/Stpd_to_btps
1735     Insp_flowcal=Corr_i_flowcal
1740     Expr_flowcal=Corr_e_flowcal
1745  ELSE
1750  !              Set the correction flag to False.  Set conversion factors
1755  !              to unity.  Set the flow calibration factors equal to the
1760  !              non-corrected calibration factors.
1765     Corr_flag=False
1770     Stpd_to_btps=1
1775     Btps_to_stpd=1
1780     Insp_flowcal=Non_c_i_flowcal
1785     Expr_flowcal=Non_c_e_flowcal
1790  END IF
1795  PRINT Formfd$        ! Clear the screen.
1800  !
1805  !   See if FRC calculations are desired. (FRC calculations have not been
1810  !   fine tuned in this program.  See Rick Pieschl.)
1815  BEEP
1820  Q$=""
1825  INPUT "Do you wish to use FRC corrections? (Y/N)",Q$
1830  IF Q$="Y" OR Q$="y" THEN
1835     Frc_flag=True
1840     IF NOT (Corr_flag) THEN    !Body temp. and Pb has not been entered.
1845        BEEP
1850        INPUT "Enter the body temperature (deg F)",Body_temp
1855        Body_temp=(Body_temp-32)*5/9
1860        Ph2o_body=Vap((Body_temp-20)*10)
1865        BEEP
1870        INPUT "Enter the barometric pressure(inches)",Pb
1875        Pb=Pb*25.4
1880     END IF
1885     Frac_h2o_body=Ph2o_body/Pb
1890  ELSE
1895     Frc_flag=False
1900  END IF
1905  !
1910  !*** SEE IF PLOT OF DATA IS DESIRED
1915  !
1920  Q$="N"
1925  BEEP
1930  INPUT "WOULD YOU LIKE A PLOT OF THE DATA?(Y/N)",Q$
```

```
1935 !
1940 !*** IF SO, GO PLOT THE DATA
1945 !
1950  IF Q$="Y" OR Q$="y" THEN GOSUB Dtplot
1955 !
1960 !
1965 !                    Get the first and last data points to be analyzed.
1970  BEEP
1975  INPUT "ENTER STARTING POINT TO ANALYZE.",Start
1980  IF Start>No_points-50 THEN GOTO 1970
1985  A=Start+1
1990  Z=Start+1
1995  BEEP
2000  INPUT "ENTER ENDING POINT TO ANALYZE.",End
2005  IF End>No_points OR End<Start THEN GOTO 1995
2010 !
2015 !          See if the breath-by-breath values are to be printed during
2020 !          analysis.
2025  PRINT TABXY(1,18);"Do you wish to omit the breath by breath output of"
2030  PRINT "the results as the data is being analyzed? (Y/N)"
2035  Q$="N"
2040  BEEP
2045  INPUT Q$
2050  PRINT Formfd$
2055  IF Q$="N" OR Q$="n" THEN
2060     B_by_b_output=True
2065  ELSE
2070     B_by_b_output=False
2075  END IF
2080 !
2085 !    *** INITIALIZE NECESSARY ANALYSIS PARAM.
2090  IF OI=0 THEN OI=.11
2095  T=1/S
2100  Num_breaths=0
2105  Good_insp_count=0
2110  Good_exp_count=0
2115  Time_delay_cnt=1
2120  Tot_vol_insp=0
2125  Tot_vol_exp=0
2130  Tot_o2_insp=0
2135  Tot_co2_insp=0
2140  Tot_o2_exp=0
2145  Tot_co2_exp=0
2150  Tot_o2_cons=0
2155  Tot_co2_prod=0
2160  Tot_time_insp=0
2165  Tot_time_expr=0
2170  Lung_vol=.4
2175  Net_lung_chng=0
2180 !
2185 !         *** LOCATE FIRST INSPIRATION IN FLOW SIGNAL
2190 !         *** The first breath begins at point A when the (A-1)'th point
2195 !             is an expiratory flow and the A'th point is an inspiration.
2200  REPEAT
2205     IF Line3(A-1)-Bin_zero_flow>0 AND FNInspiration((A)) THEN
2210        Found=True
2215     ELSE
2220        Found=False
2225        A=A+1
2230     END IF
2235  UNTIL Found
2240  Init_index=A
2245 !
2250 !
2255 !                    *** GO PRINT DATA TABLE HEADER ON PRINTER LISTING
```

```
2260  GOSUB Hard_copy_head
2265  Breath_count=0
2270  Still_calc=True
2275   !
2280   !
2285   !*************************************************************************
2290   !                    Compute the breath by breath values.  Repeat the
2295   !                    computations until the stopping point has been
2300   !                    reached.
2305  REPEAT
2310     DISP "Calc. the results of breath starting at",A
2315     Insp_count=A
2320     Breath_count=Breath_count+1
2325     Resp_var(Time_of_brth,Breath_count)=A*T+Begtime_colct
2330  !
2335  !                  Determine the GMS time delay.
2340     IF Time_delay_flag<>0 THEN       ! Use the B-by-B time delay.
2345        GOSUB Bbb_time_delay
2350  !
2355  !          IF B-BY-B TIME DELAY OUTSIDE LIMITS, SUBSTITUTE AVERAGE DELAY
2360        IF (Time_delay>560) OR (Time_delay<330) THEN
2365  !
2370  !                                    *** The time delay was bad. ***
2375           Time_delay=Avg_time_delay
2380           Time_delay_flag=2
2385        ELSE
2390  !
2395  !             *** The time delay was good so update the running average
2400           Time_delay_flag=1
2405           Time_delay_sum=Time_delay_sum+Time_delay
2410           Time_delay_cnt=Time_delay_cnt+1
2415           Avg_time_delay=Time_delay_sum/Time_delay_cnt
2420        END IF
2425     END IF
2430  !
2435  !       *** ADJUST CO2 AND O2 INDEX (Z) FOR PROPER POINT SELECTION
2440     Z=A+INT(Time_delay/1000*S+.5)
2445  !
2450  !         *** COMPUTE HALF THE AREA FOR THE FIRST TRAPEZOID
2455     Calc_sig_val((A),(Z),("INSP"),Incr_vol,Fo2,Fco2)
2460     Air_insp=.5*Incr_vol
2465     Co2_insp=.5*Fco2*Incr_vol
2470     O2_insp=.5*Fo2*Incr_vol
2475  !
2480  !         *** Sum up the inspiratory volumes.
2485     A=A+1
2490     Z=Z+1
2495     REPEAT
2500        IF Z>End-50 THEN           ! Exit if to close to the end.
2505           Still_calc=False
2510           GOTO Done_calc
2515        END IF
2520   !
2525        Calc_sig_val((A),(Z),("INSP"),Incr_vol,Fo2,Fco2)
2530        Air_insp=Air_insp+Incr_vol
2535        Co2_insp=Co2_insp+Fco2*Incr_vol
2540        O2_insp=O2_insp+Fo2*Incr_vol
2545  !
2550        A=A+1
2555        Z=Z+1
2560  !                                      *** LOOP UNTIL END OF INSPIRATION
2565     UNTIL NOT (FNInspiration(A))
2570     Insp_time=(A-Insp_count)*T          !Time of inspiration in seconds
2575  !
2580  !         *** PUT 1/2 OF THE LAST TRAPEZOIDAL AREA CORRESPONDING TO THE
```

```
!       INSPIRATORY/EXPIRATORY TRANSITION ON THE INSPIRATION AND 1/2
!       OF THE AREA ON THE EXPIRATION
   Calc_sig_val((A),(Z),("INSP"),Incr_vol,Fo2,Fco2)
   Air_insp=Air_insp+.5*Incr_vol
   Co2_insp=Co2_insp+.5*Fco2*Incr_vol
   O2_insp=O2_insp+.5*Fo2*Incr_vol
!
   Calc_sig_val((A),(Z),("EXPR"),Incr_vol,Fo2,Fco2)
   Air_expr=.5*Incr_vol
   Co2_expr=.5*Fco2*Incr_vol
   O2_expr=.5*Fo2*Incr_vol
   Expr_begin=A
!
!
!          Determine the expiratory tidal volume and expired CO2 and
!          O2 volumes.
!
   A=A+1
   Z=Z+1
   REPEAT
      IF Z>End-50 THEN          ! Exit if to close to the end.
         Still_calc=False
         GOTO Done_calc
      END IF
!
      Calc_sig_val((A),(Z),("EXPR"),Incr_vol,Fo2,Fco2)
      Air_expr=Air_expr+Incr_vol
      Co2_expr=Co2_expr+Fco2*Incr_vol
      O2_expr=O2_expr+Fo2*Incr_vol
!
      A=A+1
      Z=Z+1
!                    *** LOOP UNTIL END OF EXPIRATION
   UNTIL NOT (FNExpiration(A))
!
!
!       *** INCLUDE HALF OF THE TRAPEZOIDAL AREA CORRESPONDING TO THE
!       EXPIRATORY/INSPIRATORY TRANSITION.
!
   Calc_sig_val((A),(Z),("EXPR"),Incr_vol,Fo2,Fco2)
   Air_expr=Air_expr+.5*Incr_vol
   Co2_expr=Co2_expr+.5*Fco2*Incr_vol
   O2_expr=O2_expr+.5*Fo2*Incr_vol
!
!                          ******************
!       *** BEGIN CALCULATIONS FOR THIS PARTICULAR BREATH ***
!
                           !INSPIRATORY VOLUME FOR THIS BREATH (LITERS)
   Air_insp=Air_insp*Insp_flowcal*Stpd_to_btps
!
   Co2_insp=Co2_insp*Insp_flowcal      !CO2 INSPIRED FOR THIS BREATH (L)
   O2_insp=O2_insp*Insp_flowcal        !O2 INSPIRED FOR THIS BREATH (L)
!
                          !EXPIRATORY VOLUME FOR THIS BREATH (LITERS)
   Air_expr=Air_expr*Expr_flowcal*Stpd_to_btps
!
   Co2_expr=Co2_expr*Expr_flowcal   !CO2 EXPIRED FOR THIS BREATH (L)
   O2_expr=O2_expr*Expr_flowcal     !O2 EXPIRED FOR THIS BREATH (L)
   O2cons=O2_insp+O2_expr           !CONSUMED O2 FOR THIS BREATH (L)
   Co2prod=Co2_insp+Co2_expr        !CO2 PRODUCED FOR THIS BREATH (L)
   Num_breaths=Num_breaths+1        !TOTAL NUMBER OF BREATHS ANALYZED
!
!
!            Start FRC corrections.  These corrections are not
!            implemented in version 2.0.  See Rick Pieschl.
```

```
2910      IF Frc_flag=True THEN
2915 !                  *** COMPUTE THE END EXPIRATE VALUES AND THEN COMPUTE
2920 !                      THE ALVEOLAR VALUES, THE CHANGE IN LUNG VOLUME,
2925 !                      AND THE LUNG VOLUME.
2930        Max_pnt=FNMax_co2_pntr(Line1(*),A)
2935   !     Compute the dry aveolar gas concentrations and then convert them
2940   !      to wet values.
2945        Frac_co2_1=(Line1(Max_pnt)-Co2_dc_offset)*Co2_cal
2950        Frac_o2_1=(Line2(Max_pnt)-O2_dc_offset)*O2_cal+O1
2955        Frac_co2_1=Frac_co2_1*(1-Frac_h2o_body)
2960        Frac_o2_1=Frac_o2_1*(1-Frac_h2o_body)
2965        IF Num_breaths>1 THEN
2970           Chng_fo2=Frac_o2_1-Frac_o2_2
2975           Chng_fco2=Frac_co2_1-Frac_co2_2
2980  !
2985  !              Compute the change in lung volume.
2990           Chng_lung_vol=-Air_insp/(1-Frac_h2o_body)-Air_expr+(O2cons+Co2
prod)*Stpd_to_btps
2995           Chng_lung_vol=Chng_lung_vol+Lung_vol*(Chng_fo2+Chng_fco2)
3000           Chng_lung_vol=Chng_lung_vol/(1-Frac_o2_1-Frac_co2_1)
3005  !
3010           Frc_o2cons=O2cons+(Chng_lung_vol*Frac_o2_1+Chng_fo2*Lung_vol)*
Btps_to_stpd
3015           Frc_co2prod=Co2prod+(Chng_lung_vol*Frac_co2_1+Chng_fco2*Lung_v
ol)*Btps_to_stpd
3020           Lung_vol=Lung_vol+Chng_lung_vol
3025        END IF
3030        Frac_o2_2=Frac_o2_1
3035        Frac_co2_2=Frac_co2_1
3040     ELSE
3045        Frc_o2cons=O2cons
3050        Frc_co2prod=Co2prod
3055     END IF
3060  !
3065  !
3070     Expr_time=(A-Expr_begin)*T     !TIME FOR CURRENT EXPIRATION (SECONDS)
3075     Final_index=A
3080     Final_insp=Insp_count
3085     Tot_time=(Expr_time+Insp_time)/60
3090  !
3095  !                    Compute and store the breath-by-breath values
3100  !                    for this breath.
3105     Resp_var(O2_cons,Breath_count)=-O2cons/Tot_time
3110     Resp_var(Co2_prod,Breath_count)=Co2prod/Tot_time
3115     Resp_var(Insp_min_vent,Breath_count)=-Air_insp/Tot_time
3120     Resp_var(Expr_min_vent,Breath_count)=Air_expr/Tot_time
3125     Resp_var(V_tid_insp,Breath_count)=-Air_insp
3130     Resp_var(V_tid_expr,Breath_count)=Air_expr
3135     Resp_var(Resp_freq,Breath_count)=1/(Insp_time+Expr_time)*60
3140     Resp_var(Resp_quotient,Breath_count)=ABS(Co2prod/O2cons)
3145     Resp_var(Frc_o2_cons,Breath_count)=-Frc_o2cons/Tot_time
3150     Resp_var(Frc_co2_prod,Breath_count)=Frc_co2prod/Tot_time
3155.    Resp_var(Volume_of_lung,Breath_count)=Lung_vol
3160  !
3165     IF B_by_b_output THEN
3170  !
3175  !          *** GO PRINT CALCULATED VALUES FOR THIS BREATH
3180        GOSUB Hard_output
3185     END IF
3190  !
3195  !*** KEEP TRACK OF GOOD INSPIRATIONS AND EXPIRATIONS
3200     IF ABS(Air_insp)>.4 AND ABS(Air_expr)>.4 THEN
3205        Resp_var(Breath_good,Breath_count)=True
3210     ELSE
3215        Resp_var(Breath_good,Breath_count)=False
```

```
3220      END IF
3225   !
3230      IF ABS(Air_insp)>.4 THEN
3235         Good_insp_count=Good_insp_count+1
3240      END IF
3245      IF ABS(Air_expr)>.4 THEN
3250         Good_exp_count=Good_exp_count+1
3255      END IF
3260   !
3265   !              *** ADJUST RUNNING TOTAL VALUES FOR ENTIRE TRIAL
3270      Tot_vol_insp=Tot_vol_insp+Air_insp
3275      Tot_vol_exp=Tot_vol_exp+Air_expr
3280      Tot_o2_insp=Tot_o2_insp+O2_insp
3285      Tot_o2_exp=Tot_o2_exp+O2_expr
3290      Tot_co2_insp=Tot_co2_insp+Co2_insp
3295      Tot_co2_exp=Tot_co2_exp+Co2_expr
3300      Tot_o2_cons=Tot_o2_cons+O2cons
3305      Tot_co2_prod=Tot_co2_prod+Co2prod
3310      Tot_time_insp=Tot_time_insp+Insp_time
3315      Tot_time_expr=Tot_time_expr+Expr_time
3320   !
3325      R=-Tot_co2_prod/Tot_o2_cons
3330   Done_calc:   !     Keep repeating until the end has been reached.
3335    UNTIL Still_calc=False
3340    Breath_count=Breath_count-1
3345    DISP
3350    !
3355    !*************************************************************************
3360    !
3365    !                        COMPUTE FINAL TRIAL AVERAGES
3370    !
3375    Tot_time_resp=(Final_index-Init_index)*T !TOTAL RESPIRATORY TIME (SEC)
3380    Minvoli=Tot_vol_insp*60/Tot_time_resp     !INSP. MINUTE VOLUME (L/MIN)
3385    Minvole=Tot_vol_exp*60/Tot_time_resp      !EXP. MINUTE VOLUME (L/MIN)
3390    Num_breaths=(Good_insp_count+Good_exp_count)/2
3395    Avg_air_insp=Tot_vol_insp/Num_breaths  !AVERAGE INSPIRATORY VOLUME (L)
3400    Avg_air_expr=Tot_vol_exp/Num_breaths   !AVERAGE EXPIRATORY VOLUME (L)
3405    Respf=Num_breaths*60/Tot_time_resp     !RESP. FREQUENCY (BREATHS/MIN)
3410    Avg_o2_insp=Tot_o2_insp/Num_breaths    !AVERAGE INSPIRED O2/BREATH (L)
3415    Avg_o2_expr=Tot_o2_exp/Num_breaths     !AVERAGE EXPIRED O2/BREATH (L)
3420    Avg_co2_insp=Tot_co2_insp/Num_breaths  !AVERAGE INSPIRED CO2/BREATH (L)
3425    Avg_co2_expr=Tot_co2_exp/Num_breaths   !AVERAGE EXPIRED CO2/BREATH (L)
3430    Avo2cons=Tot_o2_cons/Num_breaths       !AVERAGE O2 CONSUMED/BREATH (L)
3435    Avco2prod=Tot_co2_prod/Num_breaths
3440    V_dot_o2=Tot_o2_cons/Tot_time_resp*60
3445    V_dot_co2=Tot_co2_prod/Tot_time_resp*60
3450    R=ABS(V_dot_co2/V_dot_o2)
3455    !
3460    !*** GO PRINT MEAN VALUES
3465    !
3470    PRINTER IS 9
3475    GOSUB Means
3480    PRINTER IS 1
3485    !
3490    Q$=""
3495    BEEP
3500    INPUT "DISPLAY THE BREATH-BY-BREATH DATA? (Y/N)",Q$
3505    IF Q$="Y" OR Q$="y" THEN
3510       CALL Disp_results(Breath_count,Corr_flag)
3515    END IF
3520    !
3525    BEEP
3530    Q$=""
3535    INPUT "REDO ANALYSIS? (Y/N)",Q$
3540    IF Q$="Y" OR Q$="y" THEN GOTO Analyze
```

```
3545  !
3550  !*** END OF ANALYSIS ROUTINE
3555  !
3560  RETURN              ! Exit the Analysis portion.
3565  !
3570  !*** SUBROUTINE TO PRINT STANDARD HEADER TO THE DECwriter II PRINTER
3575  !
3580 Hard_copy_head:PRINTER IS 9
3585  A$="Air"
3590  B$="O2"
3595  C$="CO2"
3600  D$="Inspired"
3605  E$="Expired"
3610  F$="(liters)"
3615  G$="BTPS"
3620  H$="STPD"
3625  PRINT USING "2/,K";"SUBJECT IDENTIFIER:   "&Name$
3630  PRINT USING "2/,K";"DATE:         "&Date$
3635  PRINT USING "/,K,2/";"COMMENT:     "&Comment$
3640  IF B_by_b_output THEN
3645     PRINT USING "#,2X,K,5X,K,7X,K,8X,K,8X,K,7X,K";"Breath";A$;A$;B$;B$;C
$
3650     PRINT USING "#,7X,K,8X,K,7X,K,7X,K,6X,K,5X,K";C$;B$;C$;"Insp";"Expr".
;"Delay"
3655     PRINT
3660     PRINT USING "#,2X,K,4X,K,2X,K,3X,K,2X,K";"Start";D$;E$;D$;E$
3665     PRINT USING "#,3X,K,2X,K,3X,K,2X,K";D$;E$;"Consumed";"Produced"
3670     PRINT USING "#,4X,K,6X,K,6X,K";"TIME";"TIME";"TIME"
3675     PRINT
3680     PRINT USING "#,2X,K,4X,K,2X,K,2X,K,2X,K,2X,K";"Index";F$;F$;F$;F$;F$
3685     PRINT USING "#,2X,K,2X,K,2X,K,3X,K,5X,K,5X,K";F$;F$;F$;"(sec)";"(sec
)";"(msec)"
3690     PRINT
3695     IF Corr_flag THEN
3700        PRINT USING "#,13X,K,6X,K,6X,K,6X,K,6X,K";G$;G$;H$;H$;H$
3705        PRINT USING "#,6X,K,6X,K,6X,K";H$;H$;H$
3710        PRINT
3715     END IF
3720     PRINT USING "120(""-"")"
3725  END IF
3730  PRINTER IS 1
3735  RETURN
3740  !
3745  !*** SUBROUTINE TO PRINT A SINGLE LINE OF BREATH-BY-BREATH INFORMATION
3750  !
3755 Hard_output: !
3760  PRINTER IS 9
3765 !PRINT USING "#,70X,K,18X,K";"|";"|"
3770  PRINT
3775  PRINT USING "#,2X,DDDDD";Insp_count
3780  PRINT USING "#,5X,DD.DDD,3(4X,DD.DDD)";Air_insp;Air_expr;O2_insp;O2_expr
3785  PRINT USING "#,4X,D.DDD,5X,D.DDD,3X,K,1X,D.DDD";Co2_insp;Co2_expr;"|";O
2cons
3790  PRINT USING "#,5X,D.DDD,2X,K";Co2prod;"|"
3795  PRINT USING "#,3X,D.DD,6X,D.DD";Insp_time;Expr_time
3800  IF Time_delay_flag<>2 THEN PRINT USING "#,6X,DDD";Time_delay
3805  IF Time_delay_flag=2 THEN PRINT USING "#,4X,K,DDD,K";"**";Time_delay;"*
*"
3810  PRINT
3815  PRINTER IS 1
3820  RETURN
3825  !
3830  !*** SUBROUTINE TO PRINT TRIAL AVERAGE VALUE INFORMATION
3835  !
3840 Means:  !
```

```
3845  PRINT
3850  PRINT
3855  PRINT "       Summary data and average values"
3860  PRINT
3865  PRINT "                                  Data Point   Time (sec)"
3870 An_pnts:IMAGE K,5D,K,3D.DD
3875  Temporary=(Start-1)*T+Begtime_colct
3880  PRINT USING An_pnts;"Beginning analysis point: ",Start,"          ",Temp
orary
3885  Temporary=(Init_index-1)*T+Begtime_colct
3890  PRINT USING An_pnts;"The first inspiration:    ",Init_index,"          "
,Temporary
3895  Temporary=(Final_insp-1)*T+Begtime_colct
3900  PRINT USING An_pnts;"The last inspiration:     ",Final_insp,"          "
,Temporary
3905  Temporary=(End-1)*T+Begtime_colct
3910  PRINT USING An_pnts;"Ending analysis point:    ",End,"          ",Tempor
ary
3915  PRINT
3920  PRINT
3925  IF NOT (Corr_flag) THEN
3930     H$=""
3935     G$=""
3940  END IF
3945  PRINT USING "2/,#,K,4D.D";"Inspiratory minute volume = ";Minvoli
3950  PRINT USING "K,4A";" liters per minute ",G$
3955  PRINT USING "/,#,K,3D.D";"Expiratory minute volume = ";Minvole
3960  PRINT USING "K,4A";" liters per minute ",G$
3965  PRINT USING "/,#,K,3D.3D";"O2 consumed per minute = ";V_dot_o2
3970  PRINT USING "K,4A";" liters per minute ",H$
3975  PRINT USING "/,#,K,3D.3D";"CO2 produced per minute =";V_dot_co2
3980  PRINT USING "K,4A";" liters per minute ",H$
3985  PRINT USING "/,K,DDD.3D";"RESPIRATORY QUOTIENT = ";R
3990  PRINT USING "2/,K,DDD.4D,K,4A";"Inspiratory tidal volume = ";Avg_air_in
sp;" liters ",G$
3995  PRINT USING "/,K,DDD.4D,K,4A";"Expiratory tidal volume = ";Avg_air_expr
;" liters ",G$
4000  PRINT USING "/,K,DDDD.D,K";"Respiratory frequency = ";Respf;" breaths p
er minute"
4005  PRINT USING "/,K,DDD.3D,K,4A";"Mean O2 inspired = ";Avg_o2_insp;" liter
s ",H$
4010  PRINT USING "/,K,DDD.3D,K,4A";"Mean O2 expired = ";Avg_o2_expr;" liters
 ",H$
4015  PRINT USING "/,K,DDD.3D,K,4A";"Mean CO2 inspired = ";Avg_co2_insp;" lit
ers ",H$
4020  PRINT USING "/,K,DDD.3D,K,4A";"Mean CO2 expired = ";Avg_co2_expr;" lite
rs ",H$
4025  PRINT USING "/,K,DDD.3D,K,4A";"Mean O2 consumed per breath = ";Avo2cons
;" liters ",H$
4030  PRINT USING "/,K,DDD.3D,K,4A";"Mean CO2 produced per breath =";Avco2pro
d;" liters ",H$
4035  PRINT USING "/,K,3D.D,K";"Total time of inspiration = ";Tot_time_insp;"
 sec"
4040  PRINT USING "/,K,3D.D,K";"Total time of expiration = ";Tot_time_expr;"
sec"
4045  PRINT USING "/,K,3D.D,K";"Total time of respiration = ";Tot_time_resp;"
 sec"
4050  PRINT USING "/,K,3D.D";"Number of good inspirations = ";Good_insp_count
4055  PRINT USING "/,K,3D.D";"Number of good expirations = ";Good_exp_count
4060  PRINT USING "/,K,3D.D";"Number of good breaths = ";Num_breaths
4065  PRINT USING "3/"
4070  PRINT "       Calibration data and File names"
4075  IF Corr_flag THEN
4080     PRINT USING "2/,#,K";"The flow signal was incrementally corrected "
4085     PRINT "for changes in gas"
```

```
4090      PRINT "temperature and viscosity."
4095      PRINT USING "/,K,DD.D,K";"Relative Humidity = ";Rel_humid*100;"%"
4100      PRINT USING "/,K,DD.DD,K";"Body Temperature = ";Body_temp;" deg C"
4105      PRINT USING "/,K,DD.DD,K,DD.3D,K";"PH2O at ";Body_temp;" deg C = ";P
h2o_body;" torr"
4110      PRINT USING "/,K,3D.DD,K";"Barometric Pressure = ";Pb;" torr"
4115      PRINT USING "/,K,3D.DD,K";"Room temperature = ";Room_temp;" deg C"
4120    END IF
4125    PRINT USING "/,K,4D";"FLOW DC OFFSET = ";Bin_zero_flow
4130    PRINT USING "/,K,4D";"CO2 DC OFFSET = ";Co2_dc_offset
4135    PRINT USING "/,K,4D";"O2 DC OFFSET = ";O2_dc_offset
4140    PRINT USING "/,K,D.4DE";"CO2 CALIBRATION FACTOR = ";Co2_cal
4145    PRINT USING "/,K,D.4DE";"O2 CALIBRATION FACTOR = ";O2_cal
4150    PRINT USING "/,K";"THE NON CORRECTED FLOW CALIBRATION FACTORS"
4155    PRINT USING "/,#,K";"        INSPIRATORY FLOW CALIBRATION FACTOR = "
4160    PRINT USING "D.4DE";Non_c_i_flowcal
4165    PRINT USING "/,#,K";"        EXPIRATORY FLOW CALIBRATION FACTOR = "
4170    PRINT USING "D.4DE";Non_c_e_flowcal
4175    PRINT USING "2/,K";"THE TEMPERATURE CORRECTED FLOW CALIBRATION FACTORS"
4180    PRINT USING "/,#,K";"        INSPIRATORY FLOW CALIBRATION FACTOR = "
4185    PRINT USING "D.4DE";Corr_i_flowcal
4190    PRINT USING "/,#,K";"        EXPIRATORY FLOW CALIBRATION FACTOR = "
4195    PRINT USING "D.4DE,/";Corr_e_flowcal
4200    PRINT USING "/,#,K,MD.4DE,K";"TEMPERATURE CORRECTION = ";Ta;"X^2 "
4205    PRINT USING "#,K,MD.4DE,K";"+ ";Tb;"X "
4210    PRINT USING "K,MD.4DE";"+ ";Tc
4215    PRINT USING "/,K,DDD";"SAMPLING FREQUENCY =";S
4220    PRINT USING "/,K";"FLOW CALIBRATION FILENAME:  "&Ca1$
4225    PRINT USING "/,K";"CO2 DATA FILENAME:  "&C1$
4230    PRINT USING "/,K";"O2 DATA FILENAME:  "&O$
4235    PRINT USING "/,K";"FLOW DATA FILENAME:  "&V$
4240    PRINT USING "/,K";"TEMPERATURE DATA FILENAME:  "&T$
4245    PRINTER IS 1
4250    RETURN
4255    !
4260    !*** SUBROUTINE TO PLOT OUT THE FOUR BINARY DATA SETS
4265    !
4270  Dtplot:PRINT Formfd$;"                    DATA PLOTTING ROUTINE"
4275    PRINT USING "3/,K,5D,K";"THERE ARE ";No_points;" DATA POINTS AVAILABLE"
4280    !
4285    !*** OBTAIN STARTING AND ENDING POINTS TO PLOT
4290    !
4295    BEEP
4300    INPUT "ENTER STARTING POINT TO PLOT.",Start
4305    IF Start<1 OR Start>No_points THEN GOTO 4300
4310    BEEP
4315    INPUT "ENTER ENDING POINT TO PLOT.",End
4320    IF End<=Start OR End>No_points THEN GOTO 4315
4325    !
4330    !*** COMPUTER NUMBER OF POINTS TO PROCESS
4335    !
4340    P=End-Start+1
4345    !
4350    !*** COMPUTE PLOTTING OFFSET FOR MASS SPECTROMETER DELAYS
4355    !
4360    Offset=INT(Time_delay/1000*S)
4365    !
4370    !*** DETERMINE MAXIMUM AND MINIMUM VALUES FOR THE PLOTTED POINTS
4375    !
4380    Cmax=Line1(Start)
4385    Cmin=Line1(Start)
4390    Omax=Line2(Start)
4395    Omin=Line2(Start)
4400    Fmax=Line3(Start)
4405    Fmin=Line3(Start)
```

```
4410  Tmax=Line4(Start)
4415  Tmin=Line4(Start)
4420  FOR A=Start+1 TO End
4425     IF Cmax<Line1(A) THEN Cmax=Line1(A)
4430     IF Cmin>Line1(A) THEN Cmin=Line1(A)
4435     IF Cmax<Line2(A) THEN Cmax=Line2(A)
4440     IF Cmin>Line2(A) THEN Cmin=Line2(A)
4445     IF Fmax<Line3(A) THEN Fmax=Line3(A)
4450     IF Fmin>Line3(A) THEN Fmin=Line3(A)
4455     IF Tmax<Line4(A) THEN Tmax=Line4(A)
4460     IF Tmin>Line4(A) THEN Tmin=Line4(A)
4465  NEXT A
4470  !
4475  !*** DISPLAY THE MAXIMUM AND MINIMUM VALUES ON THE CRT
4480  !
4485  DISP
4490  PRINT
4495  PRINT "CO2 MAX: ";Cmax;TAB(25);"CO2 MIN: ";Cmin
4500  PRINT
4505  PRINT "O2 MAX: ";Omax;TAB(25);"O2 MIN: ";Omin
4510  PRINT
4515  PRINT "FLOW MAX: ";Fmax;TAB(25);"FLOW MIN: ";Fmin
4520  PRINT
4525  PRINT "TEMP MAX: ";Tmax;TAB(25);"TEMP MIN: ";Tmin
4530  !
4535  !*** ADJUST MAXIMUM AND MINIMUM PLOTTING VALUES FOR SMALL INPUT CHANGES
4540  !
4545  IF Fmax-Fmin<=100 THEN
4550     Fmax=4095
4555     Fmin=0
4560  END IF
4565  IF Cmax-Cmin<=100 THEN
4570     Cmax=4095
4575     Cmin=0
4580  END IF
4585  IF Omax-Omin<=100 THEN
4590     Omax=4095
4595     Omin=0
4600  END IF
4605  IF Tmax-Tmin<=100 THEN
4610     Tmax=4095
4615     Tmin=0
4620  END IF
4625  !
4630  !*** IF NO TEMPERATURE CALIBRATION DATA IS AVAILABLE, ONLY PLOT BINARY
4635  !*** TEMPERATURE DATA
4640  !
4645  IF (Ta=0) AND (Tc=0) THEN
4650     Ta=0
4655     Tc=0
4660     Tb=1
4665  END IF
4670  !
4675  !*** ALLOW USER TO SELECT PLOTTING DEVICE
4680  !
4685  BEEP
4690  ON KBD,15 GOTO Sel_plt_dev
4695  PRINT
4700  PRINT
4705  PRINT "You have 20 seconds to select a PLOTTING device."
4710  PRINT "If you do not select a device in that time, the"
4715  PRINT "CRT will automatically be selected.  To select"
4720  PRINT "a device press the space bar."
4725  FOR I=1 TO 100000
4730  !                    -- This loop delays approximately 20 seconds.
```

```
4735  NEXT I
4740  OFF KBD
4745  Q$="CRT"
4750  GOTO 4795                !  Skip and use the CRT.
4755 !
4760 !
4765 Sel_plt_dev:OFF KBD   !   Allow the user to select the plotting device.
4770  Q$=""
4775  BEEP
4780  INPUT "OUTPUT ON PLOTTER OR CRT ? (PLOTTER/CRT)",Q$
4785 !
4790 !
4795  IF Q$="PLOTTER" THEN
4800  !
4805  !*** SET SYSTEM FOR HP9872C PLOTTER
4810  !
4815     PLOTTER IS 705,"HPGL"
4820     PRINTER IS 705
4825     PRINT "VS5;"
4830     PRINTER IS 1
4835  ELSE
4840     PLOTTER IS 3,"INTERNAL"
4845     DUMP DEVICE IS 701
4850  END IF
4855  !
4860  !*** INITIALZE GRAPHICS SYSTEM
4865  !
4870  GRAPHICS ON
4875  GCLEAR
4880  PRINT Formfd$
4885  PEN 1
4890  DEG      !SET DEGREES MODE
4895  !
4900  !*** PLOT THE TEMPERATURE DATA ARRAY
4905  !
4910  View(0,100,4,24)
4915  Ctmin=Ta*Tmin^2+Tb*Tmin+Tc
4920  Ctmax=Ta*Tmax^2+Tb*Tmax+Tc
4925  Left=-P*.13
4930  Right=P
4935  WINDOW Left,Right,Ctmin,Ctmax
4940  CLIP 0,Right,Ctmin,Ctmax
4945  LINE TYPE 1      !SET FOR SOLID LINE
4950  CSIZE 2.8
4955  LDIR 0
4960  LORG 2      !SET LABEL ORIGIN TO POSITION 8
4965  AXES P/20,(Ctmax-Ctmin)/10,0,Ctmin
4970  CLIP Left,Right,Ctmin,Ctmax
4975  IF (Ctmin=Tmin) AND (Ctmax=Tmax) THEN 5015
4980  Incr=(Ctmax-Ctmin)/5
4985  LORG 8
4990  FOR I=Ctmin+Incr TO 1.001*(Ctmax-Incr) STEP Incr
4995     MOVE 0,I
5000     LABEL USING 5005;I
5005     IMAGE ZZ.D," "
5010  NEXT I
5015  CSIZE 3.3
5020  MOVE 1,Ta*Line4(Start)^2+Tb*Line4(Start)+Tc
5025  PEN 2
5030  FOR I=Start TO End
5035     Temp=Ta*Line4(I)*Line4(I)+Tb*Line4(I)+Tc
5040     PLOT I-Start+1,Temp
5045  NEXT I
5050  PENUP
5055  PEN 1
```

```
5060  CSIZE 2.5
5065  LINE TYPE 1
5070  !
5075  !*** LABEL THE TEMPERATURE PLOT
5080  !
5085  MOVE Left,(Ctmin+Ctmax)/2
5090  LORG 6
5095  LDIR 90
5100  LABEL "FLOW TEMP"
5105  MOVE Left*.85,(Ctmin+Ctmax)/2
5110  LABEL "DEGREES C"
5115  View(0,100,24,28)
5120  WINDOW 0,10,0,10
5125  MOVE 2,10
5130  LDIR 0
5135  LORG 3
5140  LABEL "START POINT ="&VAL$(Start)&"  END POINT = "&VAL$(End)
5145  MOVE 10,10
5150  LORG 9
5155  LABEL "1 TICK =";DROUND(P/(20*S),3);" SECONDS "
5160  !
5165  !*** PLOT THE FLOW DATA ARRAY
5170  !
5175  View(0,100,28,48)
5180  WINDOW Left,Right,Fmin*1.02,Fmax*1.02
5185  LINE TYPE 1
5190  CLIP 0,Right,Fmin,Fmax
5195  AXES P/20,(Fmax-Fmin)/10,0,Bin_zero_flow
5200  CLIP Left,Right,Fmin,Fmax
5205  IF NOT (Non_c_i_flowcal=0) THEN
5210     CSIZE 2.8
5215     LORG 8
5220     IMAGE DDD.DD," "
5225     Incr=(Fmax-Fmin)/5
5230  !
5235  !   Label the expiratory flow axis
5240  !
5245     FOR I=Bin_zero_flow TO Fmax STEP Incr
5250        MOVE 0,I
5255        LABEL USING 5220;(I-Bin_zero_flow)*Non_c_e_flowcal
5260     NEXT I
5265  !
5270  !   Label the inspiratory flow
5275     FOR I=Bin_zero_flow-Incr TO Fmin STEP -Incr
5280        MOVE 0,I
5285        LABEL USING 5220;(I-Bin_zero_flow)*Non_c_i_flowcal
5290     NEXT I
5295  END IF
5300  CSIZE 3.3
5305  MOVE 1,Line3(Start)
5310  PEN 2
5315  FOR I=Start TO End
5320     PLOT I-Start+1,Line3(I)
5325  NEXT I
5330  PENUP
5335  PEN 1
5340  LINE TYPE 1
5345  CSIZE 2.5
5350  !
5355  !*** LABEL THE FLOW PLOT
5360  !
5365  MOVE Left,(Fmax+Fmin)/2
5370  LORG 6
5375  LDIR 90
5380  LABEL "FLOW [L/S]"
```

```
5385  !
5390  !*** PLOT THE O2 DATA ARRAY
5395  !
5400  View(0,100,52,72)
5405  WINDOW Left,Right,Cmin,Cmax
5410  CLIP 0,Right,Cmin,Cmax
5415  AXES P/20,Cmax/10,0,Cmin
5420  CLIP Left,Right,Cmin,Cmax
5425  LDIR 0
5430  IF NOT (O2_cal=0) THEN
5435     CSIZE 2.8
5440     LORG 8
5445     IMAGE DD.D," "
5450     Incr=(Cmax-Cmin)/5
5455     FOR I=Cmin+Incr TO (Cmax-Incr)*1.01 STEP Incr
5460        MOVE 0,I
5465        LABEL USING 5445;((I-O2_dc_offset)*O2_cal+O1)*100
5470     NEXT I
5475  END IF
5480  MOVE 1,Line2(Start+Offset)-O2_dc_offset
5485  PEN 2
5490  FOR I=Start TO End-Offset
5495     PLOT I-Start+1,Line2(I+Offset)
5500  NEXT I
5505  PENUP
5510  PEN 1
5515  !
5520  !*** LABEL THE O2 PLOT
5525  !
5530  MOVE Left,(Cmax+Cmin)/2
5535  LORG 6
5540  LDIR 90
5545  CSIZE 2.5
5550  LABEL "FRACTIONAL O2"
5555  MOVE Left*.85,(Cmax+Cmin)/2
5560  LABEL "CONCENTRATION"
5565  !
5570  !*** PLOT THE CO2 DATA ARRAY
5575  !
5580  View(0,100,76,96)
5585  WINDOW Left,Right,Cmin,Cmax
5590  CLIP 0,Right,Cmin,Cmax
5595  AXES P/20,(Cmax-Cmin)/10,0,Cmin
5600  CLIP Left,Right,Cmin,Cmax
5605  IF NOT (Co2_cal=0) THEN
5610     LDIR 0
5615     CSIZE 2.8
5620     LORG 8
5625     Incr=(Cmax-Cmin)/5
5630     FOR I=Cmin+Incr TO (Cmax-Incr)*1.01 STEP Incr
5635        MOVE 0,I
5640        LABEL USING 5445;(I-Co2_dc_offset)*Co2_cal*100
5645     NEXT I
5650  END IF
5655  LINE TYPE 1
5660  MOVE 1,Line1(Start+Offset)
5665  PEN 2
5670  FOR I=Start TO End-Offset
5675     PLOT I-Start+1,Line1(I+Offset)
5680  NEXT I
5685  PENUP
5690  PEN 1
5695  !
5700  !*** LABEL THE CO2 PLOT
5705  !
```

```
5710  LORG 6
5715  CSIZE 2.3
5720  LDIR 90
5725  MOVE Left,(Cmax+Cmin)/2
5730  LABEL "FRACTIONAL CO2"
5735  MOVE Left*.85,(Cmax+Cmin)/2
5740  LABEL "CONCENTRATION"
5745  Q$=""
5750  !
5755  !*** PUT PEN AWAY AND PAUSE FOR USER TO OBSERVE PLOT
5760  !
5765  PEN 0
5770  BEEP
5775  PAUSE
5780  !
5785  !*** ONCE PAUSE IS COMPLETE, PROMPT THE USER FOR REDO OF GRAPHICS
5790  !
5795  GRAPHICS OFF
5800  Q$=""
5805  BEEP
5810  INPUT "REDO GRAPHICS? (Y/N)",Q$
5815  !
5820  !*** IF DESIRED, GO START PLOTTING SUBROUTINE OVER
5825  !
5830  IF Q$="Y" OR Q$="y" THEN GOTO Dtplot
5835  !
5840  !*** OTHERWISE, RETURN BACK TO BEGIN BREATH-BY-BREATH ANALYSIS
5845  !
5850  RETURN
5855  !
5860  !***SUBROUTINE FOR RETRIEVING FOUR CHANNELS OF BINARY DATA FROM STORAGE
5865  !
5870  !
5875  !*** GET THE NAMES OF THE FOUR FILES
5880  !
5885 Rtdata:BEEP
5890  INPUT "ENTER THE CO2 SIGNAL FILE NAME",C1$
5895  BEEP
5900  O$="O"&C1$[2,LEN(C1$)]  ! Set the default oxygen file name
5905  INPUT "ENTER THE O2 SIGNAL FILE NAME",O$
5910  BEEP
5915  V$="V"&C1$[2,LEN(C1$)]  ! Set the default flow file name
5920  INPUT "ENTER THE FLOW SIGNAL FILE NAME",V$
5925  BEEP
5930  T$="T"&C1$[2,LEN(C1$)]  ! Set the default temperature file name
5935  INPUT "ENTER THE FLOW TEMPERATURE SIGNAL FILE NAME",T$
5940  !
5945  !*** SELECT PROPER MASS STORAGE UNIT AND OPEN THE FILES
5950  !
5955  MASS STORAGE IS ":HP9895,700,0"
5960  ASSIGN @Co2_path TO C1$
5965  ASSIGN @O2_path TO O$
5970  ASSIGN @Flow_path TO V$
5975  ASSIGN @Temp_path TO T$
5980  !
5985  !*** TELL PROGRAM WHEN TO QUIT READING THE FILES
5990  !
5995  ON END @Co2_path GOTO 6055
6000  ON END @O2_path GOTO 6080
6005  ON END @Flow_path GOTO 6105
6010  ON END @Temp_path GOTO 6130
6015  !
6020  !*** READ THE CO2 DATA FILE
6025  !
6030  ENTER @Co2_path;Cmax,Cmin
```

```
6035  ENTER @Co2_path;Line1(*)
6040  !
6045  !*** READ THE O2 DATA FILE
6050  !
6055  ENTER @O2_path;Omax,Omin
6060  ENTER @O2_path;Line2(*)
6065  !
6070  !*** READ THE FLOW DATA FILE
6075  !
6080  ENTER @Flow_path;Fmax,Fmin
6085  ENTER @Flow_path;Line3(*)
6090  !
6095  !*** READ THE TEMPERATURE DATA FILE
6100  !
6105  ENTER @Temp_path;Tmax,Tmin
6110  ENTER @Temp_path;Line4(*)
6115  !
6120  !*** CLOSE THE FILES AND SET MASS STORAGE UNIT BACK TO INTERNAL FLOPPY
6125  !
6130  ASSIGN @Co2_path TO *
6135  ASSIGN @O2_path TO *
6140  ASSIGN @Flow_path TO *
6145  ASSIGN @Temp_path TO *
6150  MASS STORAGE IS ":INTERNAL"
6155  !
6160  !*** RETURN BACK TO ANALYSIS ROUTINE
6165  !
6170  RETURN
6175  !
6180  !*** SUBROUTINE FOR RETRIEVING CALIBRATION FACTORS FROM MASS STORAGE
6185  !
6190  !
6195  !*** GET CALIBRATION FILE NAME AND ASSIGN PROPER MASS STORAGE UNIT
6200  !
6205 Rtcal:BEEP
6210  INPUT "ENTER CALIBRATION FILE FILENAME",Cal$
6215  !
6220  !*** OPEN CALBRATION FILE AND TELL PROGRAM WHEN TO STOP READING FILE
6225  !
6230  ASSIGN @Cal_path TO Cal$&":HP9895,700,0"
6235  ON END @Cal_path GOTO Cal_fl_is_read
6240  !
6245  !*** READ PARAMETERS IN CALIBRATION FILE
6250  !
6255  ENTER @Cal_path;Co2_dc_offset,O2_dc_offset,Bin_zero_flow
6260  ENTER @Cal_path;Co2_cal,O2_cal
6265  ENTER @Cal_path;Non_c_i_flowcal,Non_c_e_flowcal
6270  ENTER @Cal_path;Time_delay,S,O1,Ta,Tb,Tc,Date$
6275  ENTER @Cal_path;Corr_i_flowcal,Corr_e_flowcal
6280  ENTER @Cal_path;Pb,Rel_humid,Room_temp
6285  !
6305 Cal_fl_is_read: ! The calibration file has been entered so return.
6310  ASSIGN @Cal_path TO *          !close Cal_path
6315  RETURN
6320  !
6325  !*** SUBROUTINE TO DETERMINE MASS SPECTROMETER TIME DELAY ON A BREATH-
6330  !*** BY-BREATH BASIS
6335  !
6340 Bbb_time_delay:Temp_a=A
6345  Temp_z=Z
6350  C=Co2_dc_offset      !SET C TO BINARY ZERO CO2 VALUE
6355  !
6360  !*** BEGINNING AT POINT ON FLOW SIGNAL CORRESPONDING TO ZERO FLOW,
6365  !*** LOCATE PEAK END EXPIRED CO2 VALUE
6370  !
```

```
6375 Hunt_max:Z=A              !CO2 INDEX CORRESPONDING TO ZERO FLOW
6380  IF No_points-Z<50 THEN Bomb_out    !MAKE SURE 50 POINTS FOLLOW ZERO CROS
SING
6385  Co2max=Line1(Z)-C       !SET INITIAL CO2MAX LEVEL TO FIRST CO2 VALUE
6390  Max_index=Z
6395  FOR Wye=Z TO Z+.75*S   !SEARCH AHEAD FOR THE MAX END EXPIRED FCO2 VALUE
6400     IF Line1(Wye)-C>Co2max THEN Max_index=Wye
6405     IF Line1(Wye)-C>Co2max THEN Co2max=Line1(Wye)-C
6410  NEXT Wye
6415  !
6420  !*** FIND THE MIDDLE INDEX (THAT POINT CORRESPONDING TO 50% OF THE MAX.
6425  !*** END EXPIRED CO2 VALUE)
6430  !
6435  Mid_index=Max_index
6440  FOR Wye=Max_index TO Z+.75*S
6445     IF Line1(Wye)-C>.5*Co2max THEN Next_wye
6450     Mid_index=Wye      ! INDEX OF THE 50% DOWN POINT ON FCO2 CURVE
6455     GOTO Set_limits
6460 Next_wye:NEXT Wye
6465  !
6470  !** FIND THE MINIMUM INDEX (THAT POINT CORRESPONDING TO THE MINIMUM END
6475  !** EXPIRED CO2 VALUE
6480  !
6485 Set_limits:Min_index=Mid_index
6490  Co2min=Line1(Mid_index)-C
6495  FOR Wye=Mid_index TO Mid_index+Mid_index-Max_index
6500     IF Line1(Wye)-C>Co2min THEN Next_y
6505     Min_index=Wye
6510     Co2min=Line1(Wye)-C
6515 Next_y:NEXT Wye
6520  !
6525  !*** INITIALIZE INDEXES FOR START OF INTEGRATION OF CO2 SIGNAL
6530  !
6535  Best_match=1.E+50
6540  Beg_pt=Max_index
6545  End_pt=Min_index
6550  !
6555  !*** EXIT ROUTINE IF ADEQUATE NUMBER OF POINTS DO NOT EXIST
6560  !
6565  IF End_pt>No_points THEN Bomb_out
6570  Beg_intg=Max_index
6575  End_intg=Min_index
6580  !
6585  !*** USE TRAPEZOIDAL RULE TO COMPUTE THE AREA ABOVE AND BELOW THE CURVE
6590  !
6595 New_sum:Asum=0
6600  Bsum=0
6605  !
6610  !*** FIRST, ABOVE THE CURVE, 1/2 OF FIRST AND LAST POINTS
6615  !
6620  Asum=.5*(Co2max-(Line1(Beg_pt)-C))+.5*(Co2max-(Line1(Beg_intg)-C))
6625  FOR Wye=Beg_pt+1 TO Beg_intg-1
6630     Asum=Asum+Co2max-(Line1(Wye)-C)
6635  NEXT Wye
6640  !
6645  !*** NEXT, BELOW THE CURVE, 1/2 OF FIRST AND LAST POINTS
6650  !
6655  Bsum=.5*(Line1(Beg_intg)-Co2_dc_offset)+.5*(Line1(End_pt)-Co2_dc_offset)
6660  FOR Wye=Beg_intg+1 TO End_pt-1
6665     Bsum=Bsum+Line1(Wye)-Co2_dc_offset
6670  NEXT Wye
6675  !
6680  !*** COMPUTE DIFFERENCE IN THE TWO AREAS
6685  !
6690  Adiff=ABS(Asum-Bsum)
```

```
6695  !
6700  !*** IF AREA DIFFERENCE IS A MINIMUM, REMEMBER THE PROPER INDEX
6705  !
6710  IF Adiff<Best_match THEN Best_index=Beg_intg
6715  IF Adiff<Best_match THEN Best_match=Adiff
6720  !
6725  !*** BUMP THE CENTER INTEGRATION POINT AND GO TRY ANOTHER IF STILL
6730  !*** WITHIN OUTER LIMITS OF INTEGRATION
6735  !
6740  Beg_intg=Beg_intg+1     ! IF NOT TO ENDPOINT SHIFT THE CENTER INTGR PONT
6745  IF Beg_intg<=End_intg THEN New_sum     ! GO COMPUTE NEW AREAS FOR LIMITS
6750  !
6755  !*** COMPUTE MASS SPECTROMETER TIME DELAY FOR THIS BREATH AND RETURN
6760  !*** BACK TO ANALYSIS ROUTINE WITH VARIABLES UNALTERED.
6765  !
6770 Bomb_out:  ! DATA STREAM EXHAUSTED
6775  Time_delay=(Best_index-Z)/S*1000
6780  A=Temp_a
6785  Z=Temp_z
6790  RETURN
6795  END
6800 !*************************************************************************
6805 !
6810 !
```

```
6815  SUB Calc_sig_val(INTEGER A,Z,Insp_or_expr$,REAL Incr_vol,Fo2,Fco2)
6820     GOTO Skip_comments
6825    !*************************************************************************
6830    !*************************************************************************
6835    !****                                                                 ****
6840    !****  Section Title:  Inspiratory and expiratory signal values.      ****
6845    !****                                                                 ****
6850    !****  Revision     Date                 Programer                  ****
6855    !****  --------     ------------         -----------------          ****
6860    !****    1.0        Janurary 1, 1985     Michael Masters            ****
6865    !****                                                                 ****
6870    !*************************************************************************
6875    !****                                                                 ****
6880    !****  Purpose:   To calculate the incremental signal values for      ****
6885    !****       current data sample.  The incremental volume is calc-     ****
6890    !****       ulated and is converted to STPD conditions if the         ****
6895    !****       Temperature correction is requested.  The fractional      ****
6900    !****       carbon dioxide and oxygen fractional concentrations       ****
6905    !****       are also calculated.                                      ****
6910    !****                                                                 ****
6915    !****  Parameters passed from the calling routine                     ****
6920    !****       A - The index to the flow and temperature data.           ****
6925    !****                                                                 ****
6930    !****       Z - The index to the gas (O2 and CO2) data.               ****
6935    !****                                                                 ****
6940    !****       Insp_or_expr$ - a flag indicating an inspiratory or       ****
6945    !****            an expiratory flow.                                  ****
6950    !****                                                                 ****
6955    !****  Parameters returned to the calling routine                     ****
6960    !****       Incr_vol - the incremental volume of air under STPD       ****
6965    !****            conditions.                                          ****
6970    !****                                                                 ****
6975    !****       Fo2,Fco2 - the dry fractional values of O2 and CO2.       ****
6980    !****                                                                 ****
6985    !****  Routine(s) Called:                                             ****
6990    !****       Function FNRel_viscos                                     ****
6995    !****                                                                 ****
7000    !*************************************************************************
7005  Skip_comments:  !
7010     COM /Resp_data1/ INTEGER Line1(1:24000),Line2(1:24000)
7015     COM /Resp_data2/ INTEGER Line3(1:24000),Line4(1:24000)
7020     COM /Cal_data2/ INTEGER Co2_dc_offset,O2_dc_offset,Bin_zero_flow,S
7025     COM /Cal_data3/ REAL Co2_cal,O2_cal
7030     COM /Cal_data6/ REAL Ta,Tb,Tc
7035     COM /Cal_data7/ REAL Time_delay,O1,Pb,Rel_humid,Room_temp
7040     COM /Anal_data6/ INTEGER Corr_flag,REAL Room_fh2o,T
7045     COM /Anal_data7/ Vap(0:250),Fh2o_sat(0:250)
7050    !
7055    !              Determine the CO2 and O2 fractional concentrations
7060    !
7065     Fco2=(Line1(Z)-Co2_dc_offset)*Co2_cal
7070     Fo2=(Line2(Z)-O2_dc_offset)*O2_cal+O1
7075    !
7080    !
7085     IF Corr_flag THEN
7090        Temperature=Ta*Line4(A)*Line4(A)+Tb*Line4(A)+Tc
7095        IF Insp_or_expr$="INSP" THEN
7100           Fh2o=Room_fh2o
7105        ELSE
7110           Fh2o=Fh2o_sat((Temperature-20)*10)
7115        END IF
7120        Fco2_wet=Fco2*(1.0-Fh2o)
7125        Fo2_wet=Fo2*(1.0-Fh2o)
7130        Fn2_wet=1.0-Fco2_wet-Fo2_wet-Fh2o
7135    !
```

```
7140   !              The binary flow is equal to (binary delta-P/viscosity).
7145         Rel_viscosity=FNRel_viscos(Fo2_wet,Fco2_wet,Fn2_wet,Fh2o,Temperat
ure)
7150         Bin_flow=(Line3(A)-Bin_zero_flow)/Rel_viscosity
7155   !
7160   !              Convert the incremental binary volume to STPD coditions.
7165         Incr_vol=Bin_flow*(1.0-Fh2o)*273.15/(273.15+Temperature)*Pb/760.0
*T
7170      ELSE
7175         Incr_vol=(Line3(A)-Bin_zero_flow)*T
7180      END IF
7185   SUBEND
7190   !
7195   !



7200   DEF FNRel_viscos(Fo2,Fco2,Fn2,Fh2o,Temperature)
7205      GOTO Skip_comments
7210   !*************************************************************************
7215   !*************************************************************************
7220   !****                                                                  ****
7225   !****  Routine Title:      Relative viscosity function                 ****
7230   !****                                                                  ****
7235   !****  HP BASIC FILENAME:  Analysis                                    ****
7240   !****                                                                  ****
7245   !****  Department of Electrical and Computer Engineering,              ****
7250   !****  Kansas State University                                         ****
7255   !****                                                                  ****
7260   !****  Revision     Date                 Programer                     ****
7265   !****  --------     ------------         -----------------             ****
7270   !****    1.0        August 23, 1984      Michael Masters               ****
7275   !****                                                                  ****
7280   !*************************************************************************
7285   !****                                                                  ****
7290   !****  Purpose: To calculate the relative viscosity of a gas           ****
7295   !****      containing carbon dioxide (CO2), oxygen (O2), nitrogen      ****
7300   !****      (N2), and water vapor at a temperature (T).  The            ****
7305   !****      viscosity is expressed relative to room air under STPD      ****
7310   !****      conditions.  The calling sequence is:                       ****
7315   !****                                                                  ****
7320   !****   variable = FNRel_viscos(Fo2,Fco2,Fn2,Fh2o,Temperature)         ****
7325   !****                                                                  ****
7330   !****      where:                                                      ****
7335   !****                                                                  ****
7340   !****         Fo2,Fco2,Fn2,Fh2o -  the fractional gas conc.            ****
7345   !****             of O2, CO2, and water vapor in the gas.              ****
7350   !****                                                                  ****
7355   !****        Temperature - the insantaneous temperature of the gas     ****
7360   !****                                                                  ****
7365   !*************************************************************************
7370   Skip_comments:  !
7375   !
7380   !           Calculate the individual viscosities
7385      Vis_n2=165.4+.451*Temperature
7390      Vis_o2=188.0+.606*Temperature
7395      Vis_co2=187.1+.612*Temperature
7400      Vis_h2o=117.6+.506*Temperature
7405   !
7410      Rel_vis=(Fo2*Vis_o2+Fco2*Vis_co2+Fn2*Vis_n2+Fh2o*Vis_h2o)/170.17502
7415      RETURN Rel_vis
7420   FNEND
7425   !
7430   !
```

```
7435  DEF FNInspiration(INTEGER Pntr)
7440     GOTO Skip_comments
7445    !*************************************************************************
7450    !*************************************************************************
7455    !****                                                                 ****
7460    !****  Function Title:  Inspiratory flow.                             ****
7465    !****                                                                 ****
7470    !****  Revision     Date                Programer                   ****
7475    !****  --------     ------------        -----------------           ****
7480    !****    1.0        Janurary 1, 1985    Michael Masters             ****
7485    !****                                                                 ****
7490    !*************************************************************************
7495    !****                                                                 ****
7500    !****  Purpose:  To determine if the flow point is an inspiration.****
7505    !****     The point is an inspiratory flow if it is less than the ****
7510    !****     zero flow value or any of the next five data points are ****
7515    !****     less than or equal to the zero flow value.                  ****
7520    !****                                                                 ****
7525    !****  Parameters passed from the calling routine                     ****
7530    !****                                                                 ****
7535    !****       Pntr - Points to the flow data point.                     ****
7540    !****                                                                 ****
7545    !****  The routine returns a logical value of True if the flow is ****
7550    !****  inspiratory, otherwise false.                                  ****
7555    !****                                                                 ****
7560    !****  Routine(s) Called:       None                                  ****
7565    !****                                                                 ****
7570    !*************************************************************************
7575    !*************************************************************************
7580  Skip_comments: !
7585     COM /Resp_data2/ INTEGER Line3(1:24000),Line4(1:24000)
7590     COM /Cal_data2/ INTEGER Co2_dc_offset,O2_dc_offset,Bin_zero_flow,S
7595     INTEGER Inspiration,I
7600     Inspiration=(Line3(Pntr)<Bin_zero_flow)
7605     I=1
7610     WHILE NOT (Inspiration) AND I<=5
7615        Inspiration=(Line3(Pntr+I)<=Bin_zero_flow)
7620        I=I+1
7625     END WHILE
7630     RETURN Inspiration
7635  FNEND
7640  !
7645  !
```

```
7650  DEF FNExpiration(INTEGER Pntr)
7655     GOTO Skip_comments
7660    !*************************************************************************
7665    !*************************************************************************
7670    !****                                                                 ****
7675    !****  Function Title:  Expiratory flow.                              ****
7680    !****                                                                 ****
7685    !****  Revision     Date                Programer                   ****
7690    !****  --------     ------------        -----------------           ****
7695    !****    1.0        Janurary 1, 1985    Michael Masters             ****
7700    !****                                                                 ****
7705    !*************************************************************************
7710    !****                                                                 ****
7715    !****  Purpose:   To determine if the flow point is an expiration.****
7720    !****     The point is an expiratory flow if it is greater than       ****
7725    !****     the zero flow value or any of the next five data points     ****
7730    !****     are greater than or equal to the zero flow value.           ****
7735    !****                                                                 ****
7740    !****  Parameters passed from the calling routine                     ****
7745    !****                                                                 ****
7750    !****      Pntr - Points to the flow data point.                      ****
7755    !****                                                                 ****
7760    !****  The routine returns a logical value of True if the flow is     ****
7765    !****  expiratory, otherwise false.                                   ****
7770    !****                                                                 ****
7775    !****  Routine(s) Called:      None                                   ****
7780    !****                                                                 ****
7785    !*************************************************************************
7790    !*************************************************************************
7795  Skip_comments: !
7800     COM /Resp_data2/ INTEGER Line3(1:24000),Line4(1:24000)
7805     COM /Cal_data2/ INTEGER Co2_dc_offset,O2_dc_offset,Bin_zero_flow,S
7810     INTEGER Expiration,I
7815     Expiration=(Line3(Pntr)>Bin_zero_flow)
7820     I=1
7825     WHILE NOT (Expiration) AND I<=5
7830        Expiration=(Line3(Pntr+I)>=Bin_zero_flow)
7835        I=I+1
7840     END WHILE
7845     RETURN Expiration
7850  FNEND
```

```
7855  DEF FNMax_co2_pntr(INTEGER Line1(*),A)
7860   !*********************************************************************
7865   !*********************************************************************
7870   !****                                                             ****
7875   !****  Function Title:  Find maximum CO2 value.                   ****
7880   !****                                                             ****
7885   !****  Revision     Date                 Programer                ****
7890   !****  --------     ------------         -----------------        ****
7895   !****    1.0        December 1, 1984     Rick Pieschl             ****
7900   !****                                                             ****
7905   !*********************************************************************
7910   !****                                                             ****
7915   !****  Purpose:  To determine the point where the maximum CO2     ****
7920   !****     value occured.  This point is assumed to be the end     ****
7925   !****     expiratory point.                                       ****
7930   !****                                                             ****
7935   !****  Parameters passed from the calling routine                 ****
7940   !****                                                             ****
7945   !****       Line1 - The Binary CO2 data.                          ****
7950   !****                                                             ****
7955   !****       A - The point corresponding to expiratory/insprirator ****
7960   !****          transition.                                        ****
7965   !****                                                             ****
7970   !****  The routine returns a value equal to the end expirate value****
7975   !****                                                             ****
7980   !****  Routine(s) Called:      None                               ****
7985   !****                                                             ****
7990   !*********************************************************************
7995   !*********************************************************************
8000     INTEGER Max,I
8005     Max=A-50
8010     I=A-50
8015     REPEAT
8020        IF Line1(I)>Line1(Max) THEN Max=I
8025        Max=I
8030        I=I+1
8035     UNTIL I>A
8040     RETURN Max
8045  FNEND
8050  !
8055  !
8060  SUB Disp_results(INTEGER Breaths,Corr_flag)
8065   !*********************************************************************
8070   !****                                                             ****
8075   !****  Routine Title:     Disp_results                            ****
8080   !****                                                             ****
8085   !****  HP BASIC FILENAME: ANALYSIS                                ****
8090   !****                                                             ****
8095   !****  Department of Electrical and Computer Engineering,         ****
8100   !****  Kansas State University                                    ****
8105   !****                                                             ****
8110   !****  Revision     Date                 Programer                ****
8115   !****  --------     ------------         -----------------        ****
8120   !****    1.0        Sept. 10, 1984       Michael Masters          ****
8125   !****                                                             ****
8130   !*********************************************************************
8135   !****                                                             ****
8140   !****  Purpose:                                                   ****
8145   !****          This subroutine displays the breath by breath      ****
8150   !****       respiratory data calculated in the analysis routine.  ****
8155   !****                                                             ****
8160   !****                                                             ****
8165   !****  Routine(s) Called                                          ****
8170   !****       Max_min -  Determines the maximum and minum values of ****
8175   !****                  the respiratory variable to be plotted.    ****
```

```
8180  !****                                                                ****
8185  !****       Veiw -  Sets the plotting areas on the screen or the     ****
8190  !****               plotter.                                         ****
8195  !****                                                                ****
8200  !****       Make_axes - Defines the dimensions of the plotting       ****
8205  !****               area based on the maximum and minum values to    ****
8210  !****               be plotted.  Draws and labels the axes and       ****
8215  !****               also labels the plot.                            ****
8220  !****                                                                ****
8225  !****       Plot_values - Plots the values of the respiratory        ****
8230  !****               variable.                                        ****
8235  !****                                                                ****
8240  !****       Print_values - Plots the values of the respiratory       ****
8245  !****               variable.                                        ****
8250  !****                                                                ****
8255  !*************************************************************************
8260  !*************************************************************************
8265    COM /Cal_data1/ Cal$[10],Date$[25]
8270    COM /Anal_data2/ INTEGER True,False
8275    COM /Anal_data3/ INTEGER Time_of_brth,O2_cons,Co2_prod,Insp_min_vent
8280    COM /Anal_data4/ INTEGER Expr_min_vent,V_tid_insp,V_tid_expr,Resp_fr
eq
8285    COM /Anal_data5/ INTEGER Breath_good,Resp_quotient,Formfd$[1]
8290    COM /Anal_data5a/ INTEGER Frc_o2_cons,Frc_co2_prod,Volume_of_lung
8295    COM /Anal_data8/ Resp_var(0:12,1:500)
8300    COM /Anal_data9/ Name$[25],Comment$[46]
8305    COM /Plot_resp_data1/ INTEGER Omit_bad,Num_breaths,No_b_avg
8310    COM /Plot_resp_data2/ Start_time,End_time,W_period,W_wdth
8315    INTEGER Fig1,Fig2
8320    DIM Time1$[20],Event1$[50],Time2$[20],Event2$[50]
8325    DIM Title$[40],Avg_bad_comnt$[65],Avg_type$[7],Result_type$[5]
8330    Num_breaths=Breaths
8335  !                 Repeat until the user no longer desires to plot the
8340  !                 results.
8345    REPEAT
8350       PRINT Formfd$
8355       PRINT "Which of the following options do you wish to do?"
8360       PRINT
8365       PRINT "   1.  Plot the results."
8370       PRINT "   2.  Print the results."
8375       REPEAT
8380          Q$=""
8385          BEEP
8390          INPUT "Enter the option (1 or 2) ",Q$
8395       UNTIL Q$="1" OR Q$="2"
8400       IF Q$="1" THEN
8405          Result_type$="PLOT"
8410       ELSE
8415          Result_type$="PRINT"
8420       END IF
8425       PRINT Formfd$,"How do you wish to combine breaths?"
8430       PRINT
8435       PRINT "   1.  The individual breaths."
8440       PRINT "   2.  Average individual breaths."
8445       PRINT "   3.  Avarage windows of breaths."
8450       REPEAT
8455          Q$=""
8460          BEEP
8465          INPUT "Enter the option (1,2 or 3) ",Q$
8470       UNTIL Q$="1" OR Q$="2" OR Q$="3"
8475       PRINT Formfd$
8480       SELECT Q$
8485       CASE ="1"
8490          Avg_type$="AVERAGE"
8495          No_b_avg=1
```

```
8500          Start_time=Resp_var(Time_of_brth,1)
8505          IF Start_time<=5 THEN Start_time=0
8510          End_time=Resp_var(Time_of_brth,Num_breaths)
8515       CASE ="2"
8520          Avg_type$="AVERAGE"
8525          BEEP
8530          INPUT "How many breaths do you wish to average?",No_b_avg
8535          Start_time=Resp_var(Time_of_brth,1)
8540          IF Start_time<=5 THEN Start_time=0
8545          End_time=Resp_var(Time_of_brth,Num_breaths)
8550       CASE ="3"
8555          Avg_type$="WINDOW"
8560          BEEP
8565          INPUT "Enter the windowing period. (seconds)",W_period
8570          BEEP
8575          INPUT "Enter the window width. (seconds)",W_wdth
8580          BEEP
8585          INPUT "Enter the starting time (seconds)",Start_time
8590          BEEP
8595          INPUT "Enter the ending time (seconds)",End_time
8600       END SELECT
8605    !
8610    !                  See if the user wishes to omit those breaths tagged
8615    !                  as bad by the data calculation routine.  These bad
8620    !                  breaths are those with an inspiratory tidal volume
8625    !                  or expiratory tidal volume below a specified volume.
8630       Q$=""
8635       BEEP
8640       INPUT "Do you wish to omit the bad breaths? (Y/N)",Q$
8645       IF Q$="Y" OR Q$="y" THEN
8650          Omit_bad=True
8655       ELSE
8660          Omit_bad=False
8665       END IF
8670       PRINT Formfd$
8675       IF Result_type$="PLOT" THEN
8680          GCLEAR
8685          GINIT
8690    !
8695    !                              Prompt the user for the plots he wants.
8700          PRINT
8705          PRINT "Which of the following two respiratory variables"
8710          PRINT "do you wish to plot?  Enter the number "
8715          PRINT "corresponding to the variable you wish."
8720          PRINT
8725          PRINT "    1.  Oxygen consumed"
8730          PRINT "    2.  Carbon dioxide produced"
8735          PRINT "    3.  Aveolar oxygen consumption"
8740          PRINT "    4.  Aveolar carbon dioxide production"
8745          PRINT "    5.  Inspiratory Minute ventilation"
8750          PRINT "    6.  Expiratory Minute ventilation"
8755          PRINT "    7.  Inspiratory Tidal volume"
8760          PRINT "    8.  Expiratory Tidal volume"
8765          PRINT "    9.  Lung volume"
8770          PRINT "   10.  Respiratory frequency"
8775          PRINT "   11.  Respiratory quotinet"
8780          REPEAT
8785             BEEP
8790             INPUT "Enter the first plot",Fig1
8795             IF Fig1<O2_cons OR Fig1>Resp_quotient THEN
8800                PRINT "Invalid response"
8805             END IF
8810          UNTIL Fig1>=O2_cons AND Fig1<=Resp_quotient
8815          REPEAT
8820             BEEP
```

```
8825            INPUT "Enter the second plot",Fig2
8830            IF Fig2<O2_cons OR Fig2>Resp_quotient THEN
8835               PRINT "Invalid response"
8840            END IF
8845         UNTIL Fig2>=O2_cons AND Fig2<=Resp_quotient
8850         Plot_dev$="CRT"            ! Set the default
8855         BEEP
8860         INPUT "OUTPUT ON PLOTTER OR CRT ? (PLOTTER/CRT)",Plot_dev$
8865         IF Plot_dev$="PLOTTER" THEN
8870            PRINT
8875            PRINT "Press CONTINUE when you have prepared the plotter"
8880            PAUSE
8885            PLOTTER IS 705,"HPGL"
8890            PRINTER IS 705
8895            PRINT "VS5;"
8900            PRINTER IS 1
8905            PEN 1
8910         ELSE
8915            DUMP DEVICE IS 701
8920         END IF
8925         PRINT Formfd$
8930   !
8935   !
8940   !
8945   !*********************************************************************
8950   !                  Determine the maximimum and minimum values to be
8955   !                  plotted.  Allow for the user to define their own
8960   !                  maximum and minum valuse of R.
8965   !
8970         Max_min(Fig1,Avg_type$,Max1,Min1)
8975         IF ABS(Min1)<.001 THEN Min1=0
8980         Max_min(Fig2,Avg_type$,Max2,Min2)
8985         IF ABS(Min2)<.001 THEN Min2=0
8990         IF Fig1=Resp_quotient THEN
8995            BEEP
9000            INPUT "Enter min value of R or CONT for calculated min.",Mi
n1
9005            BEEP
9010            INPUT "Enter max value of R or CONT for calculated max.",Ma
x1
9015         END IF
9020         IF Fig2=Resp_quotient THEN
9025            BEEP
9030            INPUT "Enter min value of R or CONT for calculated min.",Mi
n2
9035            BEEP
9040            INPUT "Enter max value of R or CONT for calculated max.",Ma
x2
9045         END IF
9050   !
9055   !
9060   !
9065   !*********************************************************************
9070   !      Plot the values.
9075   !
9080   !               Set up the plotting area, determine the starting and
9085   !               ending times, make the axes, and finally plot the values
9090         GRAPHICS ON
9095         VIEW(0,100,58,100)
9100   !                           *************** First Plot *************
9105         Make_axes(Fig1,Corr_flag,Min1,Max1)
9110         Plot_values(Fig1,Avg_type$)
9115   !                           *************** Second Plot *************
9120         VIEW(0,100,15,57)
9125         Make_axes(Fig2,Corr_flag,Min2,Max2)
```

```
9130          Plot_values(Fig2,Avg_type$)
9135    !
9140    !*************************************************************************
9145    !                  Title the plot and specify events
9150    !
9155          BEEP
9160          View(0,100,0,15)
9165          WINDOW 0,100,0,100
9170          CSIZE 3.5
9175          IF Avg_type$="AVERAGE" THEN
9180             Avg_bad_comnt$=VAL$(No_b_avg)&" Breath(s)/point"
9185          ELSE
9190             Avg_bad_comnt$=VAL$(W_period)&" s Window period;    "
9195             Avg_bad_comnt$=Avg_bad_comnt$&VAL$(W_wdth)&"s Window width"
9200          END IF
9205          IF Omit_bad THEN
9210             Avg_bad_comnt$=Avg_bad_comnt$&";   Bad breaths omitted"
9215          END IF
9220          LORG 6
9225          MOVE 50,98
9230          LABEL Avg_bad_comnt$
9235          PENUP
9240          WAIT 1   ! Allow the user to look at the data a second.
9245          GRAPHICS OFF
9250    !
9255    !
9260    !                    See if the user desires to define certain events.
9265    !
9270          Q$=""
9275          BEEP
9280          INPUT "Do wish to define events on the plot? (Y/N)",Q$
9285          IF Q$="Y" OR Q$="y" THEN
9290             BEEP
9295             INPUT "Enter the time of the first event.",Time1$
9300             BEEP
9305             INPUT "Enter the event occurring at the first time.",Event1$
9310             BEEP
9315             INPUT "Enter the time of the second event.",Time2$
9320             BEEP
9325             INPUT "Enter the event occurring at the second time.",Event2
$
9330             LORG 3
9335             MOVE 10,78
9340             LABEL Time1$&":  "&Event1$
9345             MOVE 10,60
9350             LABEL Time2$&":  "&Event2$
9355             PENUP
9360          END IF
9365          CSIZE 5.5
9370          LORG 4
9375          MOVE 50,0
9380          BEEP
9385          INPUT "Enter a Title for this plot",Title$
9390          LABEL Title$&"; "&Name$&"; "&Date$
9395          PENUP
9400          PEN 0
9405          GRAPHICS ON
9410          PAUSE
9415          GRAPHICS OFF
9420       ELSE
9425          Print_results(Avg_type$)
9430       END IF
9435    !
9440    !
9445       BEEP
```

```
9450        Q$=""
9455        INPUT "Do you wish to exit to the analysis routine? (Y/N)",Q$
9460      UNTIL Q$="Y" OR Q$="y"
9465      GINIT
9470    SUBEND
9475    !
9480    !
9485    !



9490    SUB View(Left,Right,Bottom,Top)
9495     !*********************************************************************
9500     !****                                                             ****
9505     !****  Routine Title:  "   View                                  ****
9510     !****                                                             ****
9515     !****  HP BASIC FILENAME: ANALYSIS                               ****
9520     !****                                                             ****
9525     !****  Department of Electrical and Computer Engineering,        ****
9530     !****  Kansas State University                                   ****
9535     !****                                                             ****
9540     !****  Revision     Date              Programer                  ****
9545     !****  --------     ------------      -----------------          ****
9550     !****    1.0        Sept. 10, 1984    Michael Masters            ****
9555     !****                                                             ****
9560     !*********************************************************************
9565     !****                                                             ****
9570     !****  Purpose:                                                   ****
9575     !****          This subroutine defines a viewport on the plotting****
9580     !****       device.  The total plotting area is made to be 100 by ****
9585     !****       100.  By using this routine to define the VIEWPORT the****
9590     !****       plots will have the same symetry independent of which ****
9595     !****       axis is longer.                                       ****
9600     !****                                                             ****
9605     !****                                                             ****
9610     !****                                                             ****
9615     !****                                                             ****
9620     !****  Routine(s) Called:   None                                  ****
9625     !****                                                             ****
9630     !****  Parameters passed from the calling routine                 ****
9635     !****                                                             ****
9640     !****       Left - The location of the left edge of the plotting  ****
9645     !****              window (0 to 100).                             ****
9650     !****                                                             ****
9655     !****       Right - The location of the right edge of the plotting****
9660     !****              window (0 to 100 and larger than Left).        ****
9665     !****                                                             ****
9670     !****       Bottom - The location of the bottom edge of the       ****
9675     !****              plotting window.                               ****
9680     !****                                                             ****
9685     !****       Top  - The location of the Top edge of the plotting   ****
9690     !****              window (0 to 100 and larger than Bottom).      ****
9695     !****                                                             ****
9700     !*********************************************************************
9705     !*********************************************************************
9710      IF RATIO>1 THEN
9715        VIEWPORT Left*RATIO,Right*RATIO,Bottom,Top
9720      ELSE
9725        VIEWPORT Left,Right,Bottom/RATIO,Top/RATIO
9730      END IF
9735    SUBEND
9740    !
9745    !
9750    !
```

```
9755  SUB Max_min(INTEGER Variable,Avg_type$,REAL Max,Min)
9760   !*********************************************************************
9765   !*********************************************************************
9770   !****                                                             ****
9775   !****  Routine Title:  Compute maximum and minimum.               ****
9780   !****                                                             ****
9785   !****  Revision     Date                Programer                ****
9790   !****  --------     ------------        -----------------        ****
9795   !****    1.0        Janurary 1, 1985    Michael Masters          ****
9800   !****                                                             ****
9805   !*********************************************************************
9810   !****                                                             ****
9815   !****  Purpose:  Determe the maximum and minum values of the resp-****
9820   !****    iratory parameter being plotted.                         ****
9825   !****                                                             ****
9830   !****  Parameters passed from the calling routine                 ****
9835   !****                                                             ****
9840   !****     Variable - The parameter for which the maximum and min.****
9845   !****           values are desired.                               ****
9850   !****                                                             ****
9855   !****     Avg_type$ - Contains the desired averaging technique.   ****
9860   !****                                                             ****
9865   !****                                                             ****
9870   !****  Parameters returned to the calling routine                 ****
9875   !****                                                             ****
9880   !****     Max - The maximum averaged value.                       ****
9885   !****                                                             ****
9890   !****     Min - The minimum averaged value.                       ****
9895   !****                                                             ****
9900   !****                                                             ****
9905   !****  Routine(s) Called:                                         ****
9910   !****     Avg_breath or Avg_window.                               ****
9915   !****                                                             ****
9920   !*********************************************************************
9925   !*********************************************************************
9930    COM /Anal_data8/ Resp_var(0:12,1:500)
9935    COM /Plot_resp_data1/ INTEGER Omit_bad,Num_breaths,No_b_avg
9940    COM /Plot_resp_data2/ Start_time,End_time,W_period,W_wdth
9945    INTEGER Br_pntr,Br_in_w_flag,Num_bad_breaths
9950   !
9955   !     Point to the first breath and initialize the maximum and minimum
9960   !     values.
9965    Br_pntr=1
9970    Min=999999999999.9
9975    Max=-999999999999.9
9980    IF Avg_type$="AVERAGE" THEN
9985   !
9990   !       Determine the maximum and minimum values of all the breaths or
9995   !       averaged breaths.
10000       REPEAT
10005          Avg_breath(Variable,Br_pntr,Time,Average,Num_bad_breaths)
10010          IF Average>Max THEN
10015             Max=Average
10020          END IF
10025          IF Average<Min THEN
10030             Min=Average
10035          END IF
10040          Br_pntr=Br_pntr+No_b_avg+Num_bad_breaths
10045       UNTIL Br_pntr>Num_breaths
10050    ELSE
10055       W_cntr=Start_time
10060       REPEAT
10065          Avg_window(Average,Br_in_w_flag,Variable,Br_pntr,W_cntr)
```

```
10070           IF Br_in_w_flag THEN
10075              IF Average>Max THEN
10080                 Max=Average
10085              END IF
10090              IF Average<Min THEN
10095                 Min=Average
10100              END IF
10105           END IF
10110           W_cntr=W_cntr+W_period
10115        UNTIL W_cntr>End_time
10120     END IF
10125  SUBEND
10130  !
10135  !
10140  !


10145  SUB Avg_breath(INTEGER Variable,Br_pntr,REAL Time,Average,INTEGER Num_b
ad_breaths)
10150    !*********************************************************************
10155    !*********************************************************************
10160    !****                                                             ****
10165    !****  Routine Title:  Average breaths                            ****
10170    !****                                                             ****
10175    !****  Revision     Date              Programer                   ****
10180    !****  --------     ------------      -----------------           ****
10185    !****    1.0        Janurary 1, 1985  Michael Masters             ****
10190    !****                                                             ****
10195    !*********************************************************************
10200    !****                                                             ****
10205    !****  Purpose:  To average the breath-by-breath values of consec-****
10210    !****     utive breaths.  Bad breaths may be ommitted in the      ****
10215    !****     average.                                                ****
10220    !****                                                             ****
10225    !****  Parameters passed from the calling routine                 ****
10230    !****                                                             ****
10235    !****     Variable - The respiratory parameter being averaged.    ****
10240    !****                                                             ****
10245    !****     Br_pntr - points to the initial breath to be averaged.  ****
10250    !****                                                             ****
10255    !****     Num_bad_breaths - The number of breaths skipped because****
10260    !****          they were a 'Bad Breath'                           ****
10265    !****                                                             ****
10270    !****  Parameters returned to the calling routine                 ****
10275    !****                                                             ****
10280    !****     Time - The time midway between the first breath         ****
10285    !****        averaged and the last breath averaged.               ****
10290    !****                                                             ****
10295    !****     Average-The average of the specified number of breaths  ****
10300    !****                                                             ****
10305    !****  Routine(s) Called:   None                                  ****
10310    !****                                                             ****
10315    !*********************************************************************
10320    !*********************************************************************
10325      COM /Anal_data3/ INTEGER Time_of_brth,O2_cons,Co2_prod,Insp_min_vent
10330      COM /Anal_data5/ INTEGER Breath_good,Resp_quotient,Form_fd$[1]
10335      COM /Anal_data5a/ INTEGER Frc_o2_cons,Frc_co2_prod,Volume_of_lung
10340      COM /Anal_data8/ Resp_var(0:12,1:500)
10345      COM /Plot_resp_data1/ INTEGER Omit_bad,Num_breaths,No_b_avg
10350      INTEGER J,Pointer
10355    !
10360    !
10365    !     Sum up the specified number of breaths to determine the average
10370    !     value.
```

```
10375     Average=0
10380     J=0
10385     Num_bad_breaths=0
10390     Pointer=Br_pntr
10395     WHILE J+1<=No_b_avg AND Pointer<=Num_breaths
10400     !
10405         IF Omit_bad THEN
10410           WHILE NOT (Resp_var(Breath_good,Pointer)) AND Pointer<=Num_bre
aths
10415               Num_bad_breaths=Num_bad_breaths+1
10420               Pointer=Br_pntr+J+Num_bad_breaths
10425           END WHILE
10430         END IF
10435     !
10440     !
10445         Average=Average+Resp_var(Variable,Pointer)
10450         J=J+1
10455         Pointer=Br_pntr+J+Num_bad_breaths
10460     END WHILE
10465     !
10470     !
10475     !                 Determine the average and the time which is centered
10480     !               about the number of breaths which are averaged.  The
10485     !               value of J is used rather then No_b_avg in the event
10490     !               the data array was exhausted before the specified number
10495     !               of breaths had been averaged.
10500     !
10505     Average=Average/J
10510     Time=(Resp_var(Time_of_brth,Br_pntr)+Resp_var(Time_of_brth,Pointer-1
))/2
10515   SUBEND
10520   !
10525   !
10530   !
10535   !
```

```
10540 SUB Avg_window(Average,INTEGER Br_in_w_flag,Variable,Br_pntr,REAL W_cnt
r)
10545  !*********************************************************************
10550  !*********************************************************************
10555  !****                                                             ****
10560  !****  Routine Title:  Average a time window.                     ****
10565  !****  .                                                          ****
10570  !****  Revision     Date                 Programer               ****
10575  !****  --------     ------------         -----------------       ****
10580  !****    1.0        Janurary 1, 1985     Michael Masters         ****
10585  !****                                                             ****
10590  !*********************************************************************
10595  !****                                                             ****
10600  !****  Purpose:  To compute the average of the breath-by-breath   ****
10605  !****     values occurring in a time window.                      ****
10610  !****                                                             ****
10615  !****  Parameters passed from the calling routine                  ****
10620  !****                                                             ****
10625  !****     Variable - The respiratory parameter being averaged.    ****
10630  !****                                                             ****
10635  !****     Br_pntr - points to the initial breath to be averaged.  ****
10640  !****                                                             ****
10645  !****     W_cntr - the time which the widow is centered around.   ****
10650  !****                                                             ****
10655  !****  Parameters returned to the calling routine                  ****
10660  !****                                                             ****
10665  !****     Average - The average of all breaths in the window.     ****
10670  !****                                                             ****
10675  !****     Br_in_w_flag - Set to true if the window did contain    ****
10680  !****          data and set to false if it did not.               ****
10685  !****                                                             ****
10690  !****  Routine(s) Called:  None                                   ****
10695  !****                                                             ****
10700  !*********************************************************************
10705  !*********************************************************************
10710    COM /Anal_data2/ INTEGER True,False
10715    COM /Anal_data3/ INTEGER Time_of_brth,O2_cons,Co2_prod,Insp_min_vent
10720    COM /Anal_data5/ INTEGER Breath_good,Resp_quotient,Form_fd$[1]
10725    COM /Anal_data5a/ INTEGER Frc_o2_cons,Frc_co2_prod,Volume_of_lung
10730    COM /Anal_data8/ Resp_var(0:12,1:500)
10735    COM /Plot_resp_data1/ INTEGER Omit_bad,Num_breaths,No_b_avg
10740    COM /Plot_resp_data2/ Start_time,End_time,W_period,W_wdth
10745    INTEGER Num_b_in_window,Fstbr_in_w_fnd
10750  !
10755  !         Find the first breath in the window.
10760    Left_edge_w=W_cntr-W_wdth/2
10765    REPEAT
10770       SELECT Resp_var(Time_of_brth,Br_pntr)
10775       CASE >=Left_edge_w
10780          ! Breath is right of the left window edge.
10785          IF Br_pntr=1 THEN
10790             Fstbr_in_w_fnd=True
10795          ELSE
10800             IF Resp_var(Time_of_brth,Br_pntr-1)<Left_edge_w THEN
10805             !  The first breath in the window has been found
10810                Fstbr_in_w_fnd=True
10815             ELSE
10820                Br_pntr=Br_pntr-1
10825             END IF
10830          END IF
10835       CASE <Left_edge_w
10840          ! Breath is left of the left window edge.
10845          Br_pntr=Br_pntr+1
10850          Fstbr_in_w_fnd=False
10855       END SELECT
```

```
10860     UNTIL Fstbr_in_w_fnd OR Br_pntr>Num_breaths
10865   !
10870   !
10875     Num_b_in_window=0
10880     Average=0
10885     WHILE Resp_var(Time_of_brth,Br_pntr)<W_cntr+W_wdth/2 AND Br_pntr<=Nu
m_breaths
10890   !
10895   !              If the breath is not good and bad breaths are to be
10900   !              omitted do not include this breath, other wise include it.
10905         IF Omit_bad AND NOT (Resp_var(Breath_good,Br_pntr)) THEN
10910           Br_pntr=Br_pntr+1
10915         ELSE
10920           Average=Average+Resp_var(Variable,Br_pntr)
10925           Br_pntr=Br_pntr+1
10930           Num_b_in_window=Num_b_in_window+1
10935         END IF
10940     END WHILE
10945     IF Num_b_in_window=0 THEN
10950         Br_in_w_flag=False
10955     ELSE
10960         Br_in_w_flag=True
10965         Average=Average/Num_b_in_window
10970     END IF
10975   SUBEND
10980   !
10985   !
10990   !
```

```
10995 SUB Make_axes(INTEGER Variable,Corr_flag,REAL Min,Max)
11000 !*************************************************************************
11005 !*************************************************************************
11010 !****                                                                  ****
11015 !****  Routine Title:  Make axes                                       ****
11020 !****                                                                  ****
11025 !****  Revision     Date                Programer                     ****
11030 !****  --------     ------------        -----------------             ****
11035 !****    1.0        Janurary 1, 1985    Michael Masters               ****
11040 !****                                                                  ****
11045 !*************************************************************************
11050 !****                                                                  ****
11055 !****  Purpose:   Used to draw the axes on the plotting device and****
11060 !****       label plots for the breath-by-breath data.                 ****
11065 !****                                                                  ****
11070 !****  Parameters passed from the calling routine                      ****
11075 !****                                                                  ****
11080 !****       Variable - The prameter to be plotted; O2_cons,            ****
11085 !****            Co2_prod,etc.                                         ****
11090 !****                                                                  ****
11095 !****       Corr_flag - Set to true if temperature and viscosity       ****
11100 !****            corrections were used in analyzing the data.          ****
11105 !****                                                                  ****
11110 !****       Min - The minimum value of the parameter to be plotted****
11115 !****                                                                  ****
11120 !****       Max - The maximum value of the parameter to be plotted****
11125 !****                                                                  ****
11130 !****                                                                  ****
11135 !****  Routine(s) Called:    None                                      ****
11140 !****                                                                  ****
11145 !*************************************************************************
11150 !*************************************************************************
11155 !
11160   COM /Anal_data3/ INTEGER Time_of_brth,O2_cons,Co2_prod,Insp_min_vent
11165   COM /Anal_data4/ INTEGER Expr_min_vent,V_tid_insp,V_tid_expr,Resp_fr
eq
11170   COM /Anal_data5/ INTEGER Breath_good,Resp_quotient,Form_fd$[1]
11175   COM /Anal_data5a/ INTEGER Frc_o2_cons,Frc_co2_prod,Volume_of_lung
11180   COM /Plot_resp_data2/ Start_time,End_time,W_period,W_wdth
11185   DIM Heading$(1:11)[30]
11190 !
11195 !
11200 !                    Define the individual plot titles.
11205 !
11210   Heading$(O2_cons)="Minute O2 Consumption"
11215   Heading$(Co2_prod)="Minute CO2 Production"
11220   Heading$(Frc_o2_cons)="Aveolar Minute O2 Consumption"
11225   Heading$(Frc_co2_prod)="Aveolar Minute CO2 Production"
11230   Heading$(Insp_min_vent)="Inspiratory Minute Ventilation"
11235   Heading$(Expr_min_vent)="Expiratory Minute Ventilation"
11240   Heading$(V_tid_insp)="Inspiratory Tidal Volume"
11245   Heading$(V_tid_expr)="Expiratory Tidal Volume"
11250   Heading$(Volume_of_lung)="Lung Volume"
11255   Heading$(Resp_freq)="Respiratory Frequency"
11260   Heading$(Resp_quotient)="Respiratory Quotient"
11265 !
11270 !
11275 !                         Define the plotting area and make the axes.
11280 !
11285   Top=Max+.2*(Max-Min)
11290   Bottom=Min-.26*(Max-Min)
11295   Left=Start_time-.1*(End_time-Start_time)
11300   Right=End_time+.05*(End_time-Start_time)
11305   FRAME
11310   WINDOW Left,Right,Bottom,Top
```

```
11315    CLIP Start_time,End_time,Min,Max
11320    Xtick=(Start_time-End_time)/10
11325    Ytick=(Max-Min)/4
11330    AXES Xtick,Ytick,Start_time,Min
11335  !
11340  !                        Label the time axis.
11345    CLIP Left,Right,Bottom,Top
11350    LORG 6
11355    CSIZE 3
11360    FOR X=Start_time TO 1.05*End_time STEP (End_time-Start_time)/5
11365       MOVE X,Min-(Max-Min)/75
11370       LABEL DROUND(X,3)
11375    NEXT X
11380    LORG 4
11385    MOVE (Start_time+End_time)/2,Bottom
11390    LABEL "Data collection Time (seconds)"
11395  !
11400  !                        Label the axis of the dependent variable
11405    LORG 8
11410    FOR Y=Min TO 1.05*Max STEP (Max-Min)/4
11415       MOVE Start_time,Y
11420       LABEL DROUND(Y,3)
11425    NEXT Y
11430    IF Variable=Resp_quotient THEN
11435       MOVE Start_time,1
11440       DRAW End_time,1
11445    END IF
11450  !
11455  !              Put the apporiate units on the dependent axis.
11460    LORG 3
11465    MOVE Left,Top
11470    SELECT Variable
11475    CASE =Resp_freq
11480       LABEL " BrPM"
11485    CASE =V_tid_insp,=V_tid_expr,=Volume_of_lung
11490       IF Corr_flag THEN
11495          LABEL " Liters (BTPS)"
11500       ELSE
11505          LABEL " Liters"
11510       END IF
11515    CASE =Insp_min_vent,=Expr_min_vent
11520       IF Corr_flag THEN
11525          LABEL " L/Min (BTPS)"
11530       ELSE
11535          LABEL " L/Min"
11540       END IF
11545    CASE =O2_cons,=Co2_prod,=Frc_o2_cons,=Frc_co2_prod
11550       IF Corr_flag THEN
11555          LABEL " L/Min (STPD)"
11560       ELSE
11565          LABEL " L/Min"
11570       END IF
11575    CASE =Resp_quotient
11580         ! Respiratory quotient is unitless.
11585    END SELECT
11590  !
11595  !
11600  !                    Label the plot.
11605  !
11610    MOVE End_time,Top
11615    LORG 9
11620    CSIZE 4
11625    LABEL Heading$(Variable)&" .vs. Time"
11630    CLIP Start_time,End_time,Min,Max
11635  SUBEND
```

```
11640 !
11645 !
11650 !
11655 !


11660 SUB Plot_values(INTEGER Variable,Avg_type$)
11665  !*************************************************************************
11670  !*************************************************************************
11675  !****                                                                 ****
11680  !****  Routine Title:  Plot breath-by-breath data.                    ****
11685  !****                                                                 ****
11690  !****  Revision     Date                 Programer                   ****
11695  !****  --------     ------------         -----------------           ****
11700  !****    1.0        Janurary 1, 1985     Michael Masters             ****
11705  !****                                                                 ****
11710  !*************************************************************************
11715  !****                                                                 ****
11720  !****  Purpose:   To plot the breath by breath data using the         ****
11725  !****       specified averaging technique.                            ****
11730  !****                                                                 ****
11735  !****  Parameters passed from the calling routine                     ****
11740  !****                                                                 ****
11745  !****       Variable - The parameter to be plotted.                   ****
11750  !****                                                                 ****
11755  !****       Avg_type$ - Contains the desired averaging technique.     ****
11760  !****                                                                 ****
11765  !****  Parameters returned to the calling routine                     ****
11770  !****               None                                              ****
11775  !****                                                                 ****
11780  !****  Routine(s) Called:                                             ****
11785  !****         Avg_breath or Avg_window                                ****
11790  !****                                                                 ****
11795  !*************************************************************************
11800  !*************************************************************************
11805  !
11810    COM /Anal_data8/ Resp_var(0:12,1:500)
11815    COM /Plot_resp_data1/ INTEGER Omit_bad,Num_breaths,No_b_avg
11820    COM /Plot_resp_data2/ Start_time,End_time,W_period,W_wdth
11825    INTEGER Br_pntr,Br_in_w_flag,Num_bad_breaths
11830    LORG 5
11835    CSIZE 1.5
11840    Br_pntr=1
11845    IF Avg_type$="AVERAGE" THEN
11850       REPEAT
11855          Avg_breath(Variable,Br_pntr,Time,Average,Num_bad_breaths)
11860          MOVE Time,Average
11865          LABEL "O"
11870          Br_pntr=Br_pntr+No_b_avg+Num_bad_breaths
11875       UNTIL Br_pntr>Num_breaths
11880    ELSE
11885       W_cntr=Start_time
11890       REPEAT
11895          Avg_window(Average,Br_in_w_flag,Variable,Br_pntr,W_cntr)
11900          IF Br_in_w_flag THEN
11905             MOVE W_cntr,Average
11910             LABEL "O"
11915          END IF
11920          W_cntr=W_cntr+W_period
11925       UNTIL W_cntr>End_time
11930    END IF
11935 SUBEND
11940 !
11945 !
```

```
11950 SUB Print_results(Avg_type$)
11955  !*********************************************************************
11960  !*********************************************************************
11965  !****                                                             ****
11970  !****  Routine Title:  Print breath-by-breath data.               ****
11975  !****                                                             ****
11980  !****  Revision     Date                 Programer               ****
11985  !****  --------     ------------         -----------------       ****
11990  !****    1.0        Janurary 1, 1985     Michael Masters         ****
11995  !****                                                             ****
12000  !*********************************************************************
12005  !****                                                             ****
12010  !****  Purpose:   To print the breath by breath data using the    ****
12015  !****       specified averaging technique.                        ****
12020  !****                                                             ****
12025  !****  Parameters passed from the calling routine                 ****
12030  !****                                                             ****
12035  !****       Avg_type$ - Contains the desired averaging technique.  ****
12040  !****                                                             ****
12045  !****  Parameters returned to the calling routine                 ****
12050  !****              None                                           ****
12055  !****                                                             ****
12060  !****  Routine(s) Called:                                         ****
12065  !****          Avg_breath or Avg_window                           ****
12070  !****                                                             ****
12075  !*********************************************************************
12080  !*********************************************************************
12085    COM /Anal_data3/ INTEGER Time_of_brth,O2_cons,Co2_prod,Insp_min_vent
12090    COM /Anal_data4/ INTEGER Expr_min_vent,V_tid_insp,V_tid_expr,Resp_fr
eq
12095    COM /Anal_data5/ INTEGER Breath_good,Resp_quotient,Formfd$[1]
12100    COM /Anal_data5a/ INTEGER Frc_o2_cons,Frc_co2_prod,Volume_of_lung
12105    COM /Anal_data6/ INTEGER Corr_flag,REAL Room_fh2o,T
12110    COM /Anal_data8/ Resp_var(0:12,1:500)
12115    COM /Anal_data9/ Name$[25],Comment$[46]
12120    COM /Cal_data1/ Cal$[10],Date$[25]
12125    COM /Plot_resp_data1/ INTEGER Omit_bad,Num_breaths,No_b_avg
12130    COM /Plot_resp_data2/ Start_time,End_time,W_period,W_wdth
12135    DIM Values(1:11)
12140    INTEGER Br_pntr,Br_index,Br_in_w_flag,I,Some_rs_to_big,Num_bad_breat
hs
12145    Some_rs_to_big=(1=0)
12150  !
12155  !
12160    PRINTER IS 9
12165    PRINT USING "K,K,/";"Date:     ",Date$
12170    PRINT USING "K,K,/";"Comment: ",Comment$
12175    PRINT USING "K,K,/";"Subject identifier: ",Name$
12180    IF Avg_type$="AVERAGE" THEN
12185       IF No_b_avg=1 THEN
12190          PRINT USING "K";"The individual breath-by-breath values are sh
own."
12195       ELSE
12200          PRINT USING "K,K,/";"The number of breaths averaged is ",No_b_
avg
12205          PRINT USING "#,K,K";"The breaths are combined by AVERAGING.  "
,No_b_avg
12210          PRINT USING "K,2/";" successive breaths are averaged together.
"
12215       END IF
12220    ELSE
12225       PRINT USING "K,DDDD.D,K,/";"The window period is ",W_period," sec
```

```
onds"
12230       PRINT USING "K,DDDD.D,K,/";"The window width is  ",W_wdth," secon
ds"
12235       PRINT USING "K,K,K";"The starting time is ",Start_time," seconds"
12240       PRINT USING "K,K,K,/";"The ending time is   ",End_time," seconds"
12245       PRINT USING "#,K";"The breaths are combined by WINDOWING.  The br
eaths occurring"
12250       PRINT USING "K,K,K";" in a window ",W_wdth," second(s) wide are a
veraged."
12255       PRINT USING "#,K,K";"The center of the window is moved ",W_period
12260       PRINT USING "K";" seconds between successive averages.  The cente
r of the"
12265       PRINT USING "#,K,K";"first window is ",Start_time
12270       PRINT USING "K,K,K,2/";" seconds and the ending time is ",End_tim
e," seconds."
12275    END IF
12280    IF Omit_bad THEN
12285       PRINT "   * BAD BREATHS OMITTED *"
12290       PRINT USING "#,K";"The bad breaths are not included in the averag
ing technique."
12295       PRINT USING "K";"  Bad breaths are those breaths with"
12300       PRINT USING "#,K";"an inspiriratory volume or an expiratory"
12305       PRINT USING "K,2/";" volume of less than 0.4 Liters."
12310    END IF
12315    PRINT USING "#,K";"The R' value is equal to the averaged CO2 produce
d value "
12320    PRINT USING "K";"divided by the averaged O2 consumed value.  It is"
12325    PRINT USING "#,K";"a better estimate of the true R value for the bre
aths "
12330    PRINT USING "K,2/";"averaged than the average of the individual R va
lues."
12335    PRINT "                                  | Alveolar | Alveolar |"
12340    PRINT USING "#,K";"|       |    O2    |   CO2    |    O2    |   CO2
  | Resp.   |"
12345    PRINT USING "K";" Averaged |  Ventilation  | Tidal Volume  | Resp.
|         |"
12350    PRINT USING "#,K";"| Time | Consumed | Produced | Consumed | Produce
d | Quotient |"
12355    PRINT USING "K";"CO2/O2 :R'| Insp. | Expr. | Insp. | Expr. | Freq.
| Lung    |"
12360    PRINT USING "#,K";"| (sec)| (L/min)  | (L/min)  | (L/min)  | (L/min)  |
12365    PRINT USING "K";"          |(L/min)|(L/min)|  (L)   |  (L)   |(br/min)
| Vol. (L)|"
12370    IF Corr_flag THEN
12375       PRINT USING "#,K";"|       | (STPD)  | (STPD)  | (STPD)  | (ST
PD)  |         |"
12380       PRINT USING "K";"          | (BTPS)| (BTPS)| (BTPS)| (BTPS)|
  | (BTPS)  |"
12385    END IF
12390    PRINT USING "#,K";"|_______|_________|_________|_________|_________
__|_________|"
12395    PRINT USING "K";"_________|_______|_______|_______|_______|________
|_________|"
12400    Br_pntr=1
12405    IF Avg_type$="AVERAGE" THEN
12410       REPEAT
12415          FOR I=O2_cons TO Resp_quotient
12420             Avg_breath(I,Br_pntr,Time,Average,Num_bad_breaths)
12425             Values(I)=Average
12430          NEXT I
12435          PRINT USING "#,K,4D.D,K";"|",Time,"|"
12440          GOSUB Print_line
12445          Br_pntr=Br_pntr+No_b_avg+Num_bad_breaths
12450       UNTIL Br_pntr>Num_breaths
12455    ELSE
```

```
12460         W_cntr=Start_time
12465         REPEAT
12470           FOR I=O2_cons TO Resp_quotient
12475             Br_index=Br_pntr
12480             Avg_window(Average,Br_in_w_flag,I,Br_index,W_cntr)
12485             Values(I)=Average
12490           NEXT I
12495           Br_pntr=Br_index
12500           PRINT USING "#,K,DDDD.D,K";"|",W_cntr,"|"
12505           IF Br_in_w_flag THEN
12510             GOSUB Print_line
12515           ELSE
12520             PRINT "     ****     The window did not contain any breaths
12525           END IF
12530           W_cntr=W_cntr+W_period
12535         UNTIL W_cntr>End_time
12540       END IF
12545       PRINT USING "#,K";"|_______|____________|____________|____________|_____
__|_____________|"
12550       PRINT USING "K";"____________|________|________|________|________|_____
|________|"
12555       IF Some_rs_to_big THEN
12560         PRINT
12565         PRINT USING "#,K";"
12570         PRINT USING "K";"**.*** The averaged R value was larger than 99.9
99"
12575       END IF
12580       PRINT USING "#,2/"
12585       PRINTER IS 1
12590       SUBEXIT
12595 Print_line:   !
12600       PRINT USING "#,K,DD.3D,K";"  ",Values(O2_cons),"  |"
12605       PRINT USING "#,K,DD.3D,K";"  ",Values(Co2_prod),"  |"
12610       PRINT USING "#,K,DD.3D,K";"  ",Values(Frc_o2_cons),"  |"
12615       PRINT USING "#,K,DD.3D,K";"  ",Values(Frc_co2_prod),"  |"
12620 !
12625       IF Values(Resp_quotient)<100 THEN
12630         PRINT USING "#,K,DD.3D,K";"  ",Values(Resp_quotient),"  |"
12635       ELSE
12640         PRINT USING "#,K";"  **.***  |"
12645         Some_rs_to_big=(1=1)
12650       END IF
12655 !
12660       IF ABS(Values(Co2_prod)/Values(O2_cons))<100 THEN
12665         PRINT USING "#,K,DD.3D,K";"  ",ABS(Values(Co2_prod)/Values(O2_con
s)),"  |"
12670       ELSE
12675         PRINT USING "#,K";"  **.***  |"
12680         Some_rs_to_big=(1=1)
12685       END IF
12690 !
12695       PRINT USING "#,K,3D.DD,K";" ",Values(Insp_min_vent),"|"
12700       PRINT USING "#,K,3D.DD,K";" ",Values(Expr_min_vent),"|"
12705       PRINT USING "#,K,DD.DD,K";" ",Values(V_tid_insp),"  |"
12710       PRINT USING "#,K,DD.DD,K";" ",Values(V_tid_expr),"  |"
12715       PRINT USING "#,K,3D.2D,K";" ",Values(Resp_freq),"  |"
12720       PRINT USING "#,K,DD.3D,K";" ",Values(Volume_of_lung),"  |"
12725       PRINT
12730       RETURN
12735 SUBEND
12740 !
12745 !
12750 !
```

```
12755 SUB Expanded_dump
12760 !*************************************************************************
12765 !*************************************************************************
12770 !****                                                                  ****
12775 !****  Function Title:  Expanded graphics dump.                        ****
12780 !****                                                                  ****
12785 !****  Revision     Date                Programer                     ****
12790 !****  --------     ------------        -----------------             ****
12795 !****    1.0        March 1,1985        Michael Masters               ****
12800 !****                                                                  ****
12805 !*************************************************************************
12810 !****                                                                  ****
12815 !****  Purpose:  To dump the screen to the printer and expand the      ****
12820 !****     size by 4 to 1.  Simply use the command                      ****
12825 !****                                                                  ****
12830 !****                          CALL Expanded_dump                      ****
12835 !****     This command is keyboard executable.                         ****
12840 !****                                                                  ****
12845 !****  Parameters passed from the calling routine                       ****
12850 !****      None                                                        ****
12855 !****                                                                  ****
12860 !*************************************************************************
12865 !*************************************************************************
  12870   GRAPHICS OFF
  12875   PRINT "ADJUST THE THERMAL PAPER TO THE TOP OF THE PAGE"
  12880   PRINT "AND PRESS CONTINUE."
  12885   BEEP
  12890   PAUSE
  12895   PRINT CHR$(12)
  12900   GRAPHICS ON
  12905   DUMP DEVICE IS 701,EXPANDED  ! Set to the expanded mode
12910 !
12915 !                 Adjust the right side of the paper so that the figure
12920 !                 will fit on the page
  12925   PRINTER IS 701
  12930   PRINT CHR$(27)&"*r70xA"
  12935   PRINTER IS 1
  12940   DUMP GRAPHICS
12945 !
12950 !         Readjust the right side of the paper so that the normal
12955 !         dumps are centered on the page.
  12960   PRINTER IS 701
  12965   PRINT CHR$(27)&"*r175xA"
  12970   DUMP DEVICE IS 701
  12975   PRINTER IS 1
  12980   BEEP
12985 SUBEND
```

A7.7 STOREHR: a Basic Program Which Stores the Heart Rate Values

The STOREHR program is used to store the subjects experimental heart rate data in a file compatible with Sprick's plotting program [16]. The data are entered by the operator. Once entered the operator has the opportunity to review the data and make any necessary corrections.

A7.7.1 The Main Routine of STOREHR

The main routine provides the operator prompts necessary to obtain the heart rate data. The data are stored in the file format specified by Sprick [16]. This format is necessary in order to use Sprick's plotting routine.

Common Blocks Used by the Subroutine

None

Variable definitions

Heart_rate - a real array which contains the heart rate data.

I - a real variable used in conjunction with the variable J as a pointer into the Heat_rate array.

J - a real variable used in conjunction with the variable I as a pointer into the Heat_rate array.

Name$ - a string variable equal to an operator defined file name. The heart rate data are stored in the file specified by Name$.

Pointer - a real variable used as a pointer into the Heart_rate array.

Q$ - a string variable used to obtain the operator's response to a question.

True - a real variable used as the Boolean value of true.

Value - a real variable used as a pointer into the Heart_rate array.

A7.7.2 Load autost Subroutine

The Load_autost subroutine loads the AUTOST (A7.3) program into memory. The AUTOST program begins execution once loaded.

Common Blocks Used by the Subroutine

None

Variables passed to the procedures

None

Variables returned by the procedures

None

A7.7.3 Load_plot Subroutine

The Load_plot subroutine loads the PLOT program developed by Sprick [16] into memory. The PLOT program begins execution once loaded.

Common Blocks Used by the Subroutine

None

Variables passed to the procedures

None

Variables returned by the procedures

None

A7.7.4 Options Subroutine

The Options subroutine presents the operator with the options which may be taken once the data have been entered and stored. These options give the operator the ability to load the AUTOST program, load the PLOT program, or store additional heart rate data.

Common Blocks Used by the Subroutine

None

Variables passed to the procedures

None

Variables returned by the procedures

None

A7.7.5 STOREHR Program Listing

```
10    !*************************************************************************
20    !*************************************************************************
30    !****                                                                  ***
40    !****  Routine Title:      Store Heart Rate                            ***
50    !****                                                                  ***
60    !****  HP BASIC FILENAME:  STOREHR                                     ***
70    !****                                                                  ***
80    !****  Department of Electrical and Computer Engineering               ***
90    !****  Kansas State University                                         ***
100   !****                                                                  ***
110   !****  Revision     Date                Programer                      ***
120   !****  --------     ------------        -----------------              ***
130   !****    1.0        August 20, 1984     Michael Masters                ***
140   !****                                                                  ***
150   !*************************************************************************
160   !****                                                                  ***
170   !****  Purpose:  To store the heart rate of the subject as recorded***
180   !****            by the heart rate monitor.  The heart rate is       ***
190   !****            for each 30 second interval.                        ***
200   !****                                                                  ***
210   !*************************************************************************
220   GCLEAR
230   CALL Options
240   PRINT CHR$(12)
250   PRINT "*************************************************"
260   PRINT "*                                                *"
270   PRINT "*                                                *"
280   PRINT "*            Save the heart Rate                 *"
290   PRINT "*                                                *"
300   PRINT "*                                                *"
310   PRINT "*************************************************"
320   DIM Heart_rate(1:50)
330   PRINT
340   PRINT
350   PRINT "Enter the file name.  The file will be stored on"
360   PRINT "the data disk in a format which can be read by"
370   PRINT "the PLOT routine."
380   BEEP
390   INPUT "Enter the name of the Heart Rate file",Name$
400   True=(1=1)
410   Pointer=0
420   PRINT CHR$(12)
430   PRINT
440   PRINT
450   PRINT "Enter the heart rate for each 30 second interval."
460   PRINT
470   PRINT "If you have entered all heart rate values;"
480   PRINT "    Enter a negative number in response to the"
490   PRINT "   'Enter the Heart Rate' prompt."
500   PRINT
510   PRINT "You will be given an opportunity to correct any"
520   PRINT "mistakes at a latter time."
530   REPEAT
540      Pointer=Pointer+1
550      PRINT TABXY(1,24),"                                                "
560      PRINT TABXY(1,24),"Enter the heart rate for the time, t=";(Pointer-
1)*.5;" min."
570      BEEP
580      INPUT Heart_rate(Pointer)
590   UNTIL Heart_rate(Pointer)<0
600   Pointer=Pointer-1
610   FOR J=1 TO Pointer STEP 10
```

```
620 Print_values:True=(1=1)
630       I=0
640       PRINT CHR$(12)
650       PRINT
660       PRINT
670       PRINT "Value      Time      Heart Rate"
680       WHILE True
690          IF I+J<=Pointer AND I<10 THEN
700                PRINT USING "X,DD,7X,DD.D,8X,DDDD";I+J;(I+J-
1)*.5;Heart_rate(I+J)
710          ELSE
720             True=(1=0)
730          END IF
740          I=I+1
750       END WHILE
760       Q$="N"
770       BEEP
780 Change:INPUT "Do you wish to change any of these values? (Y/N)",Q$
790       IF Q$<>"N" AND Q$<>"n" AND Q$<>"y" AND Q$<>"Y" THEN Change
800       IF Q$="y" OR Q$="Y" THEN
810          Value=0
820          BEEP
830          INPUT "Which value do you wish to change?",Value
840          IF Value<>0 THEN
850             BEEP
860             INPUT "Enter the new value.",Heart_rate(Value)
870             GOTO Print_values
880          END IF
890       END IF
900       Q$=""
910    NEXT J
920     !
930     !
940     !  Store the heart rate values.
950     !
960    PRINT CHR$(12)
970    PRINT
980    PRINT
990    PRINT "Storing the values"
1000   CREATE BDAT Name$&":HP9895,700",Pointer*8/256+1
1010   ASSIGN @Path1 TO Name$&":HP9895,700"
1020   ON END @Path1 GOTO 1050
1030   OUTPUT @Path1;Pointer
1040   OUTPUT @Path1;Heart_rate(*)
1050    !
1060    !
1070   PRINT CHR$(12)
1080   PRINT
1090   PRINT "You may obtain a plot of these values using"
1100   PRINT "Sprick's PLOT routine.  To obtain a plot answer"
1110   PRINT "Y to the next question"
1120   Q$="N"
1130   BEEP
1140   INPUT "Would you like to plot the values? (Y/N)",Q$
1150   SELECT Q$
1160   CASE ="Y",="y"
1170       PRINT CHR$(12)
1180       PRINT
1190       PRINT
1200       PRINT "When instructed by PLOT to enter the X axis label,"
1210       PRINT "enter TIME using capital letters"
1220       PRINT
1230       PRINT "Press CONTINUE when you are ready to proceed."
1240       PAUSE
1250       PRINT CHR$(12)
```

```
1260      PRINT
1270      PRINT "     Now loading PLOT"
1280      CALL Load_plot
1290   CASE ="N",="n"
1300      PRINT CHR$(12)
1310      PRINT
1320      PRINT "Program run complete"
1330   CASE ELSE
1340      GOTO 1120
1350   END SELECT
1360   CALL Options
1370   END
1380   SUB Options
1390 Keys:  !
1400      ON KEY 1 LABEL "PLOT" CALL Load_plot
1410      ON KEY 4 LABEL "STOREHR" GOTO End
1420      ON KEY 9 LABEL "EXIT" CALL Load_autost
1430      GOTO Keys
1440         !
1450         !
1460 End:SUBEND
1470   SUB Load_plot
1480      OFF KEY
1490      MASS STORAGE IS ":HP9895,700,0"
1500      LOAD "PLOT:HP9895,702,3",1
1510   SUBEND
1520   SUB Load_autost
1530      OFF KEY
1540      LOAD "AUTOST:INTERNAL"
1550   SUBEND
```

A7.8 TXDCR_VALS: a Basic Program Which Computes the Component Values of the Temperature Transducer

The TXDCR_VALS program aids in the determination of the component values of the temperature transducer. The operator defines the input temperature range and the desired output voltage range. Also, the operator must define a value for the V_r, R_s, R_f, and R_c. With the range information and the defined components the necessary values of R_g, R_t, and R_r are computed. The operator can experiment with different ranges and values of the defined components to see the effect on the calculated components. The program may also generate an output voltage versus temperature table.

A7.8.1 The Main Routine of TXDCR_VALS

The main routine of TXDCR_VALS provides the main control of the program. Most operator input is accomplished via the various subroutines.

Variable definitions

Rc - a real variable equal to the value of the R_c.

Rf - a real variable equal to the value of the R_f.

Rs - a real variable equal to the value of the R_s.

Tambient - a real variable equal to the average ambient temperature.

Tmax - a real variable equal to the maximum temperature the transducer is to measure.

Tmin - a real variable equal to the minimum temperatuare the transducer is to measure.

Vmax - a real variable equal to the output voltage corresponding to a temperature equal to Tmax.

Vmin - a real variable equal to the output voltage correspondint to a temperature equal to Tmin.

Vr - a real variable equal to the reference voltage.

A7.8.2 Enter_rfrc Subroutine

The Enter_rfrc subroutine is used to prompt the operator for the value of the resistors R_f and R_c.

Variables passed to the procedures

None

Variables returned by the procedures

Rc - real.

Rf - real.

Variable definitions

Rc - a real variable equal to the operator defined value of R_c.

Rf - a real variable equal to the operator defined value of R_f.

A7.8.3 Enter_rs Subroutine

The Enter_rs subroutine is used to prompt the operator for the value of the resistor R_s.

Variables passed to the procedures

None

Variables returned by the procedures

Rs - real.

Variable definitions

Rs - a real variable equal to the operator defined value of R_s.

A7.8.4 Print Subroutine

The Print subroutine prints the values to transducer components. The subroutine computes the values of all components which have not been defined.

Variables passed to the procedures

None

Variables returned by the procedures

None

Variable definitions

K1 - a real variable equal to the gain of the pre-amplifier.

K2 - a real variable equal to the gain of the instrumentation amplifier.

Rc - a real variable equal to the operator defined value of

R_c.

Rf - a real variable equal to the operator defined value of R_f.

Rg - a real variable equal to the calculated value of R_g.

Rr - a real variable equal to the calcualted value of R_r.

Rs - a real variable equal to the operator defined value of R_s.

Rt - a real variable equal to the calcualted value of R_t.

T_vcomp0 - a real variable equal to the temperature corresponding to a compensation voltage of zero.

Tambient - a real variable equal to the average ambient temperature.

Tmax - a real variable equal to the maximum temperature the transducer is to measure.

Tmin - a real variable equal to the minimum temperatuare the transducer is to measure.

Tot_gain - a real variable equal to the total gain of the transducer.

Vcomp_ta - a real variable equal to the compensation voltage correspondiung to the ambient temperature.

Va - a real variable equal to the second order coeficent of the second order function relating the thermocouple voltage to the sensing junction temperature.

Vb - a real variable equal to the first order coeficent of the second order function relating the thermocouple voltage to the sensing junction temperature.

Vmax - a real variable equal to the output voltage corresponding to a temperature equal to Tmax.

Vmin - a real variable equal to the output voltage correspondint to a temperature equal to Tmin.

Vr - a real variable equal to the reference voltage.

Vr_over_rr - a real variable equal to the refence voltage divided by the reference resistor (R_r).

A7.8.5 Print_results Subroutine

The Print_results subroutine prints a table of the transducer output voltage versus the thermocouple temperature. The subroutine uses the calculated component values to determine the output voltage.

Variables passed to the procedures

None

Variables returned by the procedures

None

Variable definitions

A - a real variable used as a pointer into the Printer array.

I - a real variable used as a loop counter.

J - a real variable used as a loop counter.

Printer - a real array of three elements. The elements contain the specification of the CRT, the thermal printer, and the DECwriter printer.

Rr - a real variable equal to the calcualted value of R_r.

Rs - a real variable equal to the operator defined value of R_s.

Rt - a real variable equal to the calcualted value of R_t.

Tambient - a real variable equal to the average ambient temperature.

Temp - a real variable equal to the temperature value for which the output voltage is computed.

Tmax - a real variable equal to the maximum temperature the transducer is to measure.

Tmin - a real variable equal to the minimum temperatuare the transducer is to measure.

Tot_gain - a real variable equal to the total gain of the transducer.

Vout - a real variable equal to the output voltage corresponding to a temperature equal to Temp.

Vr - a real variable equal to the reference voltage.

A7.8.6 Txdcr_range Subroutine

The Txdcr_range subroutine prompts the operator for the temperature range the transducer will measure, the output voltage corresponding to the temperature range, and the ambient temperature.

Variables passed to the procedures

None

Variables returned by the procedures

Tambient - real.
Tmax - real.
Tmin - real.
Vmax - real.
Vmin - real.

Variable definitions

Tambient - a real variable equal to the average ambient temperature.
Tmax - a real variable equal to the maximum temperature the transducer is to measure.
Tmin - a real variable equal to the minimum temperatuare the transducer is to measure.
Vmax - a real variable equal to the output voltage corresponding to a temperature equal to Tmax.
Vmin - a real variable equal to the output voltage correspondint to a temperature equal to Tmin.

A7.8.7 Vr Subroutine

The Vr subroutine is used to prompt the operator for the value of the voltage reference V_r.

Variables passed to the procedures

None

Variables returned by the procedures

Vr - real.

Variable definitions

Vr - a real variable equal to the operator defined value of V_r.

A7.8.8 Vtc Function

The Vtc function calculates the thermocouple voltage corresponding to the specified sensing junction temperature and a specified reference junction temperature.

Variables passed to the procedures

Tj - real.

Tref - real.

Variable definitions

Tj - a real variable equal to the sensing junction temperature.

Tref - a real variable equal to the reference junction temperature.

Va - a real variable equal to the second order coeficent of the second order function relating the thermocouple voltage to the sensing junction temperature.

Vb - a real variable equal to the first order coeficent of the second order function relating the thermocouple voltage to the sensing junction temperature.

Vc - a real variable equal to the zero order coeficent of the second order function relating the thermocouple voltage to the sensing junction temperature.

Vtc - a real variable equal to the thermocouple voltage corresponding to a sensing junction temperature of Tj and a reference junction temperature of Tref.

A7.8.9 TXDCR_VALS Program Listing

```
10   !*************************************************************************
20   !*************************************************************************
30   !***                                                                    ***
40   !***  Routine Title:      TXDCR_VALS                                    ***
50   !***                                                                    ***
60   !***  HP BASIC FILENAME:  TXDCR_VALS                                    ***
70   !***                                                                    ***
80   !***  Department of Electrical and Computer Engineering                 ***
90   !***  Kansas State University                                           ***
100  !***                                                                    ***
110  !***  Revision     Date                Programer                        ***
120  !***  --------     ------------        -----------------                ***
130  !***    1.0        June 4, 1983        Michael Masters                  ***
140  !***    2.0        January 30, 1985    Micheal Masters                  ***
150  !***                                                                    ***
160  !*************************************************************************
170  !***                                                                    ***
180  !***  Purpose:                                                          ***
190  !***       To calculate the component values of the respiratory         ***
200  !***     temperature transducer.                                        ***
210  !***                                                                    ***
220  !*************************************************************************
230   COM Tmax,Tmin,Vmax,Vmin,Tambient,Rs,Rg,Rc,Rf,Vr,Rt,Rr,Tot_gain
240   PRINTER IS 1
250   CALL Txdcr_range(Tmax,Tmin,Vmax,Vmin,Tambient)
260   CALL Enter_rfrc(Rf,Rc)
270   CALL Enter_rs(Rs)
280   CALL Vr(Vr)
290   CALL Print
300 Define_keys:!
310 !
320 !                  ***** Define the special function keys *****
330   ON KEY 2 LABEL "Print" GOTO Print_results
340   ON KEY 5 LABEL "Range" GOTO Txdcr_range
350   ON KEY 6 LABEL "Rf & Rc" GOTO Enter_rfrc
360   ON KEY 7 LABEL "Rs" GOTO Enter_rs
370   ON KEY 8 LABEL "Vr" GOTO Vr
380   ON KEY 9 LABEL "EXIT" GOTO Quit
390 Do_nothing:GOTO Do_nothing
400   !
410   !
420 Txdcr_range:!
430   !-----------------------------------------------------------------------
440   !  Define the temperature range to be measured, the minimum and maximum
450   !  voltage output, and the average ambient temperature.
460   OFF KEY
470   CALL Txdcr_range(Tmax,Tmin,Vmax,Vmin,Tambient)
480   CALL Print
490   GOTO Define_keys
500   !
510   !
520 Enter_rfrc: !
530   !-----------------------------------------------------------------------
540   ! Enter the values of the resistors Rf and Rc which develop the gain K1.
550   !
560   OFF KEY
570   CALL Enter_rfrc(Rf,Rc)
580   CALL Print
590   GOTO Define_keys
600 !
610 !
620 Enter_rs:!
```

```
630  !-------------------------------------------------------------------------
640  !  Enter the value of the resistor Rs.
650  !
660   OFF KEY
670   CALL Enter_rs(Rs)
680   CALL Print
690   GOTO Define_keys
700 !
710 !
720 Vr:!
730  !-------------------------------------------------------------------------
740  !  Enter the value of the referenc voltage.
750  !
760   OFF KEY
770   CALL Vr(Vr)
780   CALL Print
790   GOTO Define_keys
800 !
810 !
820 Print_results:!
830  !-------------------------------------------------------------------------
840  !  Enter the value of the referenc voltage.
850  !
860   OFF KEY
870   PRINTER IS 1
880   CALL Print_results
890   GOTO Define_keys
900 !
910 !
920 !
930 Quit:  !
940  !-------------------------------------------------------------------------
950  !  Exit the program.
960  !
970   BEEP
980   PRINT "PROGRAM RUN COMPLETE"
990   END
1000 !
1010 !
1020  SUB Txdcr_range(Tmax,Tmin,Vmax,Vmin,Tambient)
1030 !
1040 ! define the temperature range, output voltage, and ambient temperature.
1050 !
1060       INPUT "ENTER THE MINIMUM TEMPERATURE TO BE MEASURED",Tmin
1070       INPUT "ENTER THE MAXIMUM TEMPERATURE TO BE MEASURED",Tmax
1080       INPUT "ENTER VOLTAGE CORRESPONDING TO Tmin",Vmin
1090       INPUT "ENTER VOLTAGE CORRESPONDING TO Tmax",Vmax
1100       INPUT "ENTER THE AVERAGE AMBIENT TEMPERATURE",Tambient
1110  SUBEND
1120  !
1130  !
1140  !
1150  SUB Enter_rfrc(Rf,Rc)
1160       INPUT "ENTER THE VALUE OF Rf",Rf
1170       INPUT "ENTER THE VALUE OF Rc",Rc
1180  SUBEND
1190  !
1200  !
1210  !
1220  SUB Enter_rs(Rs)
1230       INPUT "ENTER THE VALUE OF Rs (close to 100k)",Rs
1240  SUBEND
1250  SUB Vr(Vr)
1260       INPUT "ENTER THE VALUE OF THE REFERENCE VOLTAGE",Vr
1270  SUBEND
```

```
1280 !
1290 !
1300 !
1310  SUB Print
1320 !
1330 !      **** Compute the component values and print them. ****
1340 !
1350       COM Tmax,Tmin,Vmax,Vmin,Tambient,Rs,Rg,Rc,Rf,Vr,Rt,Rr,Tot_gain
1360       PRINT CHR$(12) ! Clear the screen.
1370 !
1380       Va=4.585E-8     ! The coeficients of the second order fit of the
1390       Vb=5.865E-5     ! thermocouple voltage to the temperature.
1400 !
1410 !         Compute the total gain, the pre-amp gain, AD521 gain, and Rg.
1420       Tot_gain=(Vmax-Vmin)/(FNVtc(Tmax,Tambient)-FNVtc(Tmin,Tambient))
1430       K1=1+2*Rf/Rc
1440       K2=Tot_gain/K1
1450       Rg=Rs/K2
1460 !
1470 !
1480 !         Compute the value of the feedback resistor in the reference
1490 !         junction compensator and then the reference voltage resistor.
1500       Rt=Tot_gain*(2*Va*Tambient+Vb)*(1.E+6)
1510       Vcomp_ta=Vmin-Tot_gain*FNVtc(Tmin,Tambient)
1520       Vr_over_rr=(Tambient+273.2)*(1.0E-6)-Vcomp_ta/Rt
1530       Rr=Vr/Vr_over_rr
1540       T_vcomp0=Vr_over_rr*(1.0E+6)-273.2
1550 !
1560 !
1570 !              Display the results.
1580       PRINT USING "2(K,DDD.DD)";"TEMPERATURE RANGE: ",Tmin," TO ",Tmax
1590       PRINT USING "/,4(K)";"     Vo(",Tmin,",0) = ",Vmin
1600       PRINT USING "4(K)";"     Vo(",Tmax,",0) = ",Vmax
1610       PRINT USING "2(K)";"Tambient = ",Tambient
1620       PRINT USING "/,K,DDDDDD.DD";"TOTAL GAIN = ",Tot_gain
1630       PRINT USING "K";"          K1 = ",K1
1640       PRINT USING "K";"          K2 = ",K2
1650       PRINT USING "2(K,MD.DDDE)";"Rc = ",Rc,"    Rf = ",Rf
1660       PRINT USING "2(K,MD.DDDE)";"Rs = ",Rs,"    Rg = ",Rg
1670       PRINT USING "/,K,DDD.DDDD";"VOLTAGE REFERENCE = ",Vr
1680       PRINT USING "2(K,MD.DDDE)";"Rr = ",Rr,"    Rt = ",Rt
1690       PRINT USING "/,K,MDD.DD";"Tref(Vcomp=0) = ",T_vcomp0
1700  !
1710  !          If Rr is a negative resistance the reference voltage must be
1720  !          changed in polarity.
1730       IF Rr<0 THEN
1740          BEEP
1750          PRINT USING "/,K";"THE SIGN OF Vr MUST BE CHANGED IN ORDER TO"
1760          PRINT "HAVE A POSITIVE Rr!"
1770       END IF
1780  SUBEND
1790  SUB Print_results
1800       COM Tmax,Tmin,Vmax,Vmin,Tambient,Rs,Rg,Rc,Rf,Vr,Rt,Rr,Tot_gain
1810       DIM Printer(1:3)
1820       Printer(1)=1
1830       Printer(2)=701
1840       Printer(3)=9
1850       PRINT CHR$(12)
1860       PRINT "1.  CRT"
1870       PRINT "2.  HP Thermal printer."
1880       PRINT "3.  DecWriter."
1890       A=1
1900       INPUT "WHICH PRINTER DO YOU WANT? 1,2,3",A
1910       PRINTER IS Printer(A)
1920       IF A=2 THEN
```

```
1930            PRINT CHR$(27)&"&a1L"
1940        END IF
1950        CALL Print
1960        PRINT USING "3/"
1970    !
1980    !       Print the transducer output voltages.
1990        PRINT "        ****  The output voltage of the reference junction
compensator.  ****"
2000        PRINT
2010        PRINT "       |";
2020        FOR I=0 TO .9 STEP .1
2030            PRINT USING "#,K,.D,K";"  ",I,"  .|"
2040        NEXT I
2050        PRINT USING "/,#,""       |"",10(""------|"")"
2060        FOR I=INT(Tambient-5) TO INT(Tambient+15)
2070            PRINT USING "/,#,MDD,K";I,"---|"
2080            FOR J=0 TO .9 STEP .1
2090                Temp=I+J
2100                Vout=((Temp+273.2)*1.0E-6-Vr/Rr)*Rt
2110                IF Vout>-10 THEN
2120                    PRINT USING "#,DD.DDD,K";Vout,"|"
2130                ELSE
2140                    PRINT USING "#,K";"**.***|"
2150                END IF
2160            NEXT J
2170        NEXT I
2180        PRINT USING "/,""       -"",10(""-------"")"
2190    !
2200    !       Print the transducer output voltages.
2210        PRINT USING "3/"
2220        PRINT "         ****  The respiratory temperature transducer output
voltage.  ****"
2230        PRINT
2240        PRINT "       |";
2250        FOR I=0 TO .9 STEP .1
2260            PRINT USING "#,K,.D,K";"  ",I,"  |"
2270        NEXT I
2280        PRINT USING "/,#,""       |"",10(""------|"")"
2290        FOR I=INT(Tmin-5) TO INT(Tmax+5)
2300            PRINT USING "/,#,MDD,K";I,"---|"
2310            FOR J=0 TO .9 STEP .1
2320                Temp=I+J
2330                  Vout=Tot_gain*FNVtc(Temp,Tambient)+((Tambient+273.2)*1.E-6-
Vr/Rr)*Rt
2340                IF Vout>-10 THEN
2350                    PRINT USING "#,DD.DDD,K";Vout,"|"
2360                ELSE
2370                    PRINT USING "#,K";"**.***|"
2380                END IF
2390            NEXT J
2400        NEXT I
2410        PRINT USING "/,""       -"",10(""-------"")"
2420        PRINTER IS 1
2430    SUBEND
2440    !
2450    !
2460    !
2470    DEF FNVtc(Tj,Tref)
2480        Va=4.585E-8
2490        Vb=5.865E-5
2500        Vc=1.9807E-7
2510        Vtc=Va*Tj^2+Vb*Tj+Vc-(Va*Tref^2+Vb*Tref+Vc)
2520        RETURN Vtc
2530    FNEND
```

A Computer-Based Respiratory Measurement System and a Temperature Transducer for Monitoring Respiratory Flow Temperature

by
MICHAEL HARRY MASTERS
B.S., Kansas State University, 1982

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the requirements
for the degree
MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Abstract

Breath-by-breath respiratory function studies require an accurate means of computing inspiratory and expiratory volumes, from which tbe ventilation values are computed. Most important in these calculations is the attainment of a valid respiratory flow signal under known temperature and pressure conditions. A computer based respiratory measurement system (CBRMS) having the ability to monitor respiratory flow, fractional O_2, and fractional CO_2 bas been enbanced with a respiratory flow temperature transducer and with gas viscosity calculations. These enhancements allow for more accurate volume calculations. Also the display of the results has been enhanced.

A rapidly responding temperature transducer based on a cbromel-constantan (type E), 0.001 in. diameter, thermocouple is designed and incorporated into the CBRMS for monitoring respiratory flow temperature. The instantaneous flow gas viscosity is calculated with this temperature signal, tbe fractional O_2 signal, the fractional CO_2 signal, the ambient conditions, and the calculated water vapor signal. This viscosity is used to compute respiratory flow from the pneumotacbograph differential pressure signal, a signal previously assumed to be directly proportional to the flow, in accordance witb Poiseuille's law. The temperature signal, water vapor signal, and the barometric pressure are used to convert this flow to STPD conditions. The STPD flow is multiplied by the dry gas concentrations. These products are integrated to obtain tbe inspiratory and expiratory gas volumes. The difference of these inspiratory and expiratory volumes yields O_2 consumed and CO_2 produced. Inspiratory and expiratory tidal volumes are obtained by integrating the STPD flow and scaling to BTPS conditions. Otber ventilation values are computed from tbese volumes and tbe respiratory cycle time. These breath-by-breath values may be displayed as tbe respiratory data are being analyzed and in a variety of formats post-analysis. The mean respiratory variables for the entire analyzed data are displayed post-analysis.

The averages of a specified number of consecutive breath-by-breatb values or the average of tbe breath-by-breath values appearing in a specified time window may then be displayed grapbically or printed in a tabular format. The window averages can be computed for specific points in time and compared to window averages of another subject or anotber exercise regimen.