A COMPUTER-BASED INSTRUMENTATION SYSTEM FOR MEASUREMENT OF BREATH-BY-BREATH OXYGEN CONSUMPTION AND CARBON DIOXIDE PRODUCTION IN EXERCISING HUMANS

by

LOREN EUGENE RIBLETT, JR.

B.S., Kansas State University, 1983

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

1984

Approved by:

Richard R. Gallagher
Major Professor

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION	1
II.	GENERAL SYSTEM DESCRIPTION	2
III.	INSTRUMENTATION FOR CALIBRATION AND DATA ACQUISITION	4
	3.1 Instruments and Interconnects	4
	3.2 Calibration Hardware	7
IV.	SYSTEM SOFTWARE	13
	4.1 Overview	13
	4.2 Calibration Software	16
	4.3 Data Acquisition Software	17
	4.4 File Manipulation Software	17
	4.5 Data Analysis and Display Software	19
V.	EXPERIMENTAL METHODS	26
	5.1 Subject Selection and Preparation (Breath-by-breath)	26
	5.2 System Calibration (Breath-by-breath)	26
	5.3 Data Collection and Analysis (Breath-by-breath)	28
	5.4 System Comparison with Bag Collection Technique	29
VI.	EXPERIMENTAL RESULTS	31
	6.1 Presentation of Results	31
VII.	DISCUSSION OF RESULTS	49
VIII.	CONCLUSIONS	52
IX.	IMPENDING RESEARCH	56
X.	BIBLIOGRAPHY	58
	ACKNOWLEDGEMENTS	60
	APPENDIX I. Mass Storage Management	A1.1
	APPENDIX II. DAM Offset and Gain Adjustment	A2.1
	APPENDIX III. System Calibration Procedure	A3.1
	APPENDIX IV. System Collection of Breath-by-breath Respiratory Data	A4.1

Page

APPENDIX V. PASCAL to BASIC File Conversion of Calibration Data A5.1

APPENDIX VI. PASCAL to BASIC File Conversion of Respiratory Data A6.1

APPENDIX VII. System Analysis of Breath-by-breath Respiratory Data A7.1

APPENDIX VIII. CAP.CODE A8.1

APPENDIX IX. DAP.CODE A9.1

APPENDIX X. BIT.CODE A10.1

APPENDIX XI. AUTOST A11.1

APPENDIX XII. CAPCRUNCH A12.1

APPENDIX XIII. DAPCRUNCH A13.1

APPENDIX XIV. ANALYSIS A14.1

ABSTRACT

LIST OF FIGURES

FIGURE		Page
3.1	BREATH-BY-BREATH SYSTEM PICTORIAL	5
3.2	RESEARCH LAB LAYOUT	9
3.3	FLOW SIGNAL CALIBRATION APPARATUS	10
3.4	FRACTIONAL GAS CONCENTRATION CALIBRATION APPARATUS	12
4.1	SEQUENCE FOR COLLECTING BREATH-BY-BREATH RESPIRATORY INFORMATION	14
4.2	PASCAL/BASIC FILE CONVERSION	15
4.3	DAM CONTROL SCHEME	18
4.4	DETERMINATION OF VARIABLE TIME DELAY	21
4.5	NON-TIME ALIGNED RESPIRATORY SIGNALS	23
4.6	TIME ALIGNED RESPIRATORY SIGNALS	23
4.7	HARD COPY OUTPUT OF BREATH-BY-BREATH DATA	24
4.8	HARD COPY OUTPUT OF AVERAGE RESPIRATORY DATA	25
6.1	TIME DOMAIN RESPIRATORY PLOT (50 WATTS)	33
6.2	TIME DOMAIN RESPIRATORY PLOT (100 WATTS)	34
6.3	BREATH-BY-BREATH O2 CONSUMPTION	35
6.4	BREATH-BY-BREATH O2 CONSUMPTION	36
6.5	BREATH-BY-BREATH EXPIRATORY MINUTE VOLUME	37
6.6	BREATH-BY-BREATH CO2 PRODUCTION	38
6.7	BREATH-BY-BREATH O2 CONSUMPTION	39
6.8	BAG EXPIRATORY MINUTE VOLUME	40
6.9	BAG CO2 PRODUCTION	41
6.10	BAG O2 CONSUMPTION	42

FIGURE		Page
6.11	HEATED BREATH-BY-BREATH EXPIRATORY MINUTE VOLUME	43
6.12	HEATED BREATH-BY-BREATH CO2 PRODUCTION	44
6.13	HEATED BREATH-BY-BREATH O2 CONSUMPTION	45
6.14	BREATH-BY-BREATH AND BAG EXPIRATORY MINUTE VOLUME	46
6.15	BREATH-BY-BREATH AND BAG CO2 PRODUCTION MEANS	47
6.16	BREATH-BY-BREATH AND BAG O2 CONSUMPTION MEANS	48
A1.1	RESEARCH LAB LAYOUT	A1.3
A2.1	DAM OFFSET AND GAIN ADJUSTMENTS LOCATIONS	A2.2

I. INTRODUCTION

Since the primary function of the lung is to oxygenate venous blood and remove carbon dioxide from the same blood, an automated quantitative technique designed to measure this gas exchange is very desirable [1].

In the past, the researcher and clinician have had to depend on procedures utilizing oxygen content of the blood and the oxygen concentration of end-expired gas collected in bags. These methods allowed for average O_2 consumption and CO_2 production and said nothing about the transient or breath-by-breath changes involved in the respiratory process [2,3].

Since the advent of computer-controlled instrumentation and rapidly responding gas analyzers, several researchers have begun looking at the dynamics involved in the respiratory process as well as the problems inherent in breath-by-breath measurements [4-11]. Most notable is the research by Creel [11], in which techniques were developed to study exercise stressed calves on a breath-by-breath basis.

The studies described in this thesis evolved in an attempt to not only convert Creel's work from the calf to human subjects, but also to reorganize both the developed software and hardware so a more accurate and precise system would result. This thesis presents an overview of the system, the instrumentation used for calibration and data acquisition, the system software, the experimental methods used for system verification, and the results of those experiments. In addition to these topics, all system operating procedures are described and complete software documentation is included in the appendices.

II. GENERAL SYSTEM DESCRIPTION

For the human breath-by-breath respiratory system, five physiological signals are monitored, namely fractional CO_2 and O_2 concentrations, respiratory flow, respiratory flow temperature, and body temperature. With the exception of body temperature, the mentioned signals (which are converted to electrical analog signals using various transducers) are converted to their digital representations using a custom built Data Acquisition Module (DAM) and passed onto the memory of a desktop computer, the controller for the entire instrumentation system. Using the digitized information as well as calibration factors determined from specially designed calibration procedures, various gas volumes (air, O_2, and CO_2) during both inspiration and expiration can be determined. By allowing for data windowing, any section (window) of the collected data can be analyzed.

Conversion of these volumes to both BTPS and STPD conditions is also possible provided the data analysis routine is supplied not only with the digitized respiratory signal (which allows for point-by-point temperature correction) but also with the barometric pressure, relative humidity, and the subject's body temperature.

Tabular results are organized so that a single row of the data lists values for the calculated gas volumes and respiratory times corresponding to the given breath. Also, average and time-dependent cardio-pulmonary variables for the analyzed window are displayed following the breath-by-breath results. These quantities include inspiratory and expiratory minute volumes, inspiratory and expiratory tidal volumes, respiratory frequency, mean inspiratory and expiratory O_2 and CO_2 volumes, mean O_2 consumed and CO_2 produced per breath, rates of O_2 consumption and CO_2 production, respiratory quotient, and total times for inspiration, expiration, and respiration.

Graphical representation of the windowed data includes four time domain plots for the fractional CO_2 and O_2 signals, the respiratory flow signal, and respiratory temperature signal.

III. INSTRUMENTATION FOR CALIBRATION AND DATA ACQUISITION

3.1 Instruments and Interconnects

In order to successfully measure respiratory gases (CO_2 and O_2) in exercising humans on a breath-by-breath basis a significant amount of computer monitored equipment is necessary. Following is a brief description of the basic instruments and how these instruments are interconnected so as to function in the proper manner. Refer to Figure 3.1 for a system pictorial.

The heart of the computer controlled system is the HP9826 desktop computer. By utilizing a PASCAL operating system during the data acquisition process (faster data acquisition is possible using a compiled language such as PASCAL) and a BASIC operating system for data analysis, a well controlled respiratory measurement system is possible. The HP9826 contains memory in excess of 0.5 Mbytes of RAM allowing for real time data collection of eight minutes at a 50 Hz sampling frequency.

Acquired data is stored using two mass storage devices, an HP9895A 8" flexible disk memory and an HP9134A hard disk memory. These two mass storage devices were selected because of high speed data storage capabilities (the hard disk) and the possibility of data portability (the 8" flexible disk). Connection between the HP9826 computer and these mass storage devices is accomplished via an HPIB interface at select code 7 [12].

A DECwriter II serial printer is currently connected to the HP9826 computer via an RS-232C link (at select code 9) to provide hard copy output of analysis results and program listings. Vast system improvement could be realized in analysis speed if a

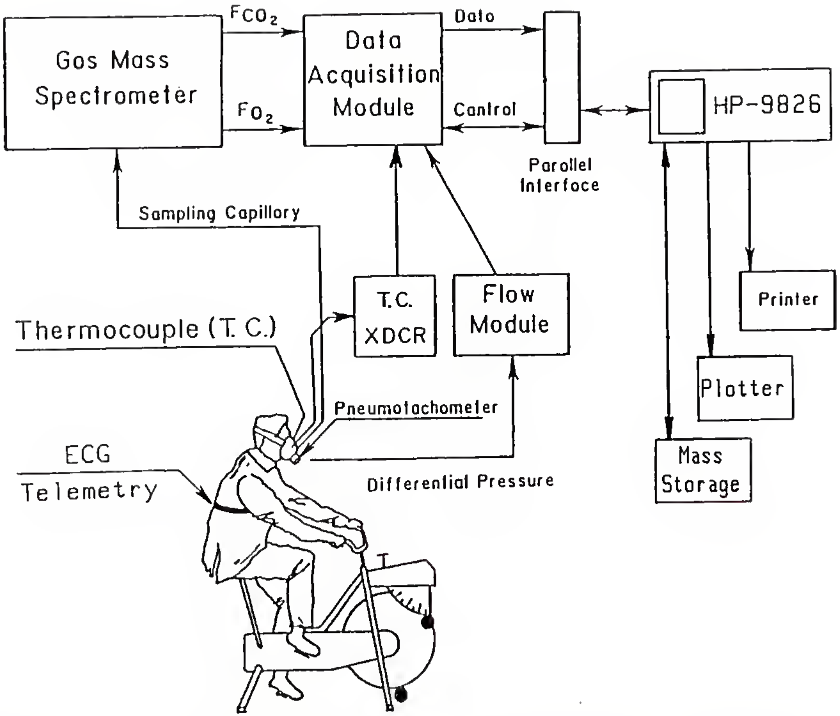


FIGURE 3.1 BREATH-BY-BREATH SYSTEM PICTORIAL

state-of-the-art printer were used in place of the DECwriter II.

Gateno's Data Acquisition Module (DAM) [13] is connected to the HP9826 using a GPIO 16-bit parallel interface. This interface provides the signal pathways for controlling and monitoring the DAM. The reader is referred to Appendix IX for details of the DAM control and status words. The GPIO interface is located at select code 12.

Four analog input channels of Gateno's DAM are currently controlled by HP9826 hardware and software, two of these inputs (DAM channels A and B) being fractional gas concentrations of CO_2 and O_2. These signals are supplied by a Gas Mass Spectrometer (GMS) (Perkin-Elmer 1100 medical gas analyzer) [14]. These analog signals supplied by the GMS are proportional to the instantaneous fractional gas concentration. Associated with the GMS, however, is a delay time associated with the gas transport time of the gas sampling capillary and electrical response of the GMS itself. These delays are corrected by using system software described in Chapter IV.

Respiratory gas flow is measured using a No. 2 Fleisch PneumoTachoMeter (PTM) and a Godart PneumoTachoGraph (PTG) [15]. The PTM produces a differential pressure across its screen proportional to the gas flow through the screen. The PTG is pneumatically connected across the PTM screen and converts this differential pressure to an electrical analog signal which is sampled at DAM input channel C. This method of inspiratory and expiratory flow is known as a closed circuit technique as mentioned by Creel [11].

A Hans Rudolf 2-way breathing respiratory mask model series

#7900 [16] is used to secure the PTM and GMS sampling capillary to the subject's face, in the region of the mouth and nose. The masks were designed not only to accommodate the PTMs used but also to insure leakage and dead space volumes are minimized.

Channel D of the DAM is reserved for measurement of the respiratory gas temperature. A complete description of the temperature apparatus may be obtained from Masters [17]. Measurement of the respiratory temperature allows for point-by-point temperature correction of respiratory gas volumes from one ambient temperature to another.

A Monark bicycle ergometer provides the desired work load for the human exercise. According to Astrand [18], bicycling is a very suitable work form, since, among other things, at a given load, (submaximal), it demands about the same energy output, whether the subject is trained or out of condition, elite bicyclist or unfamiliar with the sport. With the ergometer presently being used, work loads from 0 watts (rest) to 500+ watts are possible.

Figure 3.2 shows the overall system layout as it appears in the Bioengineering Research Laboratory. This particular organization was chosen because of the short coaxial cable runs necessary in the analog portion of the computer-controlled system. It is felt that many of the calibration and operation problems that existed in Creel's calf studies [11] have now been eliminated through proper equipment organization.

3.2 <u>Calibration Hardware</u>

Additional hardware is needed in order to calibrate the mentioned instrumentation. To calibrate the Fleisch/Godart flow

A. WORK TABLE
B. DECwriter II PRINTER
C. TEMPERATURE CALIBRATION WATER BATHS
D. ZERO SUPPRESSION BOX
E. PERKIN-ELMER GASS MASS SPECTROMETER
F. TEKTRONIX D41 OSCILLOSCOPE
G. HP2673A THERMAL PRINTER
H. HP9826 COMPUTER
I. HP9134A HARD DISK MEMORY
J. HP9895A 8" FLEXIBLE DISK MEMORY
K. TEKTRONIX TM MODULE
L. GPIO INTERFACE TESTER
M. HP9872C PLOTTER
N. DATA ACQUISITION MODULE
O. GODART PNEUMOTACHOMETER
P. HARVARD RESPIRATOR AND FITTINGS
Q. ERGOMETER REVOLUTION COUNTER
R. METRONOME
S. BOTTLED GAS (12.9% O_2-7% CO_2)
T. BICYCLE ERGOMETER
U. PNEUMOTACH HEATER
V. HEART RATE MONITOR
W. SUPPORT FOR MASKS, FLEISCH HEADS, AND METEOROLOGICAL BALLOON
X. SPIROMETER

KEY FOR FIGURE 3.2

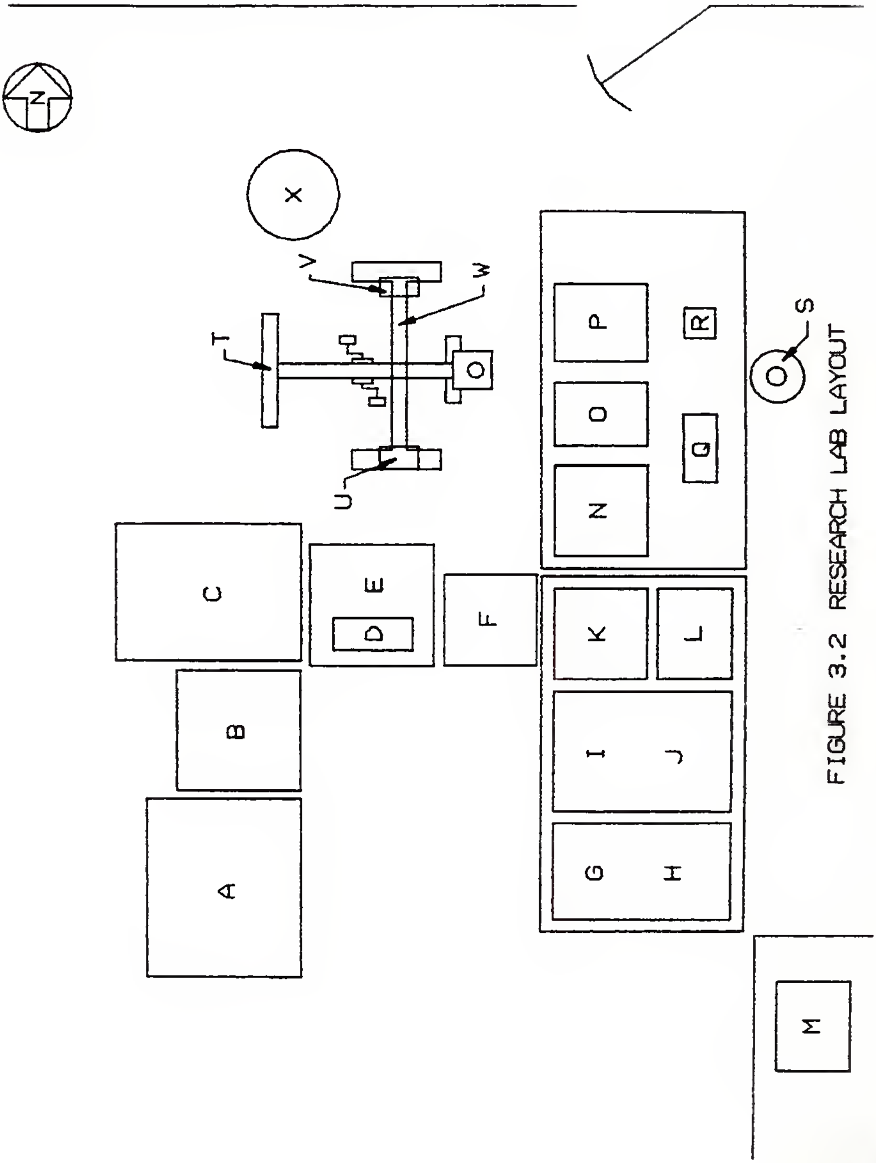


FIGURE 3.2 RESEARCH LAB LAYOUT

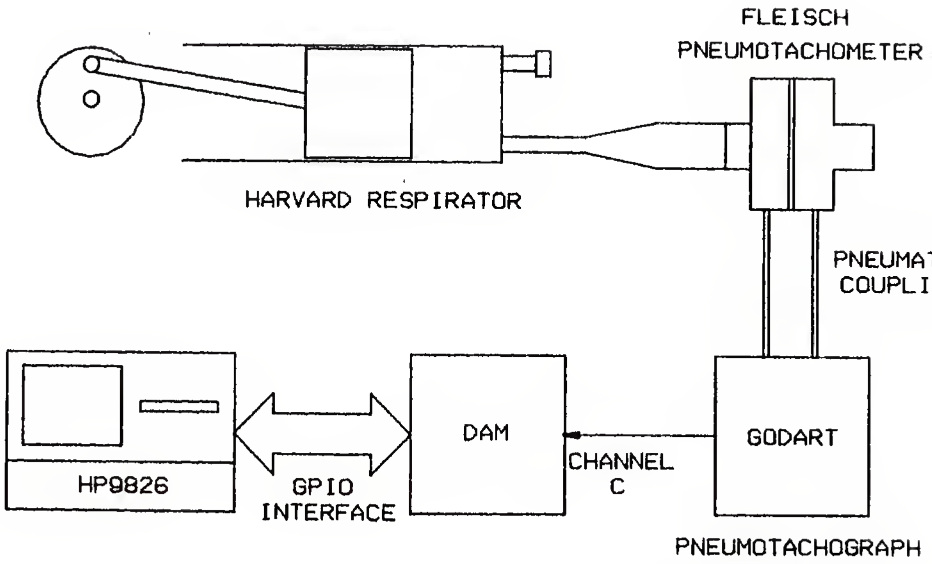


FIGURE 3.3 FLOW SIGNAL CALIBRATION APPARATUS

signal, the apparata arrangement as shown in Figure 3.3 is used. A Harvard Respirator of known stroke volume (see Creel's work [11] for calibration of respirator) is used to force known inspiratory and expiratory volumes of air through the PTM. By integrating the inspiratory and expiratory flow signals as measured by the DAM and comparing these sums with the known cylinder volume, flow calibration factors can be determined (see Chapter V for more details).

Calibration of the GMS requires the apparata as shown in Figure 3.4. A 12.9% O_2, 7% CO_2 (balance nitrogen) gas cylinder supplies the calibration point for the minimum O_2 level and the maximum CO_2 level. Adjustment of the zero suppression box as well as DAM gain adjustments can be made so the analog signals produced by the GMS fall within the operating range of the DAM (see Appendix II for complete details).

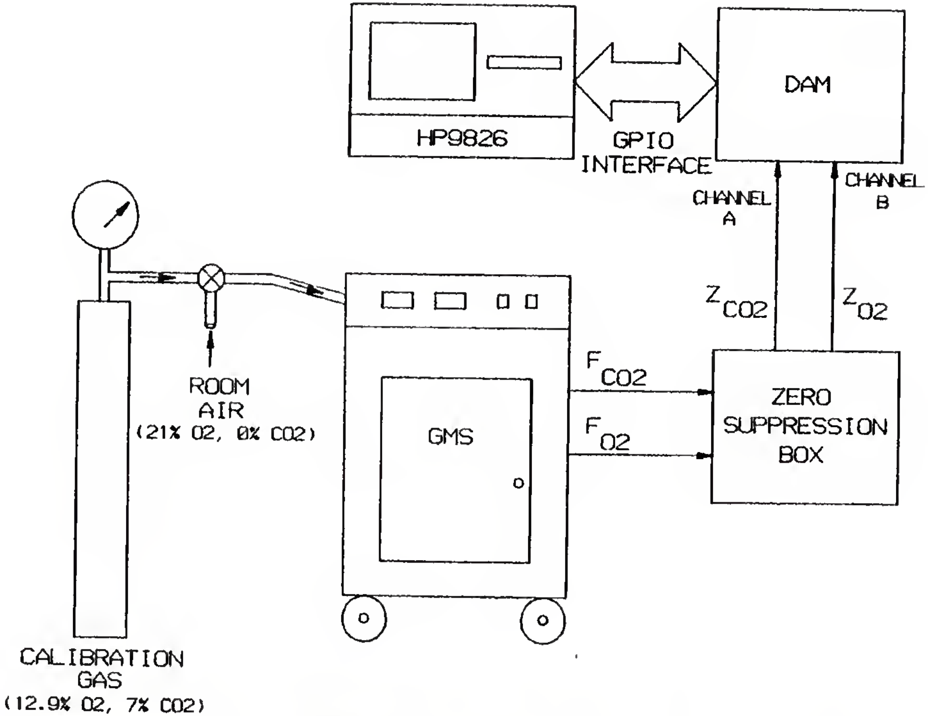


FIGURE 3.4 FRACTIONAL GAS CONCENTRATION CALIBRATION APPARATUS

IV. SYSTEM SOFTWARE

4.1 Overview

The respiratory system software was designed around the HP9826 desktop computer. It (the software) is a combination of PASCAL and BASIC programs for calibrating, acquiring data, and analyzing the data. The two programming languages were chosen because of the speed necessary to control the DAM (PASCAL) and the ability to alter analysis parameters quickly and with little or no effort (BASIC). Figure 4.1 shows the typical sequence for collecting breath-by-breath respiratory information with the system as it presently exists and the software necessary to perform the mentioned tasks.

With the use of two different systems, file compatability between the PASCAL and BASIC operating systems is of great importance. Figure 4.2 demonstrates how the issue of file compatability is resolved. Because ASCII files are the only compatable file types between the two operation systems, creation of the rather large ASCII files is necessary. ASCII calibration or data files are created by the PASCAL programs CAP.CODE and DAP.CODE. The dashed line in Figure 4.2 indicates the separate operating systems.

To reduce mass storage usage and the time necessary to load the mentioned files into HP9826 memory from disk, the ASCII files are crunched (converted to Binary DATa files (BDAT)). Once converted to BDAT files, the respiratory calibration and data files can be analyzed by ANALYSIS (the breath-by-breath analysis routine). What follows is a description of the various software segments depicted in Figure 4.2.

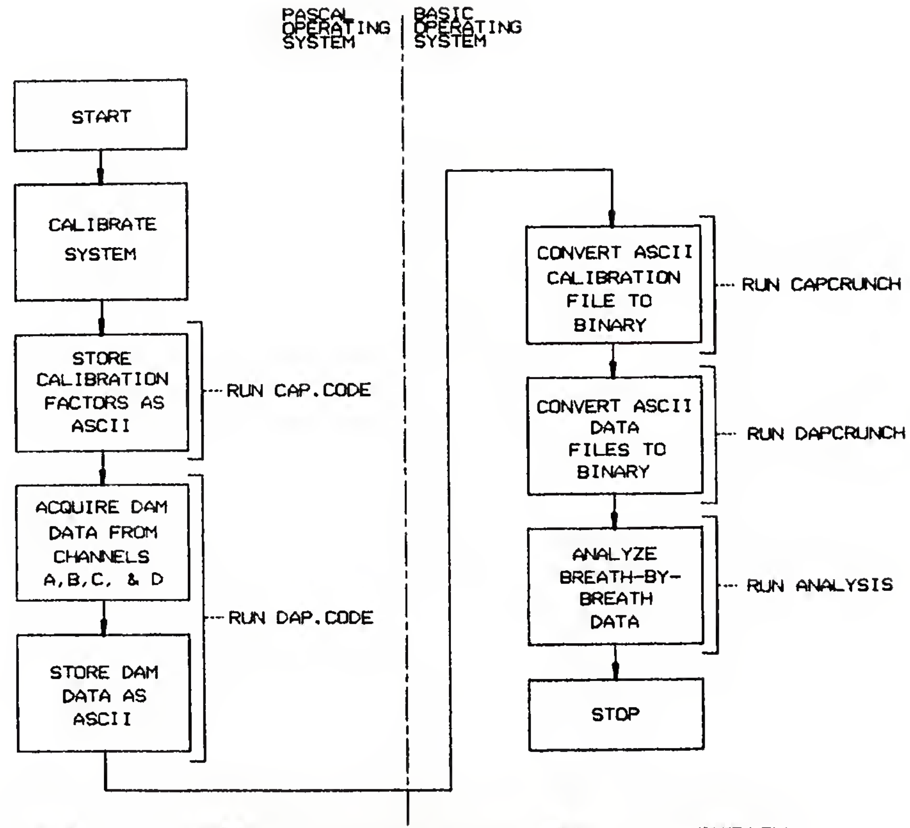


FIGURE 4.1 SEQUENCE FOR COLLECTING BREATH-BY-BREATH RESPIRATORY INFORMATION

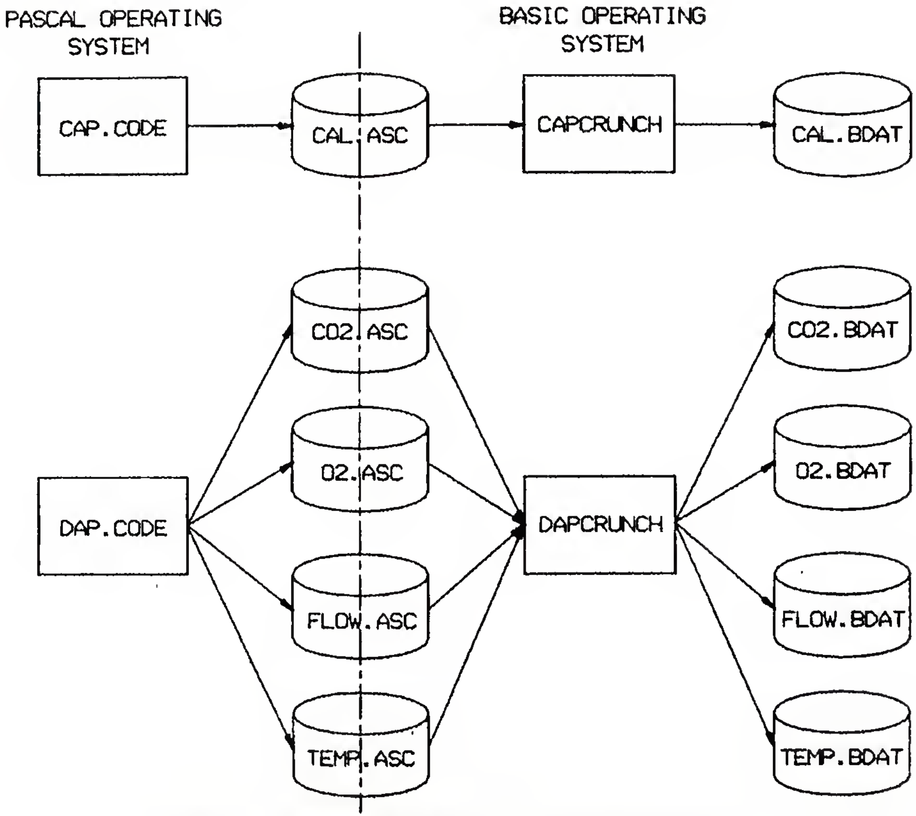


FIGURE 4.2 PASCAL/BASIC FILE CONVERSION

4.2 Calibration Software

The PASCAL routine CAP.CODE contains the various procedures for calibrating the system transducers. CAP.CODE generates an ASCII calibration file containing calibration factors (numbers used to convert the binary data collected by the DAM to known units, i.e. to fractional concentration values or degrees C) and DC offsets. Creel [11] describes in detail the means by which DC offsets and calibration factors are determined.

Three procedures internal to CAP.CODE perform the calibration necessary for the GMS (GASCAL), the Fleisch/Godart apparatus (FLOWCAL), and the respiratory temperature transducer (TEMPCAL). Following is a list of the calibration factors and DC offsets generated by these calibration procedures.

Calibration Parameter	PASCAL Procedure
Bin_zero_flow	FLOWCAL
Co2_cal	GASCAL
Co2_dc_offset	GASCAL
Expr_flow_cal	FLOWCAL
Insp_flow_cal	FLOWCAL
O2_cal	GASCAL
O2_dc_offset	GASCAL
O1	GASCAL
Ta	TEMPCAL
Tb	TEMPCAL
Tc	TEMPCAL

For a complete description of these calibration parameters and thorough CAP program documentation see Appendix VIII.

4.3 Data Acquisition Software

The software necessary to monitor and control the DAM is found throughout the PASCAL programs CAP.CODE and DAP.CODE. This software (located primarily in the procedure DATA_COLLECT) was written in PASCAL to allow the DAM to be controlled to sampling frequencies of 350 Hz. In addition to DATA_COLLECT, two 68000 assembly language routines were written to monitor the STatuS (STS) bit of the AD574 (Analog-to-Digital converter (A/D)) to determine when the A/D finishes a conversion. These two assembly language routines were written simply because comparable PASCAL routines were not fast enough to accurately monitor the STS bit. Figure 4.3 outlines the procedure used to control and monitor the DAM. Details of the control and status words for the DAM as well as complete program documentation for the mentioned software can be found in Appendices VIII, IX, and X.

Another critical operation performed by the data acquisition software is the setting of the sampling frequency. This is accomplished by writing appropriate values to the Intel 8253 programmable interval timer found on the DAM. The PASCAL procedure CLKSET documented in Appendix VIII determines these values based upon the desired sampling frequency selected by the user. Refer to the 1980 Intel Data Catalog [19] for complete programming instructions on the 8253.

4.4 File Manipulation Software

As mentioned in the software overview section, a certain amount of file manipulation (conversion from ASCII to BDAT files) is necessary. The ASCII data and calibration files are needed as the only compatable file type between the PASCAL and BASIC

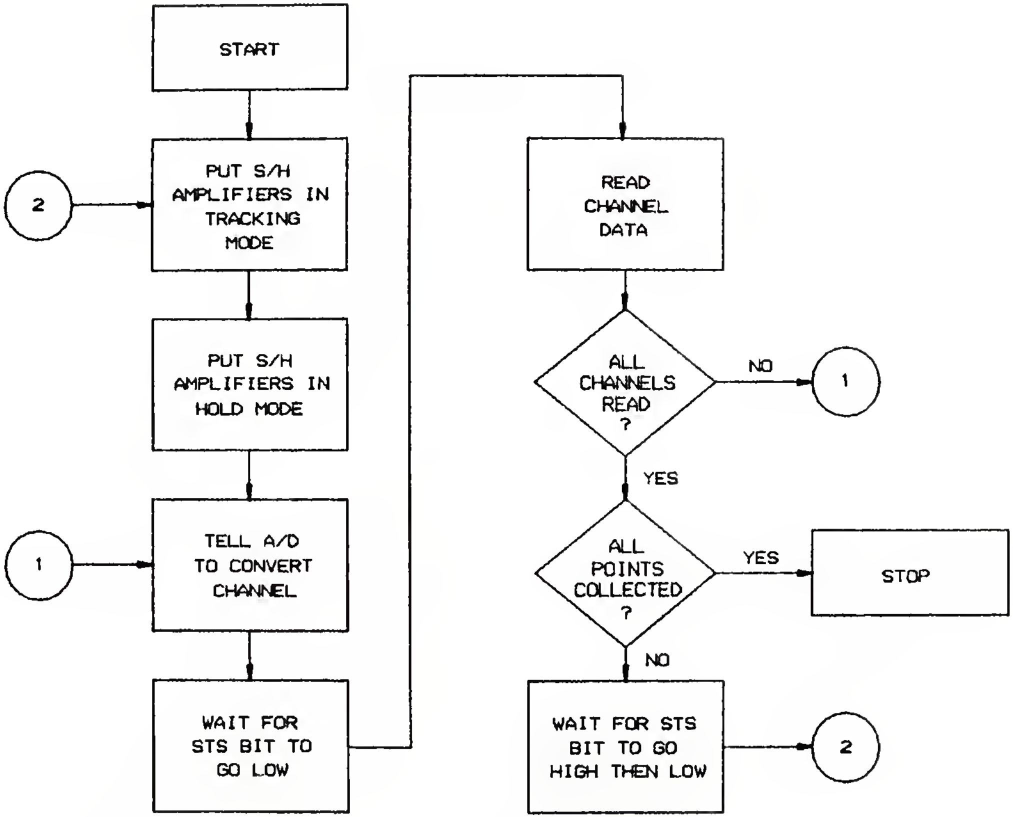


FIGURE 4.3 DAM CONTROL SCHEME

operating systems is ASCII. The conversion of these ASCII files to BDAT files is done to reduce the amount of mass storage necessary for a given data or calibration file and also to increase the speed at which these files can be loaded into memory which ultimately leads to lower analysis times.

Referring to Figure 4.2, CAP.CODE (the PASCAL system calibration routine) creates an ASCII calibration file that is compatible with both the PASCAL and BASIC operating systems on the HP9134A hard disk. CAPCRUNCH (the BASIC calibration file crunch routine) then converts the ASCII calibration file to a BDAT file which can then be read by the BASIC analysis routine ANALYSIS. The larger ASCII file is purged (deleted) and the BDAT file is stored on the HP9895A 8" flexible disk for maximum data portability.

In similar fashion, DAP.CODE creates four ASCII data files (one for each of the four analog input channels) and DAPCRUNCH crunches these ASCII files to comparable BDAT files for use by ANALYSIS. Complete software documentation for CAPCRUNCH and DAPCRUNCH can be found in Appendices XII and XIII respectively.

4.5 Data Analysis and Display Software

The BASIC routine, ANALYSIS, performs those functions on the respiratory data involved with analyzing (on a breath-by-breath basis) and displaying of the data collected from exercising humans. These functions include:

1) Calculation of the total time associated with the capillary gas transport and GMS response on a breath-by-breath basis.

2) Calculation of the breath-by-breath O_2

consumption and CO_2 production using digitized flow and fractional concentration signals and appropriate calibration factors. Windowing of the respiratory data is possible.

3) Calculation of additional respiratory quantities based on the digitized respiratory signals (see Appendix XIV for complete details).

4) Calculation and application of correction factors to compensate for temperature and pressure differentials between the subject and the environment.

5) Provides plots of the four respiratory signals (CO_2, O_2, flow, and respiratory temperature) using either the HP9826 CRT (in conjunction with the HP2673A thermal printer) or the HP9872C 8 pen plotter. Windowing of the plotted respiratory data is also possible.

Of the functions mentioned, two deserve further explanation. Capillary gas transport and GMS response is currently corrected on a breath-by-breath basis. Figure 4.4 depicts the means by which this variable time delay is determined. The respiratory flow signal is examined to locate the start of inspiration (the beginning of inspiration is that point in the flow signal that is less than or equal to binary zero flow followed by five binary points less than binary zero flow).

Once the beginning of inspiration is found, the CO_2 signal is examined to locate the maximum CO_2 level within the current breath. The location of this maximum is labeled Tmax in Figure 4.4. That time (or sample point) that corresponds to 1/2 of the maximum CO_2 level is then located. This point is labeled Tmid.

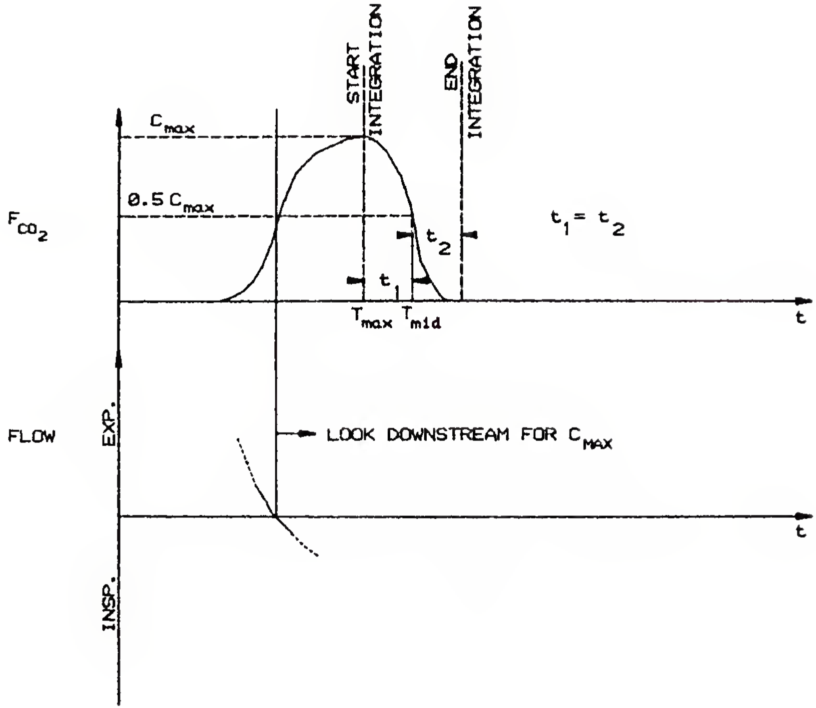


FIGURE 4.4 DETERMINATION OF VARIABLE TIME DELAY

The time (or point) difference is then found between Tmid and Tmax and this difference is added to Tmid to locate that point on the CO_2 signal known as the ending point of integration (Tmax is also known as the starting point of integration).

The time between Tmax and the ending point of integration is then located where the area below the CO_2 signal (from Tmax to the mentioned time) equals the area above the CO_2 signal (from the mentioned time to the ending point of integration. Subtracting from this absolute time the absolute time of the flow zero crossing yields the time delay for the breath in question. The previous operation is then performed for each successive breath. Figures 4.5 and 4.6 depict non-time aligned and time aligned respiratory signals.

The second function deserving explanation is the calculation of breath-by- breath respiratory gas volumes. Creel [11] goes into great detail on how these volumes are computed and how corrections to STPD and BTPS conditions are possible. To avoid repetition, the author refers you to his work.

One function that has been added to the human respiratory research is the possibility of windowing the accumulated data both in the analysis of the data and the plotting routine. This addition allows for the analysis of transient exercise phenomenon as well as steady-state analysis. By selecting the desired points over which analysis is to take place, ANALYSIS will perform the mentioned functions only on the window of choice.

Samples of hard copy outputs from an experimental run are shown in Figures 4.7 and 4.8. Examples of the plotter routine output can be found throughout this thesis.

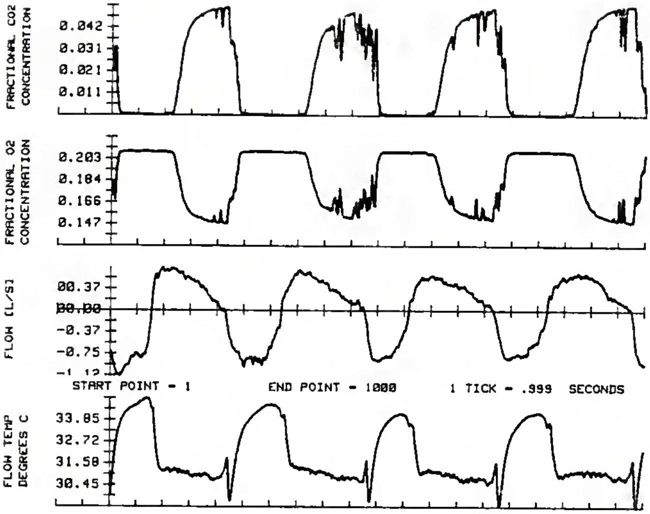


FIGURE 4.5 NON-TIME ALIGNED RESPIRATORY SIGNALS

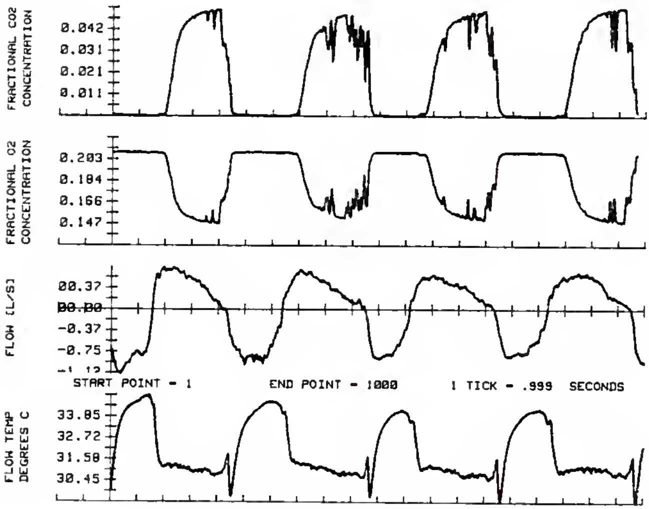


FIGURE 4.6 TIME ALIGNED RESPIRATORY SIGNALS

SUBJECT IDENTIFIER: COLD AIR 50 & 200

DATE: 7/24/84

Breath Start Index	Air Inspired (liters)	Air Expired (liters)	O2 Inspired (liters)	O2 Expired (liters)	CO2 Inspired (liters)	CO2 Expired (liters)	O2 Consumed (liters)	CO2 Produced (liters)	Insp TIME (sec)	Expr TIME (sec)	Delay TIME (msec)
4874	-1.133	1.237	-.236	.209	-.001	.047	-.027	.046	1.14	1.38	380
5000	-1.110	1.132	-.232	.193	-.001	.041	-.039	.040	1.12	1.30	380
5121	-1.148	1.083	-.240	.184	-.001	.039	-.055	.038	1.26	1.26	380
5247	-1.176	1.188	-.245	.201	-.001	.044	-.044	.043	1.14	1.30	380
5369	-1.181	.991	-.247	.170	-.001	.034	-.076	.033	1.22	1.30	400
5495	-.978	1.087	-.204	.183	-.001	.040	-.021	.039	1.16	1.32	380
5619	-1.153	1.094	-.241	.186	-.001	.039	-.055	.038	1.18	1.30	380
5743	-1.057	1.072	-.220	.181	-.001	.039	-.039	.038	1.16	1.38	380
5870	-1.114	1.099	-.232	.186	-.001	.040	-.047	.039	1.14	1.30	380
5992	-1.130	1.086	-.236	.182	-.001	.040	-.054	.039	1.20	1.32	400
6118	-1.130	1.086	-.236	.183	-.001	.039	-.053	.038	1.20	1.28	380
6242	-1.030	1.081	-.215	.181	-.001	.040	-.034	.039	1.18	1.30	380
6366	-1.143	1.076	-.238	.181	-.001	.039	-.057	.038	1.18	1.32	380
6491	-1.003	1.051	-.209	.176	-.001	.038	-.033	.037	1.20	1.30	380
6616	-1.118	1.176	-.233	.195	-.001	.044	-.038	.044	1.16	1.32	380
6740	-1.205	.985	-.251	.168	-.001	.033	-.084	.032	1.28	1.26	380
6867	-1.049	1.141	-.219	.189	-.001	.043	-.029	.042	1.14	1.38	380
6993	-1.149	1.039	-.239	.177	-.001	.036	-.063	.034	1.18	1.28	360
7116	-1.188	1.166	-.248	.195	-.001	.043	-.053	.042	1.20	1.36	380
7244	-1.096	.950	-.229	.163	-.001	.031	-.066	.030	1.10	1.36	360
7367	-.914	1.085	-.190	.179	-.001	.041	-.011	.040	1.14	1.26	380
7487	-1.169	1.249	-.244	.202	-.001	.051	-.042	.050	1.18	1.42	400
7617	-1.210	1.159	-.252	.192	-.001	.044	-.060	.043	1.24	1.26	380
7742	-1.134	1.230	-.237	.200	-.001	.049	-.036	.048	1.16	1.36	400
7868	-1.184	.963	-.247	.163	-.001	.034	-.084	.033	1.20	1.20	400
7988	-1.097	.877	-.229	.149	-.001	.030	-.080	.029	1.30	1.26	400

FIGURE 4.7 HARD COPY OUTPUT OF BREATH-BY-BREATH DATA

Inspiratory minute volume = -26.8 liters per minute

Expiratory minute volume = 26.3 liters per minute

Inspiratory tidal volume = -1.1153 liters

Expiratory tidal volume = 1.0917 liters

Respiratory frequency = 24.1 breaths per minute

Mean O2 inspired = -.233 liters

Mean O2 expired = .183 liters

Mean CO2 inspired = -.001 liters

Mean CO2 expired = .040 liters

Mean O2 consumed per breath = -.049 liters

Mean CO2 produced per breath = .039 liters

O2 consumed per minute = -1.188 liters per minute

CO2 produced per minute = .938 liters per minute

RESPIRATORY QUOTIENT = .790

Total time of inspiration = 30.8 sec

Total time of expiration = 34.1 sec

Total time of respiration = 64.8 sec

Number of good inspirations = 26.0

Number of good expirations = 26.0

Number of good breaths = 26.0

FLOW DC OFFSET = 2038

CO2 DC OFFSET = 244

O2 DC OFFSET = 498

CO2 CALIBRATION FACTOR = 1.8700E-05

O2 CALIBRATION FACTOR = 2.3400E-05

INSPIRATORY FLOW CALIBRATION FACTOR = 3.2001E-03

EXPIRATORY FLOW CALIBRATION FACTOR = 3.1789E-03

TEMPERATURE CORRECTION = 0.0000E+00X^2 + 0.0000E+00X + 0.0000E+00

SAMPLING FREQUENCY = 50

FLOW CALIBRATION FILENAME: CAL724

CO2 DATA FILENAME: C50724

O2 DATA FILENAME: O50724

FLOW DATA FILENAME: V50724

TEMPERATURE DATA FILENAME: T50724

FIGURE 4.8 HARD COPY OUTPUT OF AVERAGE RESPIRATORY DATA

V. EXPERIMENTAL METHODS

5.1 Subject Selection and Preparation (Breath-by-breath)

To evaluate the performance of the computer-based instrumentation system, a well conditioned, male graduate student (age, 24 years) was selected as the test subject. This subject was selected to minimize fluctuations in respiration (both during rest and exercise) that often occurs in the untrained individual during exercise and to reduce the possibility of physiological conditioning that would almost certainly occur in a sendentary subject.

The bicycle ergometer seat was first adjusted so as to fit the subject comfortably (a seat heighth not quite high enough to allow full leg extension is preferred). As mentioned in the Instruments and Interconnects section, the subject was fitted with the Hans Rudolf respiratory mask (#7900M) and a heated No. 2 Fleisch head assembly. Heating of the PTM was done in an attempt to warm the inspired air to body temperature so comparison between the breath- by-breath system and an end-expired bag collection technique [2,3] could be made. The GMS probe was also secured to the No. 2 Fleisch head on the room air side of the PTM. This insures the GMS samples room air at the beginning of inspiration and not a high CO_2 gas concentration that would be present on the subject side of the PTM.

5.2 System Calibration (Breath-by-breath)

Before meaningful data collection using the breath-by-breath system can begin, system calibration must be performed, preferably immediately before the exercise trial is conducted. System calibration is controlled entirely by the PASCAL routine (see Appendix VIII for complete documentation). This routine guides the operator through the entire calibration procedure which involves GMS calibration, flow signal calibration, and respiratory temperature calibration. Following is a brief summary of the calibration that is performed.

Calibration of the fractional gas concentrations is necessary to determine the relationship between the fractional CO_2 and O_2 gas concentrations and the associated binary values read from the DAM.

GMS calibration is accomplished by simply placing the GMS sampling probe in the extreme CO_2 and O_2 gas concentrations that one would most likely encounter in respiratory studies of this type (21% O_2, 0% CO_2 was one calibration point and 12.9% O_2, 7% CO_2 was the other calibration point) and determining the proper CO_2 and O_2 calibration factors as described by Creel [11].

Figure 3.4 depicts that apparatus and interconnections necessary to perform the GMS calibration. As is obvious from this figure, zero suppression (Creel [11]) of both the fractional CO_2 and O_2 channels was necessary to allow the output of the GMS to utilize the entire analog input range of the DAM (Gateno [13]). For the complete step-by-step procedure used to calibrate the GMS refer to Appendix III.

Calibration of the respiratory flow signal (PTM calibration) consists of determining inspiratory and expiratory flow calibration factors and the binary value from the DAM associated with zero flow through the PTM. Creel [11] desribes these three variables in detail; that explanation will not be repeated here. In short, flow calibration involves forcing a known volume of gas (via the Harvard Respirator) through the PTM in both inspiratory and expiratory directions, integrating the resulting flow signal generated by the PTM assembly, and determining the ratio between the known gas volume and the mentioned integration (areas).

Figure 3.3 shows the apparatus and interconnections used for the flow signal calibration. For the complete flow calibration procedure see Appendix III.

Calibration of the respiratory temperature signal involves measuring three known temperatures (water baths) and then computing a second order polynomial curve fit of the data. Masters' [17] decribes this procedure in detail.

5.3 <u>Data Collection and Analysis (Breath-by-breath)</u>

To validate the breath-by-breath respiratory system, a stringent exercise program was selected. This program consisted of both morning and afternoon runs. The morning's run included a 40 second rest period, followed immediately by a 3 minute exercise period at 50 watts, and then a 200 watt work load for 4 minutes and 20 seconds (the total exercise run was thus an 8 minute run, the maximum allowable run possible with the present system operating at 50 Hz). The afternoon run included a 40 second rest period, followed by a 3 minute exercise period at 100 watts and a 150 watt work load for the remaining 4 minutes and 20 seconds.

To collect the mentioned data, 24000 data points per channel were collected at a rate of 50 Hz. Thus the data collection period was exactly 8 minutes. Windowing of the data was performed for the last 2 minutes of the data at each of the various work loads. This insured that the analysis was performed on steady-state respiratory data and not on transient respiratory information.

During data collection, a metronome was used to pace the rider at 50 rpm and the ergometer belt was continually adjusted to achieve the desired work load. Ergometer wheel revolutions were observed using an optical revolution counter as a indicator of the total work done during exercise. As previously mentioned, the PTM was heated in an attempt to elevate the inspired air to body temperature (this was done so that all analysis could be assumed to be at body temperature and comparison could then be made to the end-expired bag collection technique which corrects its results to body temperature). Later, the PTM heat was increased after crude temperature measurements indicated that the PTM was not heating the inspired air to body temperature (see Experimental Results for details).

5.4 System Comparison with Bag Collection Technique

For system verification, comparisons with an end-expired bag collection technique were made. The bag collection technique consisted of a two-way non-rebreathing valve and a meteorological balloon. Using the valve, expired gases were collected in the bag during data collection. Once the bag was nearly full (usually 90 seconds of gas collection) the collected gas was dumped to a spirometer for accurate measurement of the gas volume. While dumping the gas, the O_2 and CO_2 fractional gas concentrations of the collected gas were measured using the GMS. Body temperature, elapsed time of gas collection, and spirometer gas temperature were recorded. From these quantities, several respiratory quantities are calculated.

The actual exercise routine used to collect the expired gas is similar in form to the breath-by-breath studies. Morning data collection consisted of 3 minutes at 50 watts followed immediately by 5 minutes at 200 watts. During the last 90 seconds of the mentioned work loads, gas collection occurred. Afternoon data collection closely paralleled the morning runs with 3 minutes at the 100 watt work load and 5 minutes at 150 watts.

VI. EXPERIMENTAL RESULTS

6.1 Presentation of Results

Although the breath-by-breath respiratory system is capable of displaying transient respiratory information as well as steady-state results, only the average steady-state values generated by this system could be compared with the bag collection technique. In particular, three parameters (expiratory minute volume, rate of CO_2 production, and rate of O_2 consumption) were compared between the two systems.

As is obvious from the discussion above, in order for comparisons to be made, steady-state conditions must be reached. Figures 6.1 and 6.2 show the transient as well as steady-state information that is contained in the breath-by-breath data for 50 watt and 100 watt work loads. To insure that only analysis of steady-state data was made, scatter plots of the rate of O_2 consumption on a per breath basis were made. Figures 6.3 and 6.4 are typical plots for the 50-200 watt and 100-150 watt work load trials. These plots enabled the justification of the use of the final two minutes at a particular work load for steady-state evaluations.

For the breath-by-breath studies, three plots (Figures 6.5-6.7) depict the results of five trials conducted at the work loads shown. As previously mentioned, expiratory minute volume, rate of CO_2 production, and rate of O_2 consumption were plotted so comparisons with the bag collection technique could be made. Figures 6.8 through 6.10 illustrate the same parameters for the bag collection method.

Having compared these six plots, it was decided that the PTM was not warming the inspired air to body temperature, thus causing the rate of O_2 consumption in the breath-by-breath system to be significantly higher at the high work loads (see the mean and standard deviation plot [Figure 6.16] for more details). After increasing the PTM heat to a level where the inspired air approached body temperature (a slowly responding Tektronix temperature probe was used for this adjustment) five more trials at the 50 and 200 watt work loads were made, the results of which are plotted in Figures 6.11 through 6.13.

For the nine plots mentioned, three mean and standard deviation plots (Figures 6.14-6.16) were generated for comparison between the two analysis techniques. Section VII includes a discussion of these three plots.

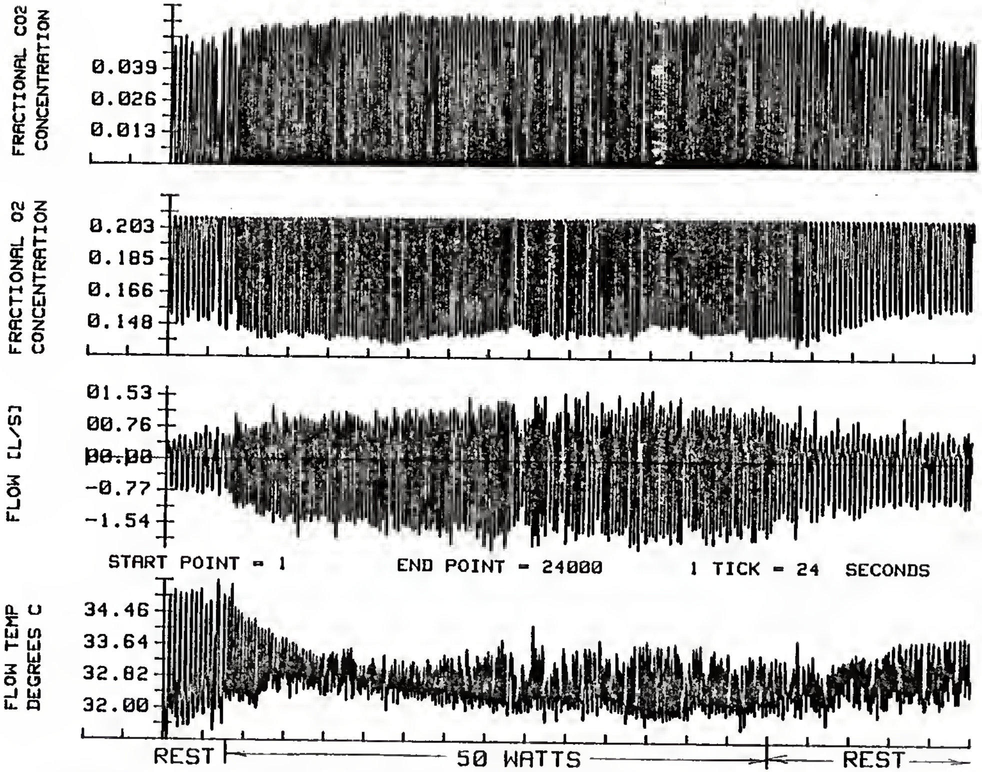


FIGURE 6.1 TIME DOMAIN RESPIRATORY PLOT (50 WATTS)

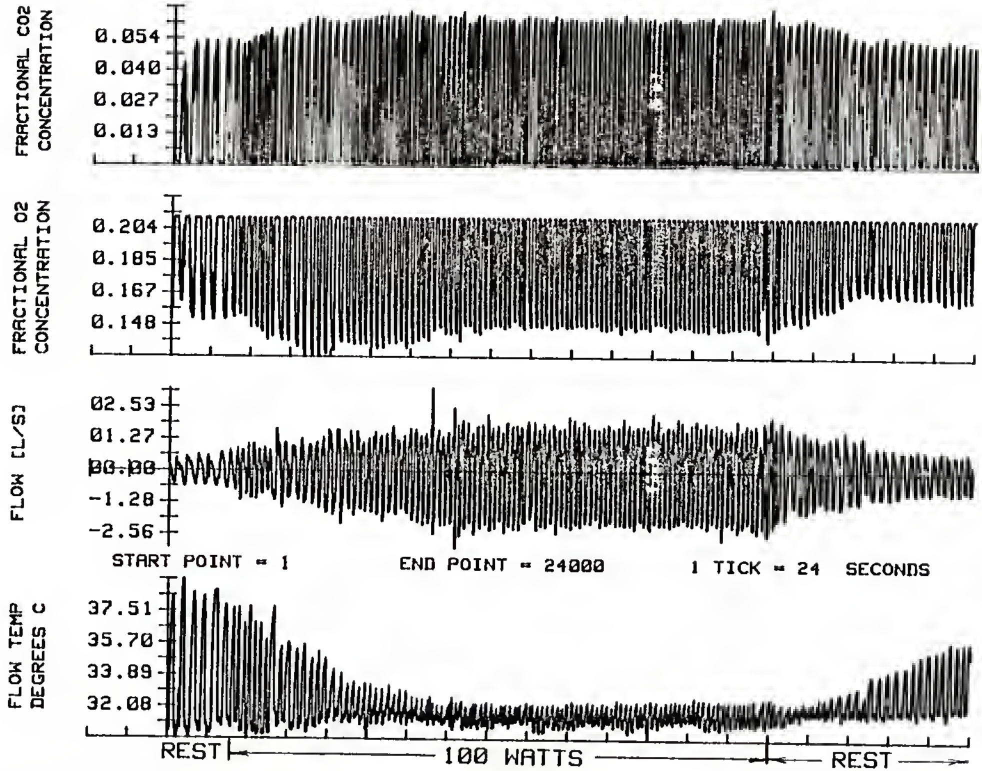


FIGURE 6.2 TIME DOMAIN RESPIRATORY PLOT (100 WATTS)

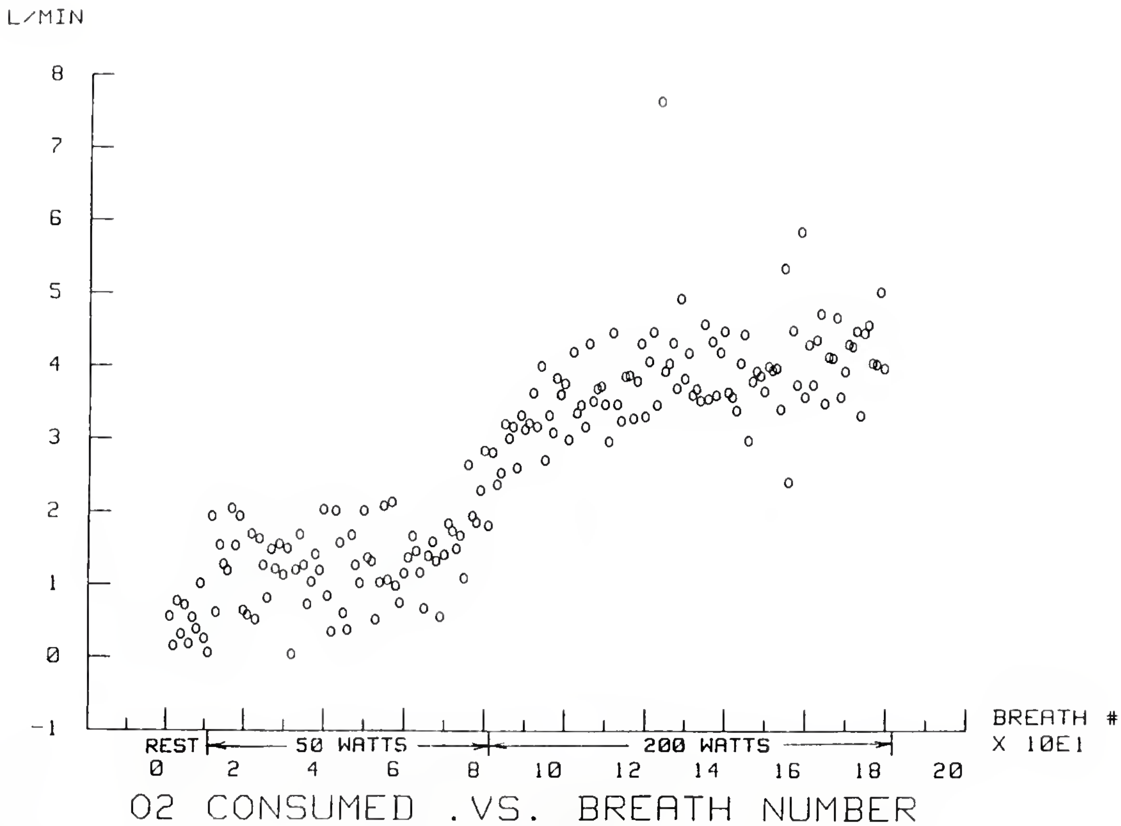


FIGURE 6.3 BREATH-BY-BREATH O2 CONSUMPTION

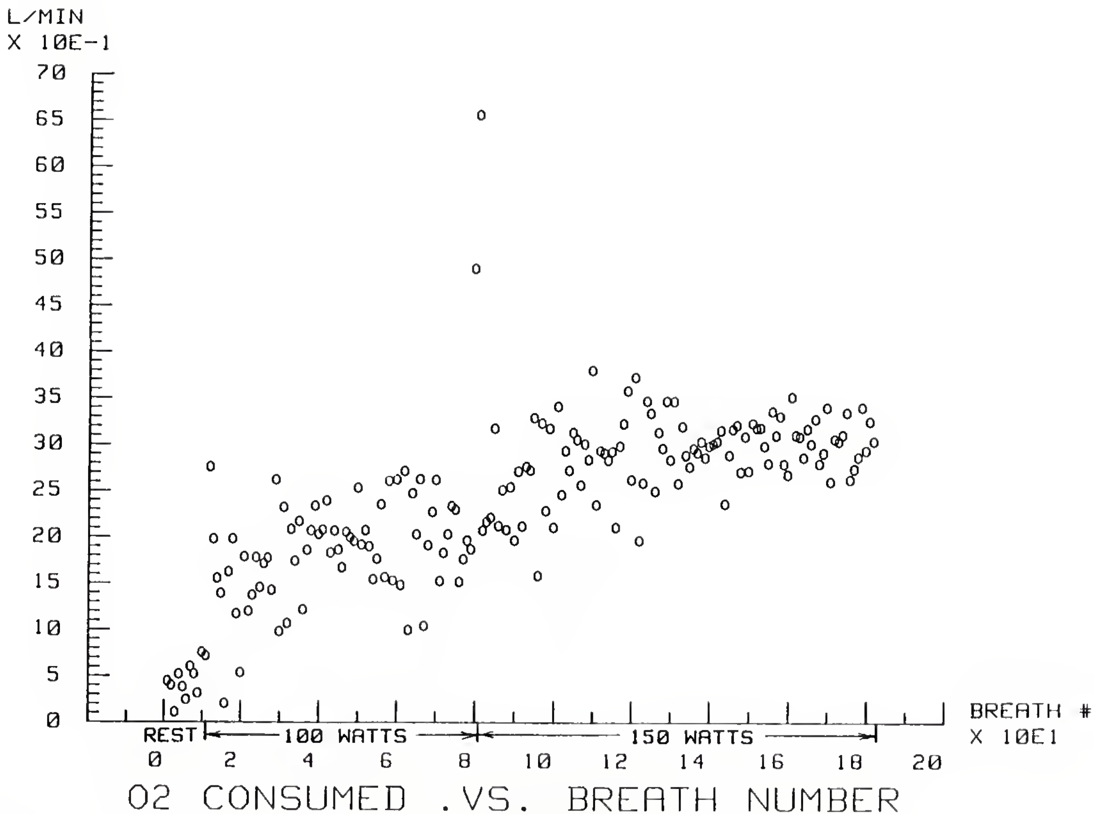
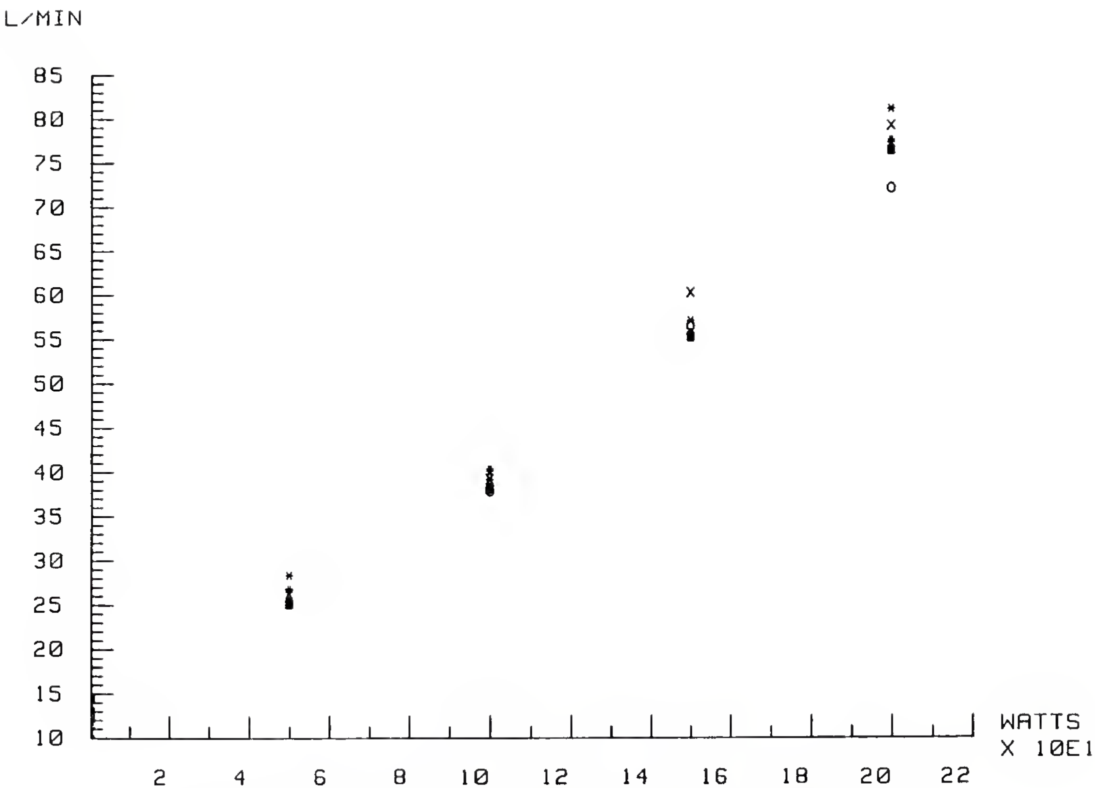


FIGURE 6.4 BREATH-BY-BREATH O2 CONSUMPTION



L AIR/MIN .VS. WORK LOAD

O. JULY 5 X. JULY 9 *. JULY 10 ■. JULY 11 ♦. JULY 12

FIGURE 6.5 BREATH-BY-BREATH EXPIRATORY MINUTE VOLUME

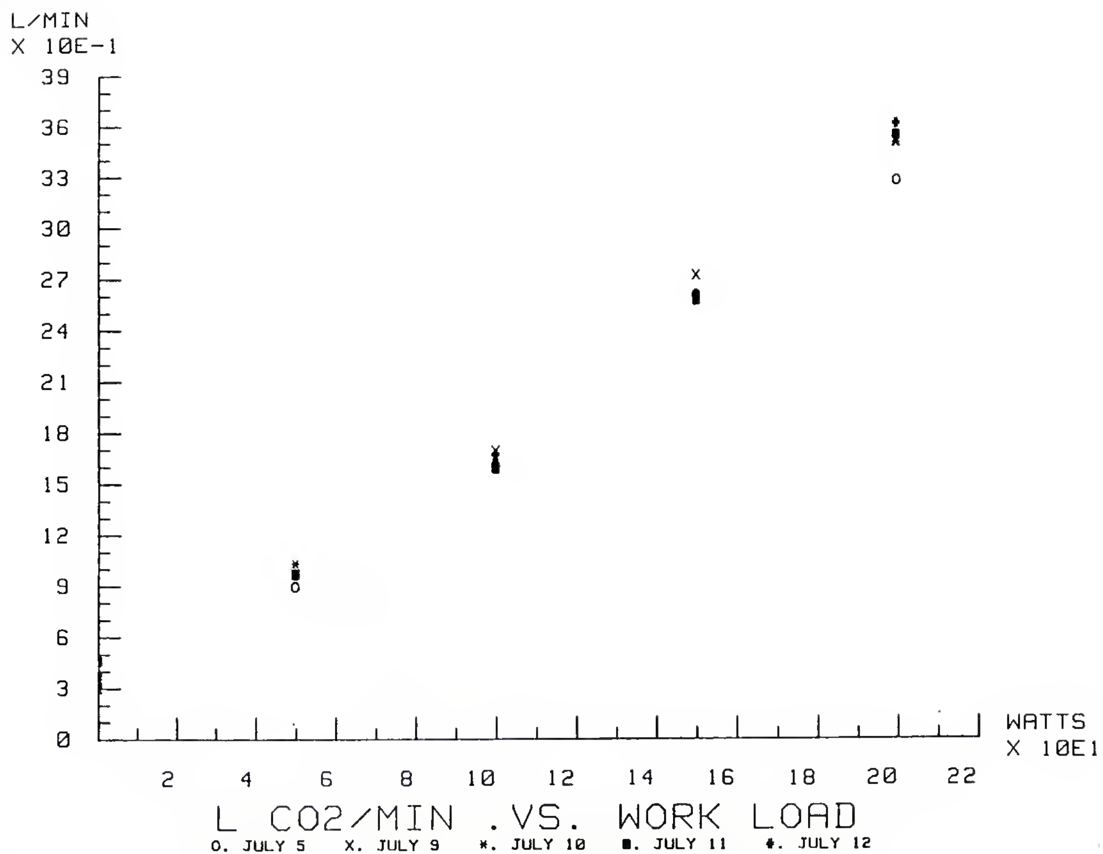


FIGURE 6.6 BREATH-BY-BREATH CO2 PRODUCTION

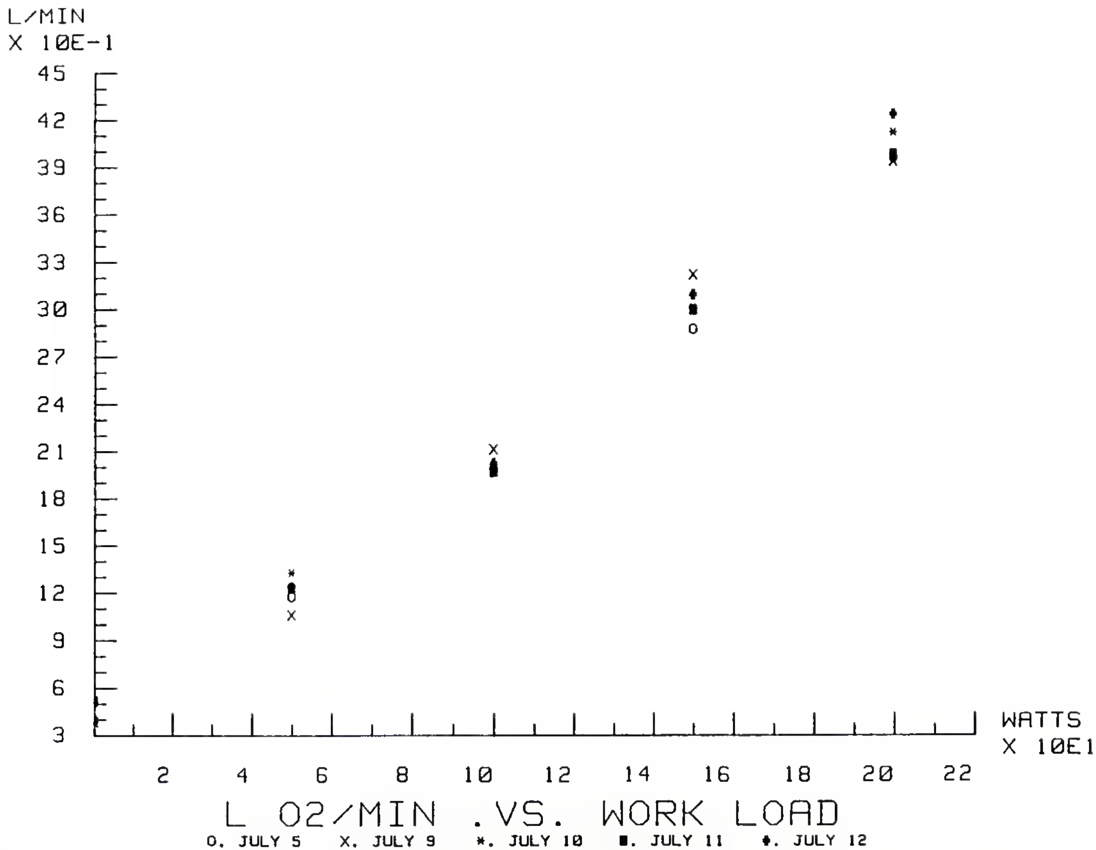


FIGURE 6.7 BREATH-BY-BREATH O2 CONSUMPTION

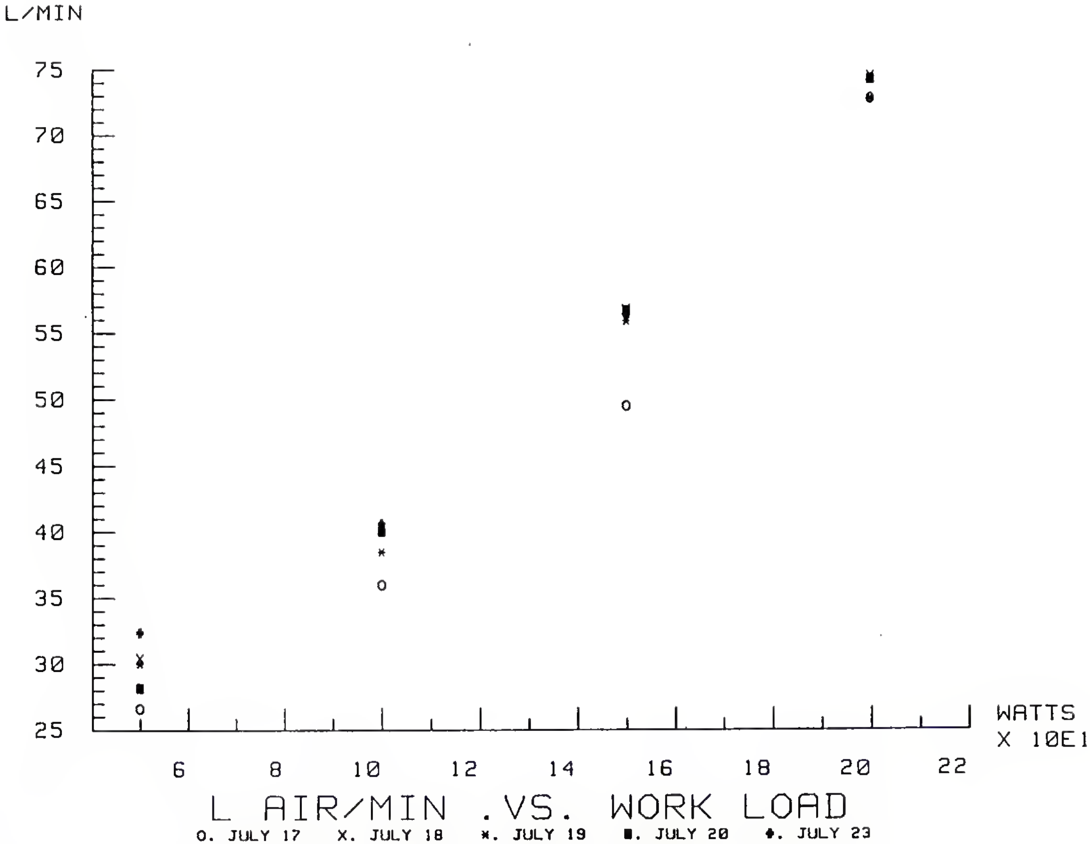


FIGURE 6.8 BAG EXPIRATORY MINUTE VOLUME

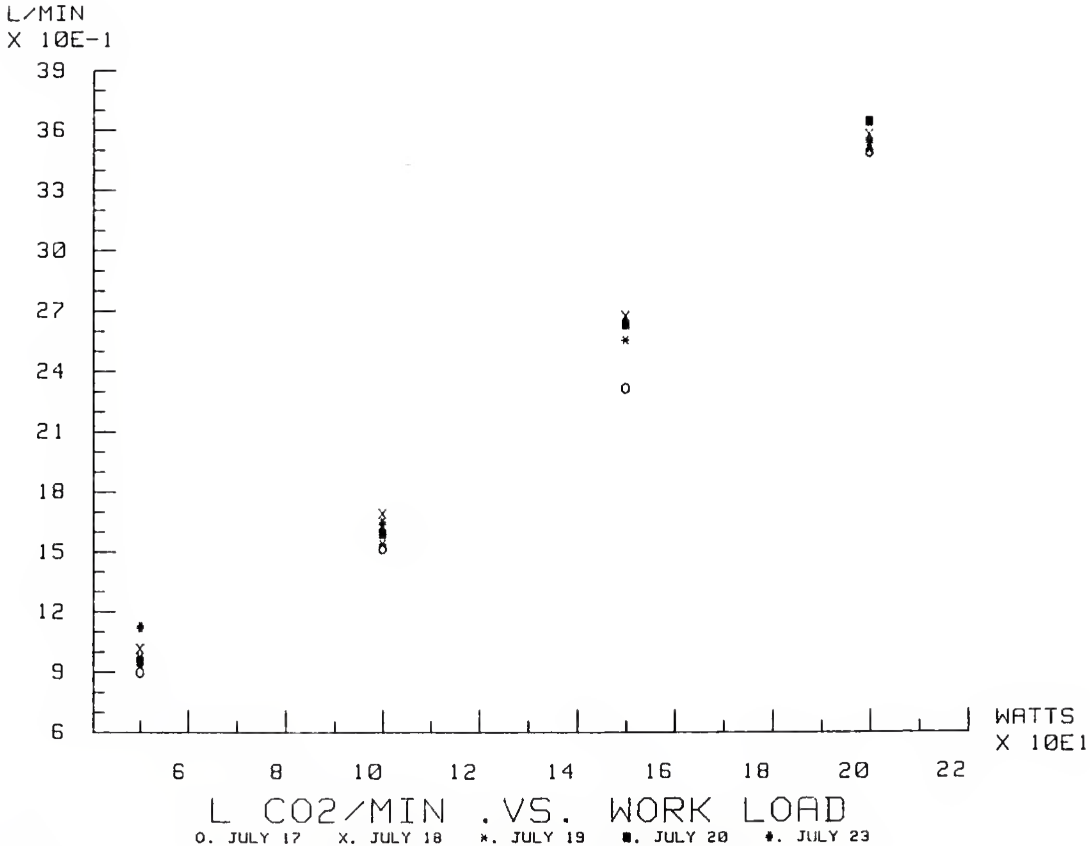


FIGURE 6.9 BAG CO2 PRODUCTION

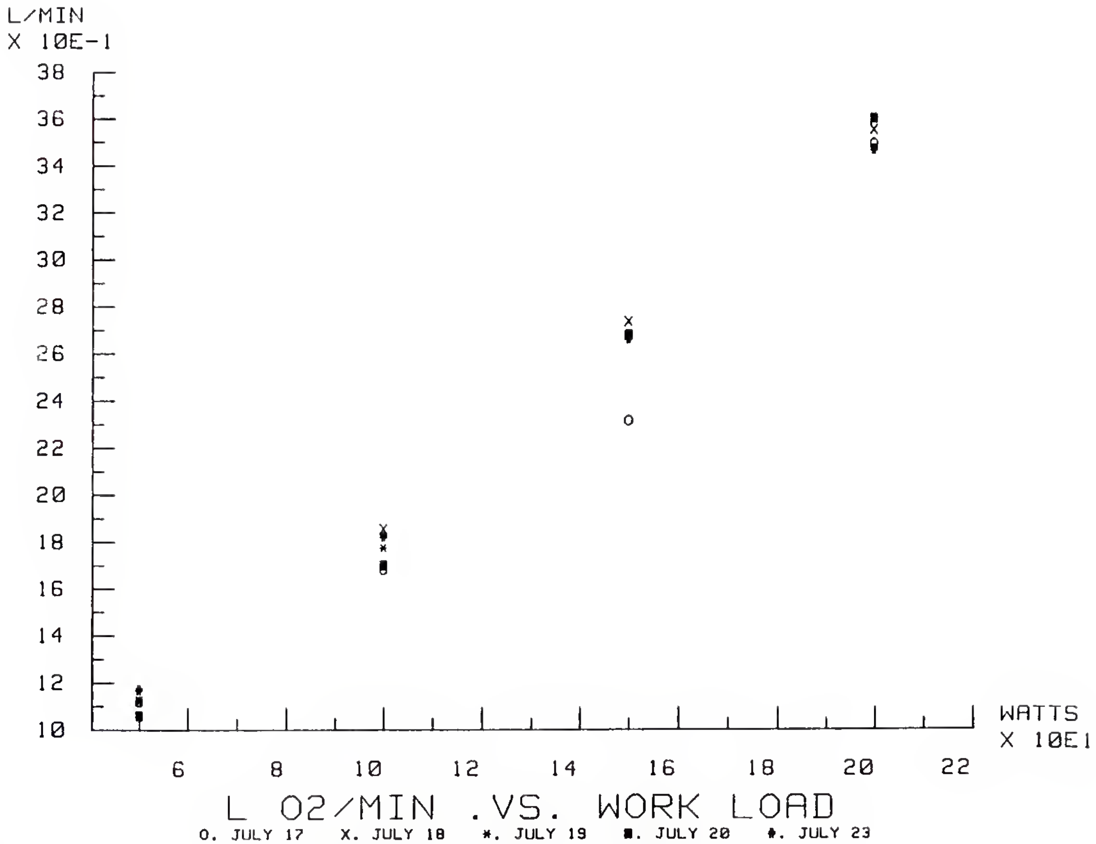
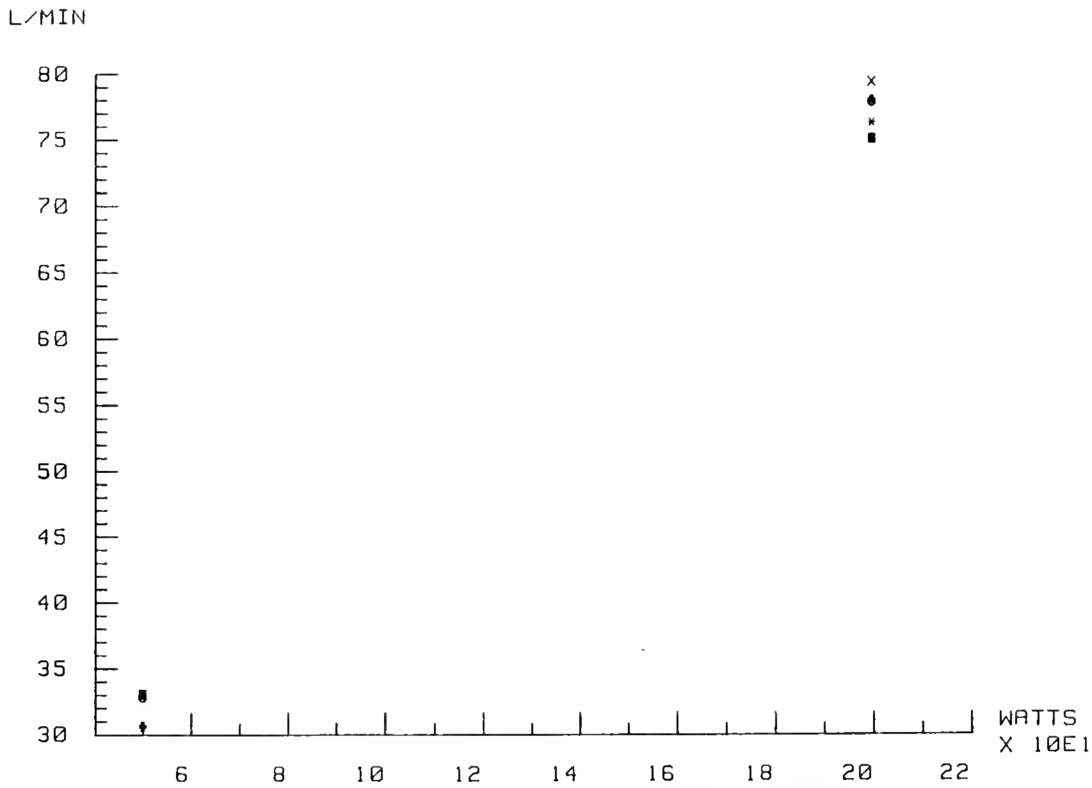


FIGURE 6.10 BAG O2 CONSUMPTION



L AIR/MIN .VS. WORK LOAD

O. JULY 26 X. JULY 27 *. JULY 30 AM ■. JULY 30 PM ♦. JULY 31

FIGURE 6.11 HEATED BREATH-BY-BREATH EXPIRATORY MINUTE VOLUME

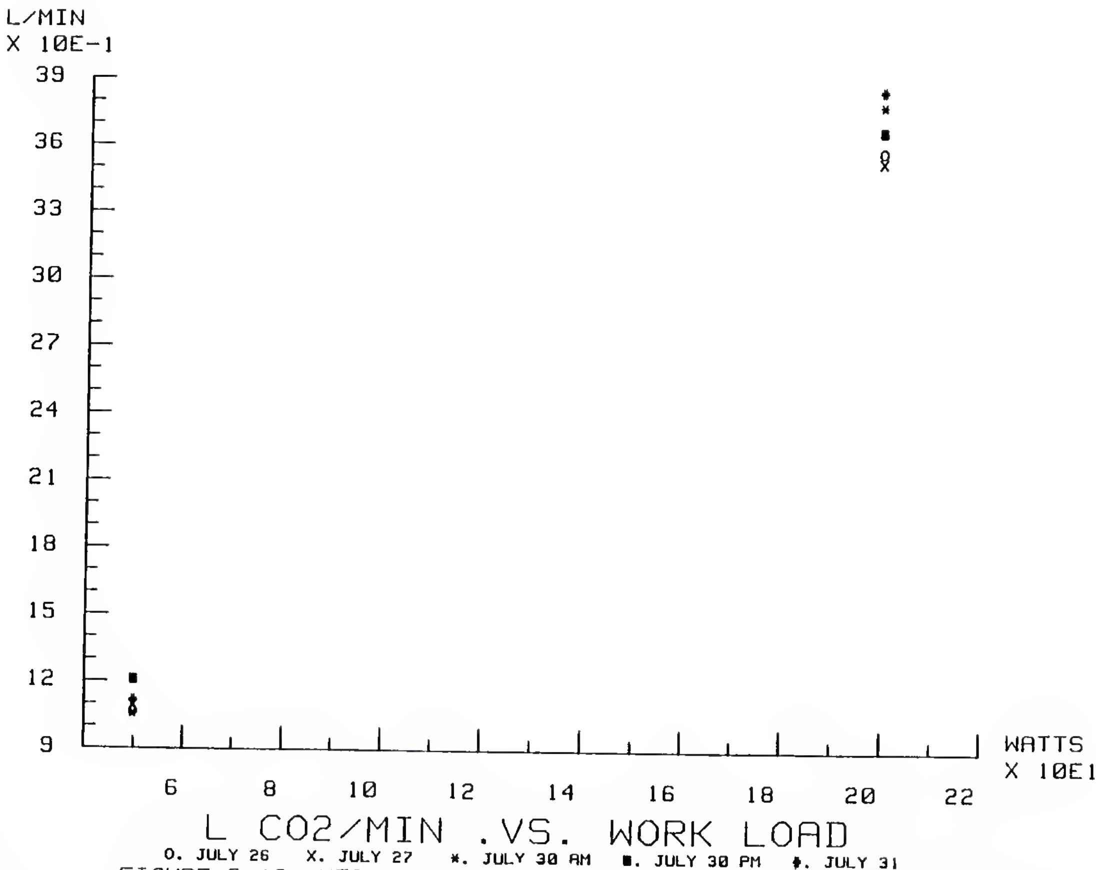


FIGURE 6.12 HEATED BREATH-BY-BREATH CO_2 PRODUCTION

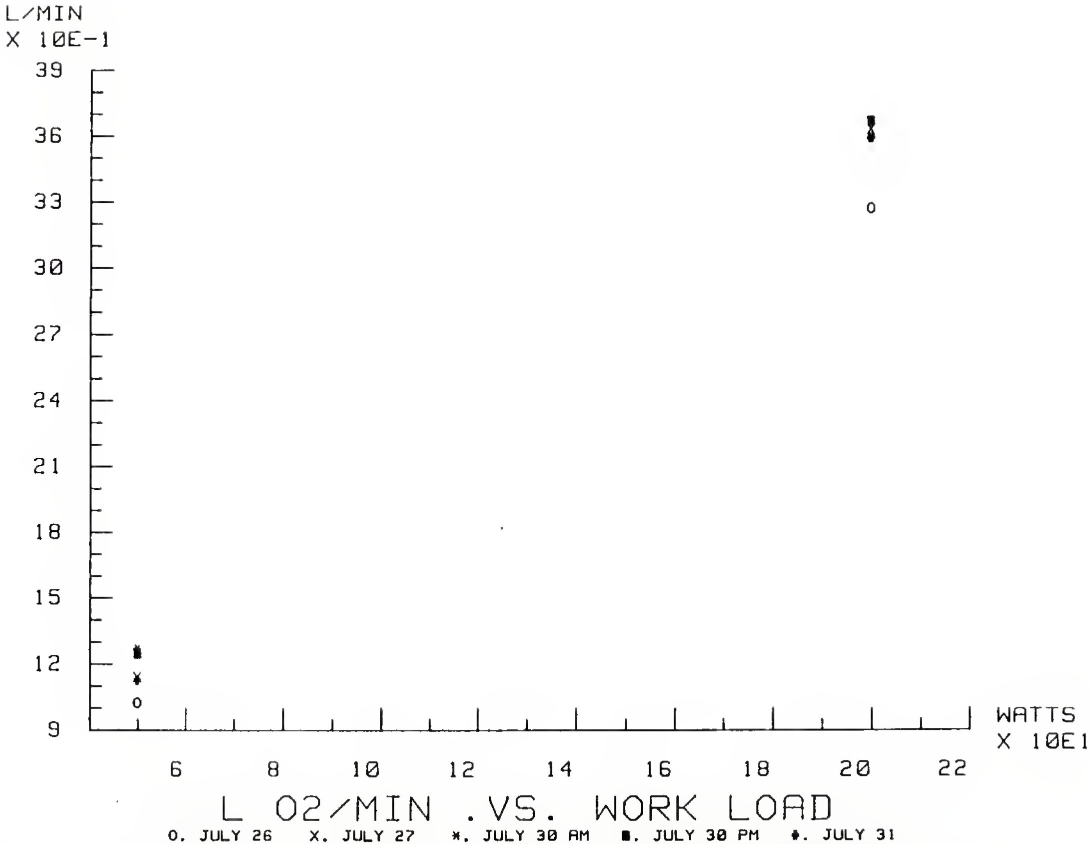


FIGURE 6.13 HEATED BREATH-BY-BREATH O2 CONSUMPTION

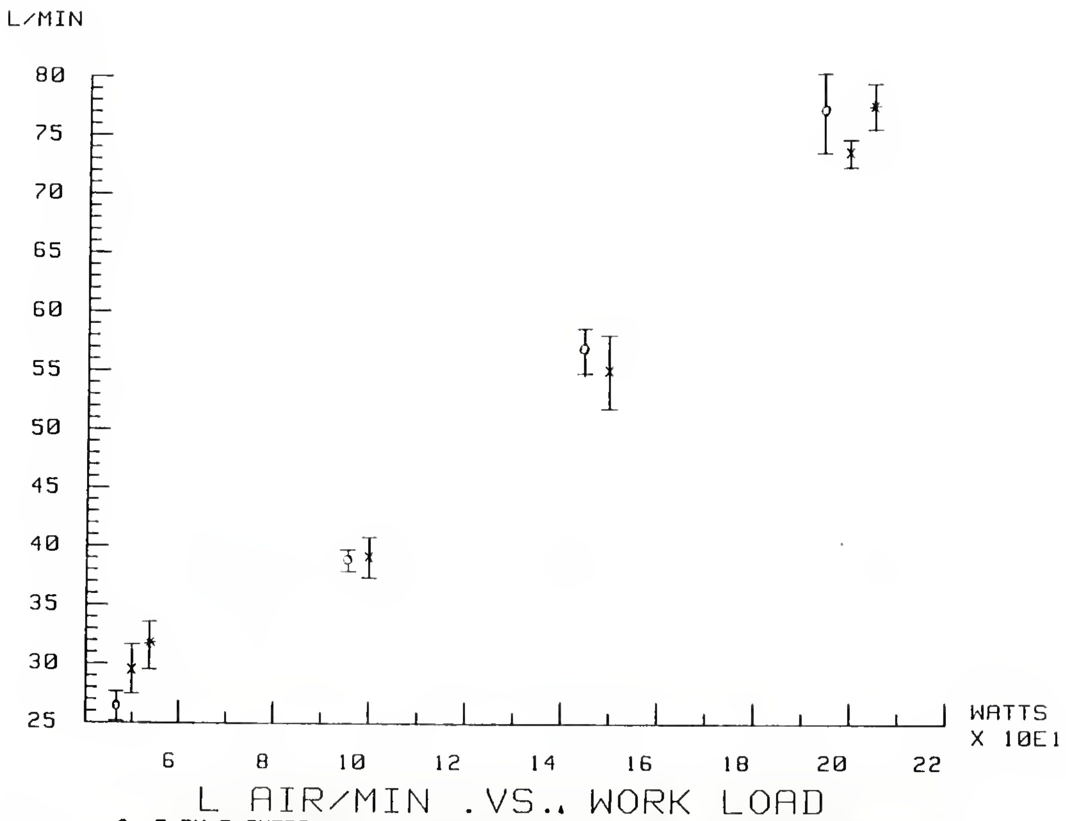
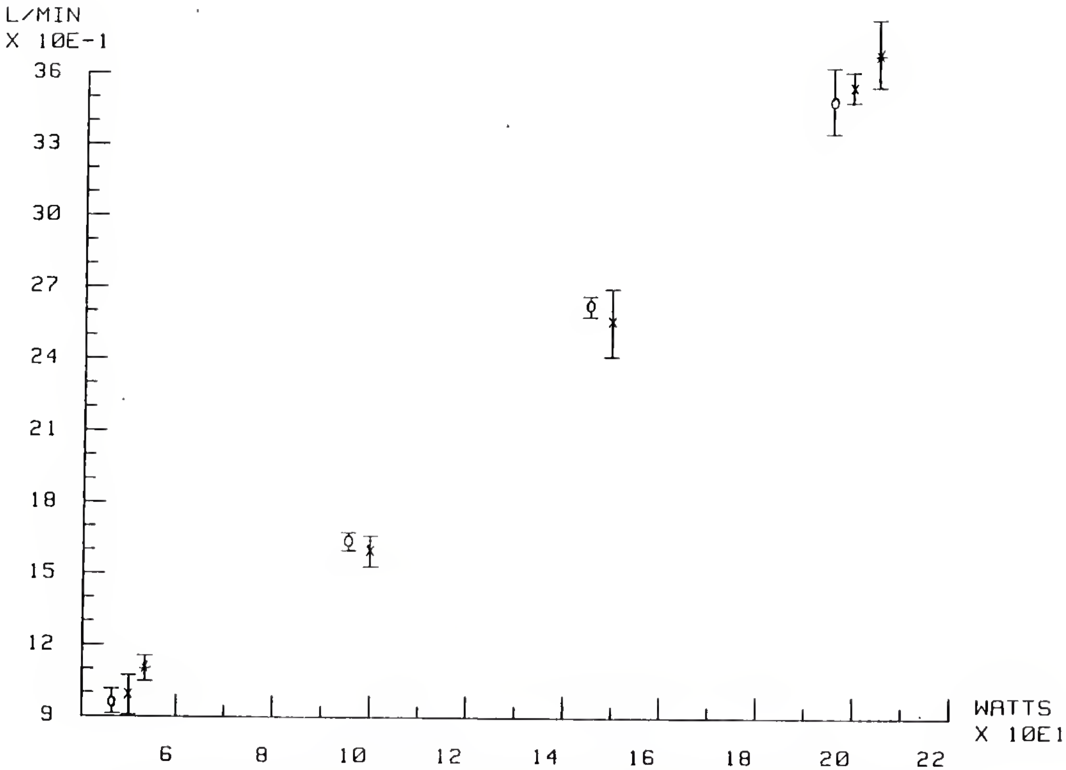


FIGURE 6.14 BREATH-BY-BREATH BAG EXPIRATORY MINUTE VOLUME



L CO2/MIN .VS. WORK LOAD

O. B-BY-B AVERAGES X. BAG AVERAGES *. HEATED B-BY-B AVERAGES

FIGURE 6.15 BREATH-BY-BREATH AND BAG CO2 PRODUCTION MEANS

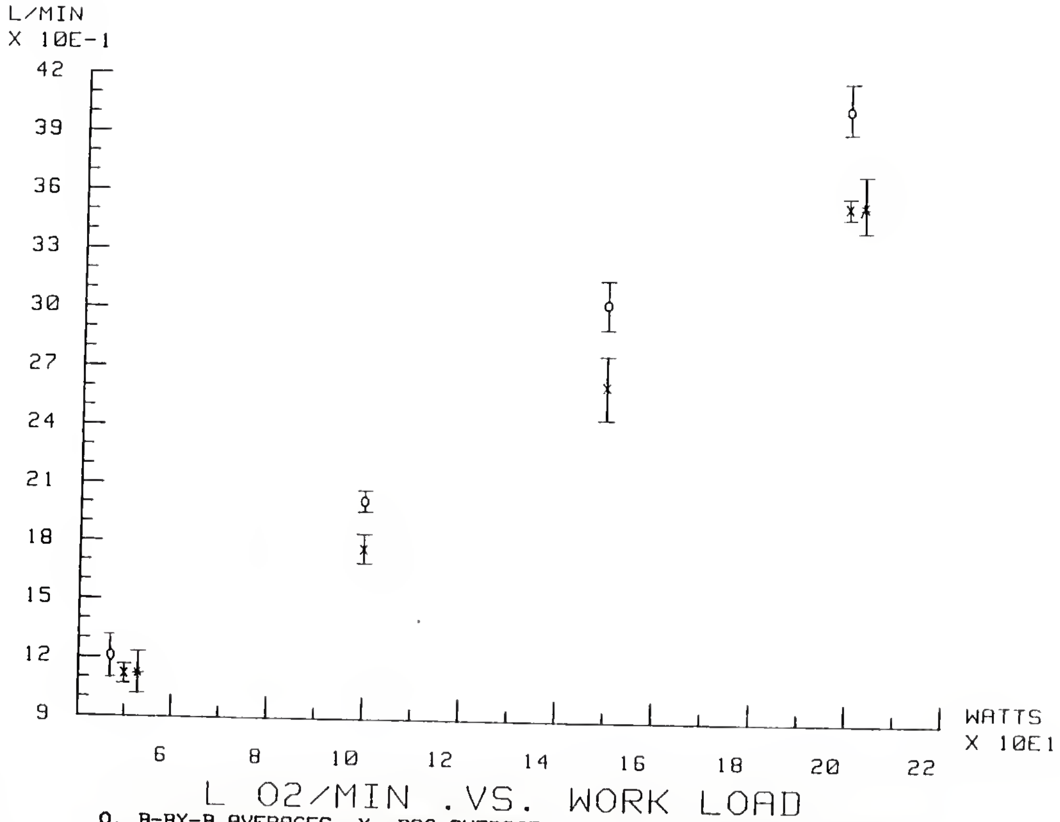


FIGURE 6.16 BREATH-BY-BREATH AND BAG O2 CONSUMPTION MEANS

VII. DISCUSSION OF RESULTS

A review of the mean and standard deviation plots (Figures 6.14-6.16) reveal several interesting results. First, at the low work level (50 watt) regardless of the degree of PTM heating, the breath-by-breath averages fell within one standard deviation of the bag collection averages. However, at the high work levels (150 and 200 watt) significant differences between the bag collection technique (heated PTM, but not close to body temperature) and the breath-by-breath studies were noted. By increasing the PTM heat, these differences were not observed.

It is felt that because the breath-by-breath analysis determines the difference between the inspired and expired oxygen volumes to compute the rate of O_2 consumption, the amount of inspired oxygen observed is actually less than what the breath-by-breath system measures with the PTM only partially heated. By increasing the PTM, the rate of O_2 consumption falls to an acceptable level while the expiratory minute volume and rate of CO_2 production remains the same as if the PTM heat was not increased. This error is not observed in the rate of CO_2 production because essentially zero percent CO_2 is inspired by the subject, thus any volume adjustments due to heating of the inspirate are of little significance to the rate of CO_2 production.

Based upon these results it is obvious that point-by-point temperature correction should be pursued. Correction of the various gas volumes on a point-by-point basis will allow for correction to BTPS (which is what this research assumes the operating condition is) or STPD. Masters' research [17] deals with the topic of point-by-point temperature correction.

The real time plots of the fractional CO_2 concentration, fractional O_2 concentration, and flow signals (Figures 6.1-6.2) show not only sample steady-state information but also transient information. By observing the envelope formed by these three signals a fairly accurate means of determining when steady-state has been reached is possible. At least, these plots clearly indicate the respiratory frequency of the subject. By further windowing of the mentioned signals, observing the changes in the various perturbations of these signals is possible, a feature that might prove invaluable when transient type analysis is performed using the breath- by-breath system.

A more obvious indicator of steady-state conditions is found in Figures 6.3-6.4. By plotting the rate of O_2 consumption on a per breath basis it is a simple matter to determine when the subject is indeed in steady-state. Again, these plots were only used to provide additional justification that the subject was in steady-state during the time analysis was performed (that time period being the last two minutes of each work load exercise).

For both the partially heated breath-by-breath studies (Figures 6.5-6.7) and the bag collection studies (Figures 6.8-6.10) precision and repeatability of both systems are obvious. It is obvious from these plots that the subject should be acclimated to the respective system before serious data collection is to commence. Good examples of this can be observed on the July 5 trial for the breath-by-breath system and the July 17 trial for the bag collection system. Both of these runs represent the first trial completed for that particular system.

Error in the data collection procedure for those initial runs is also possible. Regardless of the cause of this variance on the first trial runs, a tighter grouping of data points would almost certainly result if another trial had been run and the first trial data ignored.

VIII. CONCLUSIONS

Upon successful conversion of Creel's treadmill respiratory studies with cattle to human studies using a bicycle ergometer, the following conclusions can be made.

1. Organization of various transducers in close proximity to the DAM and HP9826 desktop computer has eliminated many of the "glitches" and calibration troubles encountered in Creel's work. Referring to Figure 3.2, an obvious attempt was made to keep coaxial cable runs found in the analog section of the breath- by-breath system as short as possible in an attempt to eliminate stray noise that might be present in the research laboratory. Not only have noise problems been reduced with this type of organization but a more repeatable system has resulted. Also, with the present system organization, system calibration can be accomplished by a single individual in about 10 minutes (this does not include temperature calibration of the thermocouple). This is in contrast to Creel's cattle research where as many as three individuals were needed to accurately calibrate the system.

2. It is felt that reorganization of the DAM control software has resulted in a more reliable data acquisition system. By paying special attention to the STS bit using 68000 assembly language code, few data acquisition problems exist. Occasionally (on the average about 1 conversion in 100000) a spike in one of the DAM channels is observed. With considerable confidence it is felt that these spikes are the result of conversion errors generated by the successive approximation converter and not due to problems in the

controlling software. The frequency of this problem does not significantly influence the accuracy of the breath-by-breath respiratory system.

3. For the types of signals encountered in these studies Gateno's DAM has functioned in an acceptable manner. Upon replacement of a faulty multiplexer, clean digitized signals were the rule. Creel encountered problems in the flow calibration procedure and it is felt that these repeated problems were due to the crosstalk that was observed between channels B (the O_2 fraction channel) and C (the flow signal channel). Occasionally the multiplexer should be checked for crosstalk by injecting four unique signals into the four DAM channels and observing the respective outputs. One major flaw in Gateno's design involves the direct connection of the 8253 timer chip select line to the read/write line. According to Intel [19], the timer chip should be selected prior to the read/write signal being set. This design flaw is responsible for problems in setting the DAM's sampling frequency that are sometimes observed when the DAM is initially turned on. This problem is usually rectified by running the routine that sets the sampling frequency a second time.

4. Temperature fluctuations and measurement play a major role in the accuracy of the breath-by-breath respiratory system. Although this research dealt very little with the issue of respiratory temperature, it has become obvious that accurate temperature measurement is essential if reasonable O_2 consumption values at the higher work loads are to be obtained. Masters' work [17] deals with the issue of respiratory

temperature measurement in detail.

5. The HP9826 desktop computer is well suited for controlling the DAM and data analysis routines. With the help of the PASCAL compiler and 68000 assembler [20-22], generation of relatively fast and simple control software is possible. By acquiring more RAM for the HP9826 (it presently has slightly more that 0.5 Mbytes of RAM) longer real time data collection would be possible for a given sampling frequency. The data filing scheme used for compatability between the HP9826 PASCAL and BASIC operating systems is probably the best solution to the problem considering the control and analysis software as it presently exists. Understanding full well the problems that would be encountered having the analysis routine written in PASCAL, the present filing scheme could be scrapped, the collected data simply being stored as binary data files (if ANALYSIS was translated to PASCAL). This action would significantly reduce the time required to process the respiratory data.

6. Comparisons made between the breath-by-breath respiratory system and the bag collection technique suggest that the breath-by-breath system is accurate over several work loads provided steady-state respiration is reached and respiratory temperature adjustments and/or measurements are made. Noting that the bag collection technique does not have the ability to measure transient changes in respiration and that the breath-by-breath system developed is just as accurate as the bag collection technique during steady-state exercise, the breath-by-breath system seems to provide more potential for

cardio-pulmonary studies. In terms of the exercise programs described in this report, the breath-by-breath system is a much simpler system to operate (as compared to the bag collection technique) and requires fewer people to operate it. Because the breath-by-breath system is accurate at steady-state, the assumption can be made that it is equally accurate in transient type measurements and can be used in a series of exercise studies to measure transient respiratory phenomenon.

7. The addition of the windowing feature in both the plotting of the respiratory data and the analysis of that data have proven to be invaluable, even in the steady-state analysis performed in this research. Using the windowing feature, plots can be expanded to any desired time base, a feature the typical chart recorder is incapable of providing. The real power of the windowing ability will manifest itself in transient respiratory measurements.

IX. IMPENDING RESEARCH

Although the computer-controlled instrumentation system is functioning as well as the accepted bag collection technique, additions to the system (both hardware and software) would further enhance the breath-by-breath system. Following are a few recommended system changes and additions.

1. As the system is presently organized, only four of the DAM channels are used (the four channels being CO_2 and O_2 fractional concentrations, respiratory flow, and respiratory temperature). By adding additional control software to the data acquisition program (DAP.CODE) and making the necessary additions to the calibration routine (CAP.CODE) as many as eight analog channels could be monitored simultaneously. Additional signals of interest might include body temperature, heart rate, blood pressure, and PTM temperature.

2. The ability to evaluate the respiratory data based upon inter-breath changes in the Functional Residual Capacity (FRC) should be added to the existing analysis routine. These additions would allow for the determination of alveolar O_2 and CO_2 gas exchange volumes using the gas fractional concentrations currently being measured at the mouth (Beaver [4]).

3. Although the existing file compatability scheme for the PASCAL and BASIC operating systems represents the best solution to this problem, by rewriting the analysis routine (ANALYSIS) in PASCAL no ASCII to Binary DATa (BDAT) file conversion would be necessary. This would allow both the data and calibration files to be stored initially as BDAT files, which the analysis routine (now written in PASCAL) could read directly. This would

eliminate the need for both of the file compaction routines (CAPCRUNCH and DAPCRUNCH) and the data handling time would be minimized. Rewriting ANALYSIS in PASCAL would, however, destroy the user friendliness that is prevalent in the BASIC operating system. The rewriting of ANALYSIS should occur only after all major additions to the analysis routines have been made.

4. Once the desired system enhancements have been made, well defined exercise programs for the purpose of respiratory research should be conducted. Of particular interest in this research would be the study of transient phenomena that occur both at the onset of exercise and the onset of rest and/or another exercise level.

5. Reapplication of this research to the cattle research being conducted by the Department of Anatomy and Physiology should be made. It is felt that by careful duplication and/or transfer of the system organization, hardware, and software, little (if any) effort to obtain an accurate, easy to use system would be necessary.

6. Further system changes or additions proposed by those who use the breath- by-breath system on a regular basis should also be seriously considered.

X. BIBLIOGRAPHY

[1] Gregg Ruppel, Manual of pulmonary function testing, St. Louis, The C.V. Mosby Company, 1979.

[2] D.H. Paterson, D.A. Cunningham, and A. Donner, "The effect of different treadmill speeds on the variability of VO2 max in children", Eur. J. Appl. Physiol., Springer-Verlag, vol. 47, pp. 113-122, 1981.

[3] R.L. Hughson and J.M. Kowalchuk, "Influence of diet on CO2 production and ventilation in constant-load exercise", Respiration Physiology, vol. 46, pp. 149-160, 1981.

[4] W.L. Beaver, N. Lamarra, and K. Wasserman, "Breath-by-breath measurement of true alveolar gas exchange", J. Appl. Physiol., vol. 51(6), pp 1662-1675, 1981.

[5] I.E. Sodal, G.D. Swanson, A.J. Micco, F. Sprague, and D.G. Ellis, "A computerized mass spectrometer and flowmeter system for respiratory gas measurements", Annals of Biomedical Engineering, vol. 11, pp 83-99, 1983.

[6] A.C. Norton, "Development and testing of a microprocessor-controlled system for measurement of gas exchange and related variables in man during rest and exercise", Beckman Instrument, Inc., Physiological Measurements Operations, Anaheim, 1982.

[7] R. Arieli, and H.D. Van Liew, "Corrections for response time and delay", a manuscript written for J. Appl. Physiol., 1981.

[8] M.P. Hlastala, B. Wranne, and C.J. Lenfant, "Cyclical variations in FRC and other respiratory variables in resting man", J. Appl. Physiol., vol. 34(5), pp 670-676, 1973.

[9] J.A. Hirsch, and B. Bishop, "Human breathing patterns on mouthpiece or face mask during air, CO2, or low O2", J. Appl. Physiol., vol. 53(5), pp 1281-1290, 1982.

[10] J.H.T. Bates, G.K. Prisk, T.E. Tanner, and A.E. McKinnon, "Correcting for the dynamic response of a respiratory mass spectrometer", J. Appl. Physiol., vol. 55(3), pp 1015-1022, 1983.

[11] Earl E. Creel, "Measurement of breath-by-breath oxygen consumption and carbon dioxide production in exercising calves", Master's thesis, Kansas State University, 1982.

[12] Pascal 2.1 procedure library user's manual for HP series 200 computers, Fort Collins, Hewlett-Packard Desktop Computer Division, 1983.

[13] Leon W. Gateno, "Design of versatile, multi-channeled, data acquisition module", Master's thesis, Kansas State University, 1982.

[14] Perkin-Elmer MGA-1100 medical gas analyzer operations and maintenance manual, Pomona, California, Perkin-Elmer Medical Instruments.

[15] Godart pneumotachograph provisional instruction manual, Holland, Godart-Statham B.V..

[16] Hans Rudolph respiratory apparatus catalogue, Kansas City, ch. 11, 1983.

[17] Mike Masters, personal communication on "A respiratory temperature transducer for studying the effects of respiratory temperature on the breath-by-breath measurement of respiratory function", Kansas State University, 1984.

[18] Per-Olof Astrand, M.D., Work tests with the bicycle, ergometer, Varberg, Sweden, Monark-Crescent AB.

[19] Intel component data catalog 1980, Santa Clara, Intel Corporation, pp 8-50.

[20] The Pascal handbook for the series 200 computers, Fort Collins, Hewlett-Packard Desktop Computer Division, 1983.

[21] BASIC programming techniques for the HP9826 and 9836 computers, Fort Collins, Hewlett-Packard Desktop Computer Division, 1983.

[22] G. Michael Schneider, Steven W. Weingart, and David M. Perlman, An introduction to programming and problem solving with Pascal, John Wiley & Sons, 1982.

[23] Analog devices 1984 databook, Analog Devices, vol. I, pp 16-1.

[24] Jeanne Agnew and Robert C. Knapp, Linear algebra with applications, Monterey, California, Brooks/Cole Publishing Company, 1978.

[25] Pascal 2.1 user's manual for HP series 200 computers, Fort Collins, Hewlett-Packard Desktop Computer Division, 1983.

ACKNOWLEDGEMENTS

I would like to thank my committe members Dr. Michael S.P. Lucas, Dr. Marion Roger Fedde, and Dr. Howard H. Erickson for their help and suggestions. A special thanks goes to my Major Professor Dr. Richard R. Gallagher for his support and guidance throughout the course of this research.

Thanks also goes to Chris "Duffer" Duffey for not only assisting me in the flawless graphics found throughout this thesis but also for being a professional colleague and trustworthy friend.

My most favorite running partners Laura Parker and Roy Peters deserve recognition for training with me even during those times when I was nothing less than fanatical about my running.

I dedicate this work to my parents Loren Sr. and Rosella, my brother Carl, and my sisters Kathy, Carol, and Laura who provided the emotional support throughout my college career which made it all possible. Without their belief in my abilities as a person as well as an engineer the completion of both my BS and MS degrees would have been left to chance.

APPENDIX I

Mass Storage Management

One of the most critical operations a system user must perform is mass storage management. With the large volume of data that is processed by the system routines, lack of mass storage organization could result in the loss of several trials worth of data. Following is the procedure one should use to monitor and maintain the two main mass storage devices in the breath-by-breath respiratory system, namely the HP9134A hard disk memory and the HP9895A 8" flexible disk memory.

1. Referring to Figure A1.1, turn on the DECwriter II printer, DAM, Tektronix TM power supply, HP9895A 8" flexible disk memory, HP9134A hard disk memory, and HP2673A thermal printer (in that order).

2. Place the 5.25" floppy disk labeled "Pascal 2.1 System, Boot:" in the computer's (HP9826) disk drive (label side up) and turn on the HP9826. The Pascal operating system will automatically be loaded and initialized to accommodate all peripherals in the instrumentation system.

3. Once a majority of the initialization is complete, the system date is requested and should be entered as DD-MON-YR. As is the case with all data input in the Pascal system, desired input is typed via the computer keyboard and information is accepted by pressing the "ENTER" key.

A. WORK TABLE
B. DECwriter II PRINTER
C. TEMPERATURE CALIBRATION WATER BATHS
D. ZERO SUPPRESSION BOX
E. PERKIN-ELMER GASS MASS SPECTROMETER
F. TEKTRONIX D41 OSCILLOSCOPE
G. HP2673A THERMAL PRINTER
H. HP9826 COMPUTER
I. HP9134A HARD DISK MEMORY
J. HP9895A 8" FLEXIBLE DISK MEMORY
K. TEKTRONIX TM MODULE
L. GPIO INTERFACE TESTER
M. HP9872C PLOTTER
N. DATA ACQUISITION MODULE
O. GODART PNEUMOTACHOMETER
P. HARVARD RESPIRATOR AND FITTINGS
Q. ERGOMETER REVOLUTION COUNTER
R. METRONOME
S. BOTTLED GAS (12.9% O2-7% CO2)
T. BICYCLE ERGOMETER
U. PNEUMOTACH HEATER
V. HEART RATE MONITOR
W. SUPPORT FOR MASKS, FLEISCH HEADS, AND METEOROLOGICAL BALLOON
X. SPIROMETER

KEY FOR FIGURE A1.1

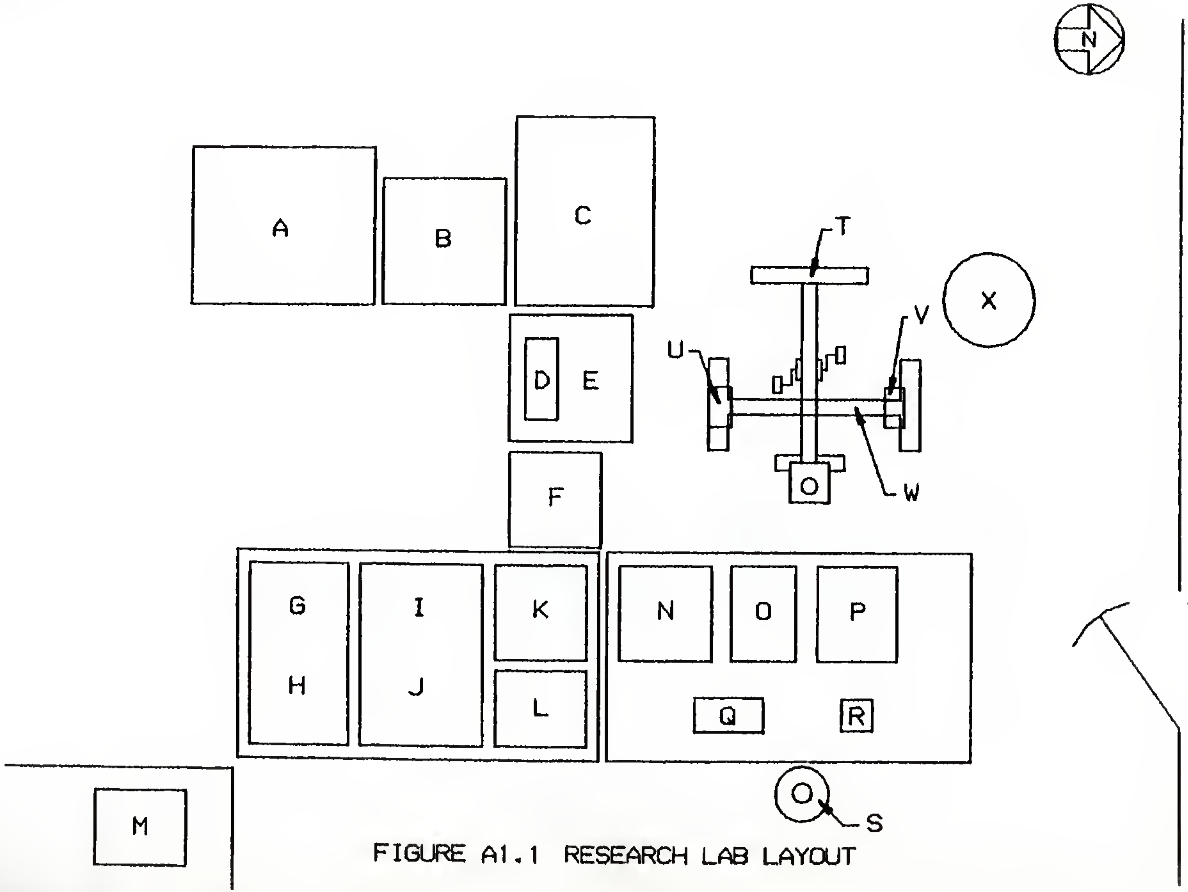


FIGURE A1.1 RESEARCH LAB LAYOUT

4. The time (24 hour) should then be entered as HH:MM:SS. This step completes the Pascal system initialization.

5. To perform standard mass storage operations (i.e. directory listings, directory crunching, etc.) in the PASCAL system, the system filer must be loaded into memory. This is accomplished by typing "F" (Filer) at the system command level.

6. Once the system filer is loaded into memory, execution of FILER begins. The operator will first notice that the menu at the top of the CRT is different. The FILER has its own menu, seperate from that of the main command level. The first operation the user should perform is to list the volumes (disks, printers, etc.) that are currently on-line. Typing "V" (Volumes) will cause FILER to display this information.

7. Of the volumes listed, only four are of interest to the system manager. Following is a table of the volumes of interest.

PASCAL Volume #	BASIC Volume	Description
#7	:HP9895,700,0	HP9895A 8" flexible disk
#12	:HP9895,702,1	HP9134A hard disk #1
#13	:HP9895,702,2	HP9134A hard disk #2
#14	:HP9895,702,3	HP9134A hard disk #3

Volume #7 is used exclusively for storing DAM and calibration binary data files. The 8" flexible disk was selected for these binary data files because of its portability and the possibility of multiple 8" flexible disks (leading to an unlimited amount of data storage capabilities).

Volume #12 currently contains all the DAM and calibration binary data files accumulated during the calf respiratory research conducted during the summer of 1983.

Volume #13 contains copies of the system software and is the volume where the ASCII files created by CAP.CODE and DAP.CODE reside prior to conversion to binary data files (see Appendicies V and VI for more information).

Volume #14 contains many of the system files necessary for the PASCAL operating system to function. It is the system volume and can be referenced either as "#14" or "*" ("*" designates the system volume).

8. For volume #7 (the 8" flexible disk memory system) there are several aspects of mass storage of which the system operator should be aware. The procedure most often performed on volume #7 is simply to determine how much room is available for data storage on the current 8" flexible disk. This can be accomplished by pressing "L" (List) followed by "#7" when asked for the volume number. Pressing enter causes the directory of

volume #7 to be displayed. (The user may have to press the space bar to continue long directory listings.) At the end of the directory listing a summary of the amount of mass storage used, the amount that is unused, and the largest unused space.

9. Depending on how many unused blocks remain will dictate when a new 8" flexible disk should be used. (As an example, a binary data file containing 24000 data points requires 189 blocks of mass storage. A binary calibration file requires only two blocks.) Should a new 8" disk be required, it will have to be initialized before being used. Disk initialization should be done in the BASIC operating environment. To initialize a disk place the 5.25" disk labeled "HP9826 Data Analysis Routines" in the disk drive and turn the HP9826 off and then back on. The BASIC operating system will be loaded and "AUTOST", the BASIC menu select routine, will be loaded into memory. The user should place the 8" disk to be initialized in the HP9895A 8" drive and select item 2 in the "MSI" option of the menu select routine. The menu select routine should then be stopped by pressing "PAUSE". Typing 'INITIALIZE ":HP9895,700,0"' followed by the "EXECUTE" key will cause the 8" disk to be initialized (initialization will take several minutes). Once initialization is complete, the "*" in the lower right hand corner of the HP9826 CRT will be removed and pressing "CONTINUE" will restart "AUTOST". Selecting the "CAT" option in the menu will now display the blank

directory of the newly initialized 8" disk.

10. As is the case with all mass storage volumes, volume #7 can be compacted so that the number of unused blocks corresponds to the largest unused space on the disk. This is accomplished by typing "K" (Krunch) while the filer is running followed by "#7" when asked for the volume number. Pressing enter causes FILER to ask the user if directory B9826 should be compacted. Answering "Y" to this question causes the compaction of volume #7 to begin. Once this compaction routine begins, FILER advises the user not to touch any of the computer equipment until the compaction procedure is complete as any action could result in the loss of some files.

11. For volume #13 (the HP9134A hard disk #2), space considerations should never be a problem provided no additional programs are stored on #13. (Even with four 24000 point ASCII files stored on volume #13 enough space remains for over 1400 ASCII calibration files.) The user is reminded, however, that the ASCII data files created by DAP.CODE must be crunched after each data collection session. (See Appendix VI for more details.)

12. Volumes #12 and #14 (HP9134A hard disks #1 and #3 respectively) do not require directory crunching or initialization as these mass storage devices are fairly static storage areas (see step 7 above for explanation). To perform directory listings, directory crunching, or disk initialization on these volumes refer to the

preceding steps used for volume #7.

APPENDIX II

Dam Offset and Gain Adjustments

Following is the step-by-step procedure to perform offset and gain adjustments on the DAM built by Gateno [13]. These adjustments are made only when analog signal gain adjustments are required or when the operator feels that significant offset error exists in the DAM. Upon initial adjustments, these adjustments should need to be made only once or twice every two to three months.

1. Referring to Figure A1.1, turn on the DECwriter II printer, DAM, Tektronix TM power supply, HP9895A 8" flexible disk memory, HP9134A hard disk memory, and HP2673A thermal printer (in that order).

2. Turn on the HP9826 computer. Once the HP9826 has completed its own internal tests, the user should type "WRITEIO 12,4;0". followed by the "EXECUTE" key. This places the DAM sample and hold amplifiers in the tracking mode so that offset adjustments can be made.

3. Referring to Figure A2.1, to adjust the offset in the AD521 differential amplifiers, the gain adjustment pots on the DAM should be turned fully counterclockwise. This sets the maximum gains possible for the AD521's.

4. With the inputs to the DAM grounded and measuring the output voltage on pin 7 of the AD521, adjust the amplifier multiturn potentiometer so the output voltage is a minimum. Perform this adjustment for

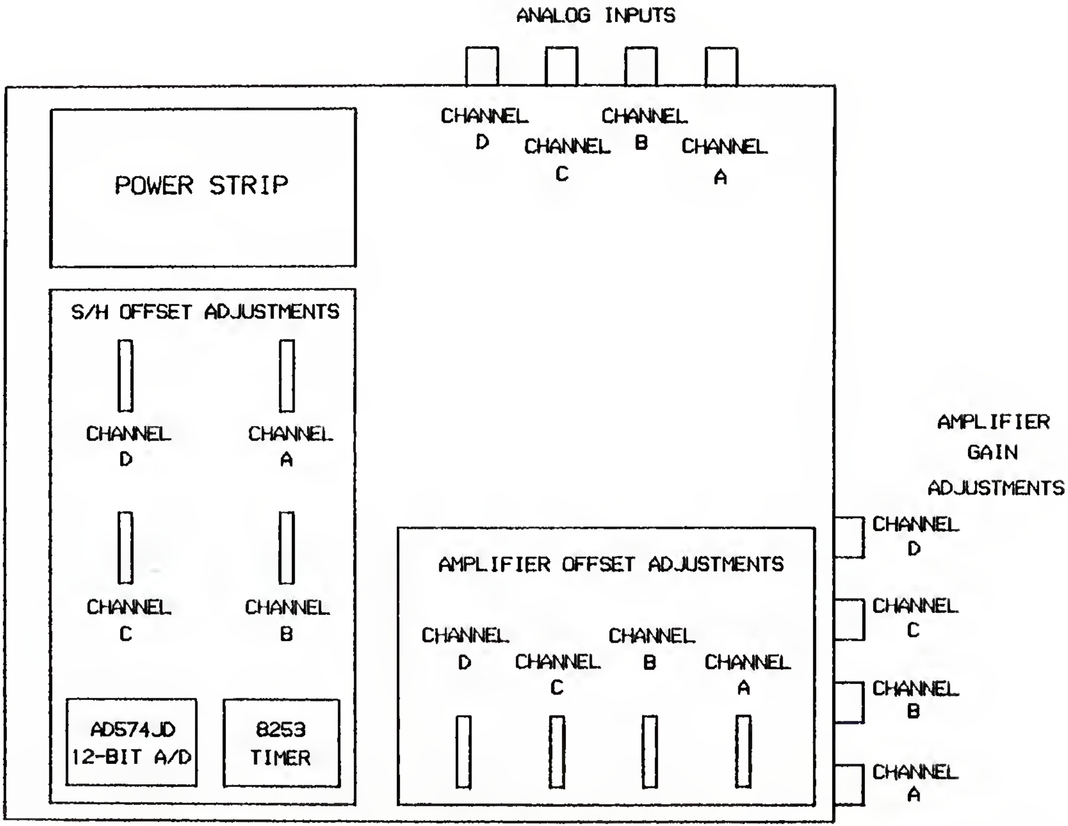


FIGURE A2.1 DAM OFFSET AND GAIN ADJUSTMENTS LOCATIONS all four channels. This completes the amplifier offset adjustments.

5. With the inputs to the DAM still grounded and measuring the output voltage on pin 8 of the AD582 sample and hold amplifiers, adjust the AD582's multiturn potentiometer so the output voltage is a minimum. Perform this adjustment on all four channels. This completes all DAM offset adjustments.

6. To set the gain adjustment potentiometers on the DAM, place the 5.25" disk labeled "Pascal 2.1 System, Boot:" in the computer's (HP9826) disk drive (label side up). Turn the computer off and then back on. The Pascal operating system will automatically be loaded and initialized to accommodate all peripherals in the instrumentation system.

7. Once a majority of the initialization is complete, the system date is requested and should be entered as DD-MON-YR. As is the case with all data input in the Pascal system, desired input is typed via the computer keyboard and information is accepted by pressing the "ENTER" key.

8. The time (24 hour) should then be entered as HH:MM:SS. This step completes the Pascal system initialization.

9. To set the proper DAM gains, the program DAP.CODE (<u>D</u>ata <u>A</u>cquisition <u>P</u>rogram) must be run. This is

accomplished by pressing the "R" key (Run) and entering DAP for the program name.

10. Once the program is loaded into memory, execution of DAP.CODE begins. The user is first asked to enter the desired sampling frequency. Typically a 50 Hz sampling rate is used; however, faster (to 350 Hz) and slower rates are allowed.

11. The number of samples per channel is then entered. For the majority of the DAM gain adjustments, DC input signals are used. Thus, 400 data points per channel is sufficient.

12. The user is then prompted to press the "ENTER" key to continue. At this point, the desired input signals should be applied to the DAM inputs. Referring to the following table, apply the appropriate input signal to obtain either a maximum or minimum for the channel in question.

Channel	Input	Acceptable binary value
A	0% CO_2	80-200
A	7% CO_2	3800-4000
B	21% O_2	3800-4000
B	13% O_2	80-200
C	Relaxed breathing	1300-1500 (minimum)
		2500-2800 (maximum)
C	Harvard respirator	80-200 (minimum)
		3800-4000 (maximum)
D	30 deg C	80-200
D	35 deg C	1990-2100
D	40 deg C	3800-4000

13. By adjusting the DAM gain potentiometers along with the zero suppression box for channel A and B DC offset levels, the mentioned binary values can be obtained. To rerun DAP.CODE after gain and offset adjustments are made, press the "CLR I/O" key on the HP9826 followed by "U" (User restart). Steps 10 through 13 should be repeated until acceptable binary values are obtained.

14. All DAM gain and offset adjustments are complete (as well as adjustment of the zero suppression box). System calibration may now be performed on the instrumentation system.

APPENDIX III

System Calibration Procedure

Following is the step-by-step procedure to calibrate the four channel instrumentation system.

1. Referring to Figure A1.1, turn on the DECwriter II printer, Perkin-Elmer gas mass spectrometer, DAM, Tektronix TM power supply, HP9895A 8" flexible disk memory, HP9134A hard disk memory, and HP2673A thermal printer (in that order).

2. Place the 5.25" floppy disk labeled "Pascal 2.1 System, Boot:" in the computer's (HP9826) disk drive (label side up) and turn on the HP9826. The Pascal operating system will automatically be loaded and initialized to accommodate all peripherals in the instrumentation system. (NOTE: If the HP9826 computer is on before the system boot disk is placed in the disk drive, turn the HP9826 off and then back on. This allows the Pascal operating system to be automatically loaded and executed.)

3. Once a majority of the initialization is complete, the system date is requested and should be entered as DD-MON-YR. As is the case with all data input in the Pascal system, desired input is typed via the computer keyboard and information is accepted by pressing the "ENTER" key.

4. The time (24 hour) should then be entered as

HH:MM:SS. This step completes the Pascal system initialization.

5. To calibrate the four data acquisition channels, the program CAP.CODE (CAlibration Program) must be run. This is accomplished by pressing the "R" key (Run) and entering CAP for the program name.

6. Once the program is loaded into memory, execution of CAP.CODE begins. The user is first asked to enter the desired sampling frequency. Typically a 50 Hz sampling rate is used; however, faster (to 350 Hz) and slower rates are allowed.

7. The user is asked if the fractional concentration signal should be calibrated. If the operator answers "N" to this question CAP.CODE jumps to the flow signal calibration procedure (see step 15 below).

8. If the fractional gas concentration signal is to be calibrated, the operator is prompted to connect the mass spectrometer probe to room air. (NOTE: The mass spectrometer's "ON" switch should be depressed while data is being collected. The "STANDBY" mode of operation should be selected otherwise. Also, make sure inlet port 1 is selected.) By removing the screened cap on the sampling capillary, room air is sampled by the mass spectrometer.

9. The user is asked to enter the actual O_2 concentration of the sampled air. This value is read directly from the mass spectrometer's left most digital readout. The O_2 concentration (as well as all other fractional concentrations) should be entered as a fractional value less than unity. Thus, if the left most display reads 20.9% the value entered for O_2 concentration should be .209.

10. Once the O_2 concentration is entered, the program prompts the operator to press "ENTER" to continue. This pause allows the user to make any final adjustments to the instrumentation prior to sampling of the mass spectrometer.

11. One thousand data points are taken and averaged for the O_2 and CO_2 channels (room air is assumed to be 0% CO_2). The CO_2 DC offset (binary value read for 0% CO_2), average value read for 0% CO_2 (same as CO_2 DC offset), and average value read for 21% O_2 are then displayed.

12. The operator is instructed to connect the Gas Mass Spectrometer (GMS) probe to 7% CO_2 and 13% O_2. These gas concentrations allow for calibration on the upper CO_2 levels and calibration on the lower O_2 levels. By placing the GMS probe in the gas delivery port leading from the calibrated gas cylinder and opening the main valve on the cylinder, the desired gas concentrations are available for calibration.

13. The fractional gas concentrations of CO_2 and O_2 are then entered. The CO_2 concentration is read from the right most digital readout on the GMS. Again, these values should be entered as fractional quantities less than one.

14. Once the CO_2 and O_2 concentrations are entered, 1000 data points are collected and averaged for the two channels. The O_2 DC offset (binary value read for 13% O_2), average value read for 7% CO_2, and average value read for 13% O_2 (same as O_2 DC offset) are displayed.

15. The user then has the option of calibrating the flow signal. Should the operator enter "N" for the flow calibration prompt, the flow calibration procedure will not be executed and CAP.CODE will jump to the temperature calibration routine (refer to step 22). Any input besides "N" will cause execution of the flow calibration section.

16. CAP.CODE then instructs the operator to connect zero flow to the pneumotach. This is accomplished by placing the mask and Fleisch head inside the pneumotachometer auto-zeroing box and closing the box. (NOTE: The Godart/Fleisch assembly should be zeroed prior to its use. This is accomplished by placing the Godart in the V [Volume] mode, forcing air through the Fleisch head until the Godart meter reads near midrange, and adjusting the zero balance control [while the head is in the auto-zeroing box] until no meter movement is

observed. The Godart should then be placed back in the $\dot{V}$ [flow] mode.)

17. CAP.CODE pauses at this point to allow the operator to perform the mentioned task. Pressing "ENTER" causes CAP.CODE to acquire 1000 data points for zero flow and the average binary for zero flow is determined and display on the HP9826 CRT.

18. The Harvard Respirator should then be connected to the pneumotach via the custom fittings that are available. The pump should be turned on and the highest respirator frequency selected.

19. Approximately 5 minutes should expire before the user presses the "ENTER" key to begin data acquisition. This allows an equilibrium to be reached between the pump, pneumotach, and surroundings.

20. Four thousand data points will then be acquired. The respirator may be turned off once the data collection complete prompt is displayed on the CRT. Once data collection is complete, CAP.CODE performs a series of integrations on the flow signal to determine an inspiratory and expiratory flow calibration value (see Appendix VIII for more details). As these integrations are performed, CAP.CODE displays the breath number followed by the inspiratory and expiratory integration values. If the system is functioning properly these inspiratory and expiratory values should not vary

significantly from breath-to-breath. Thus, observing these values can help in isolating system problems at the flow calibration stage.

21. Binary zero flow (binary value corresponding to zero flow), inspiratory flow calibration factor, and expiratory flow calibration factor are then displayed on the HP9826 CRT. The flow calibration procedure is complete.

22. The user is then asked whether or not the flow temperature instrumentation is to be calibrated. Answering "N" to this question causes CAP.CODE to jump to the calibration factor storage procedure (see step 26). Any other response to this question causes the temperature calibration procedure to be initiated.

23. Provided the temperature calibration procedure is requested, the operator is instructed to place the thermocouple in the lowest temperature water bath. The actual water bath temperature is then entered (deg C). This temperature is obtained using the precision mercury thermometer provided.

24. One thousand temperature data points are then collected and averaged. The average binary value for the low temperature is displayed on the CRT and the operator is instructed to place the thermocouple in the middle temperature water bath.

25. Steps 23 and 24 are repeated for both the middle and high temperature water baths. CAP.CODE then performs a 2nd order fit of these three data points and the polynomial coefficients for this fit are displayed.

26. The user is asked if the calibration factors computed should be stored. Answering "N" to this question terminates CAP.CODE. Any other answer causes CAP.CODE to prompt the user for the calibration file name and calibration date. Following are examples of appropriate calibration file names.

File Name	Description
CAL618	Calibration file created on June 18
CL2618	Second calibration file created on June 18

These file names are only suggestions. They (the names) were selected for their descriptive nature. (NOTE: Pascal file names in excess of 9 characters should not be used.)

27. Once this information is supplied, the previously mentioned calibration factors are converted to ASCII and stored on the HP9134A hard disk. CAP.CODE then ends.

28. This ASCII data file should then be crunched (converted to BASIC BDAT files) before using it in the data analysis routine. See Appendix V for more details.

APPENDIX IV

System Collection of Breath-by-breath Respiratory Data

Following is the step-by-step procedure to collect respiratory data using the current instrumentation system.

1. Referring to Figure A1.1, turn on the DECwriter II printer, Perkin-Elmer gas mass spectrometer, DAM, Tektronix TM power HP9895A 8" flexible disk memory, HP9134A hard disk memory, and HP2673A thermal printer (in that order).

2. Place the 5.25" floppy disk labeled "Pascal 2.1 System, Boot:" in the computer's (HP9826) disk drive (label side up) and turn on the HP9826. The Pascal operating system will automatically be loaded and initialized to accommodate all peripherals in the instrumentation system. (NOTE: If the HP9826 computer is on before the system boot disk is placed in the disk drive, turn its (the computer's) power off and then back on. This allows the Pascal system to be loaded automatically.)

3. Once a majority of the initialization is complete, the system date is requested and should be entered as DD-MON-YR. As is the case with all data input in the Pascal system, desired input is typed via the computer keyboard and information is accepted by pressing the "ENTER" key.

4. The time (24 hour) should then be entered as

HH:MM:SS. This step completes the Pascal system initialization.

5. To collect four channels of respiratory data, the program DAP.CODE (Data Acquistion Program) must be run. This is accomplished by pressing the "R" key (Run) and entering DAP for the program name. In the steps that follow, if an error in data entry is made, typing the "SHIFT" and "STOP" keys simultaneously returns the system to the Pascal command level.

6. Once the program is loaded into memory, execution of DAP.CODE begins. The user is first asked to enter the desired sampling frequency. Typically a 50 Hz sampling rate is used; however, faster (to 350 Hz) and slower rates are allowed. It should be remembered that a maximum of 24000 data points per channel is possible with the present data collection system, so fast sampling rates significantly limit data collection time.

7. The number of samples per channel is then entered. As previously mentioned, 1 to 24000 data points per channel can be taken. For example, if a 5 minute data collection period is desired at a sampling frequency of 50 Hz, 15000 data points per channel should be taken.

8. The user is then prompted to press the "ENTER" key to continue. Pressing "ENTER" begins the actual data acquisition process. This pause was implemented to allow the operator(s) of the system to make last minute

adjustments to the system before acquiring data.

9. Once the four channels of data are collected and loaded into HP9826 memory, maximum and minimum values for the CO_2, O_2, flow, and temperature channels are determined and displayed on the HP9826 CRT. Pressing "ENTER" following this display initiates the data storage procedure.

10. The acquired binary data are converted to ASCII data and stored on the HP9134A hard disk memory by the data storage procedure. Data conversion to ASCII is necessary to allow for file compatability between the Pascal and BASIC operating systems.

11. The data are also displayed on the HP9826 CRT. Because a 12-bit A-to-D converter is used, binary representations ranging in values from 0 to 4095 are possible. By observing the displayed values, the operator can determine if saturation of the input signals has occurred.

12. These ASCII data files should then be crunched (converted to BASIC BDAT (Binary DATa) files) before any other exercise trials are conducted. Failure to crunch the data prior to another exercise run will result in the loss of the first trial's data (see Appendix VI for more details).

13. At this point, to execute the crunch routine on

the respiratory data, go to Appendix VI, item 2.

APPENDIX V

Pascal to BASIC File Conversion of Calibration Data

Following is the procedure to crunch (convert from ASCII to binary data) files created by CAP.CODE.

1. Referring to Figure A1.1, turn on the DECwriter II printer, DAM, Tektronix TM power supply, HP9895A 8" flexible disk memory, HP9134A hard disk memory, and HP2673A thermal printer (in that order).

2. Place the 5.25" floppy disk labeled "HP9826 Data Analysis Routines" in the computer's (HP9826) disk drive (label side up) and turn on the HP9826. A BASIC autostart routine (AUTOST) is automatically loaded and initiated.

3. To start the calibration file compaction routine (CAPCRUNCH) press the special function key "k0" on the HP9826. CAPCRUNCH will then be loaded into memory from the HP9134A hard disk and executed. Once CAPCRUNCH is running, the user must press key "k0" again to continue or press key "k9" to exit back to AUTOST. This extra check allows for accidental execution of CAPCRUNCH.

4. Once key "k0" is pressed, the user is asked to enter the name of the calibration file to crunch. As in the Pascal operating system, data input is typed via the computer keyboard and information is accepted by pressing the "ENTER" key.

5. Upon receipt of the calibration file name, CAPCRUNCH searches the HP9134A hard disk for the calibration file. If the calibration file is found, it is loaded into memory and converted to binary data.

6. If the calibration file is not found an error message will be displayed and CAPCRUNCH will be aborted. The "RUN" key should be pressed followed by key "k9" to return back to AUTOST.

7. Assuming the calibration file does exist, the converted calibration file is stored on the HP9895A 8" flexible disk and the ASCII version of the calibration file is deleted from the hard disk.

8. Following compaction of the calibration file, an operator's message indicating that file compaction is complete is displayed. CAPCRUNCH loads the autostart routine from the 5.25" floppy disk (HP9826 Data Analysis Routines diskette) and AUTOST is executed.

APPENDIX VI

Pascal to BASIC File Conversion of Respiratory Data

Following is the procedure to crunch (convert from ASCII to binary data) files created by DAP.CODE.

1. Referring to Figure A1.1, turn on the DECwriter II printer, DAM, Tektronix TM power supply, HP9895A 8" flexible disk memory, HP9134A hard disk memory, and HP2673A thermal printer (in that order).

2. Place the 5.25" floppy disk labeled "HP9826 Data Analysis Routines" in the computer's (HP9826) disk drive (label side up) and turn on the HP9826. A BASIC autostart routine (AUTOST) is automatically loaded and initiated. (NOTE: If the HP9826 computer is on prior to placing the data analysis disk in the disk drive, turn off the HP9826 and then turn it back on. This allows AUTOST to be automatically loaded and executed.)

3. To start the data file compaction routine (DAPCRUNCH) press the special function key "k1" on the HP9826. DAPCRUNCH will then be loaded into memory from the HP9134A hard disk and executed. Once DAPCRUNCH is running, the user must press key "k1" again to continue or press key "k9" to exit back to AUTOST. This extra check allows for accidental execution of DAPCRUNCH.

4. Once key "k1" is pressed, the user is asked to enter the number of data points (per channel) to crunch. As in the Pascal operating system, data input is typed

via the computer keyboard and information is accepted by pressing the "ENTER" key.

5. Upon receipt of the number of data points, DAPCRUNCH searches the HP9134A hard disk for the ASCII CO_2 file MONSTER1.ASC. If MONSTER1.ASC is found, it is loaded into memory and converted to binary data. The user is then prompted for the name of the file to contain the binary data. Following are examples of appropriate file names.

File Name	Description
C50618	C - CO_2 file, 50 - 50 Hz sampling, 618 - June 18 collection date
O50618	O - O_2 file, 50 - 50 Hz sampling, 618 - June 18 collection date
V50618	V - flow file, 50 - 50 Hz sampling, 618 - June 18 collection date
T50618	T - temperature file, 50 - 50 Hz sampling, 618 - June 18 collection date
C2618	C - CO_2 file, 2 - second run of of the day, 618 - June 18 collection date
C3618	C - CO_2 file, 3 - third run of the day, 618 - June 18 collection date

These file names are only suggestions. They were selected because they portray (at a glance) information about the data contained in the named files. (NOTE: BASIC file names should not exceed 10 characters in length.)

6. If MONSTER1.ASC is not found, ASCII CO_2 data does not exist and DAPCRUNCH will be aborted. The "RUN"

key should be pressed followed by key "k9" to return back to AUTOST.

7. Assuming MONSTER1.ASC does exist and the name of the file to contain the binary data has been entered, the converted data will be stored on the HP9895A 8" flexible disk and MONSTER1.ASC will be deleted from hard disk.

8. In a similar manner the O_2 file (MONSTER2.ASC), flow file (MONSTER3.ASC), and temperature file (MONSTER4.ASC) are crunched.

9. Following compaction of the four ASCII files, an operator's message indicating that file compaction is complete is displayed. DAPCRUNCH loads the autostart routine from the 5.25" floppy (HP9826 Data Analysis Routines diskette) and AUTOST is executed.

10. To immediately analyze the crunched data, go to Appendix VII, item 3.

APPENDIX VII

System Analysis of Breath-by-breath Respiratory Data

Following is the step-by-step procedure to analyze the respiratory data on a breath-by-breath basis. Data files to be used by this analysis routine must have been crunched by DAPCRUNCH prior to their use. Calibration files must have been crunched by CAPCRUNCH prior to use in the analysis routine. (See Appendices V and VI for more information.)

1. Referring to Figure A1.1, turn on the DECwriter II printer, DAM, Tektronix TM power supply, HP9895A 8" flexible disk memory, HP9134A hard disk memory, HP2673A thermal printer, and HP9872C plotter (in that order).

2. Place the 5.25" floppy disk labeled "HP9826 Data Analysis Routines" in the computer's (HP9826) disk drive (label side up) and turn on the HP9826. A BASIC autostart routine (AUTOST) is automatically loaded and initiated. (NOTE: If the HP9826 is on prior to placing the data analysis disk into the disk drive, turn the HP9826 power off, then back on. This will allow AUTOST to be loaded automatically.)

3. To start the analysis routine (ANALYSIS) press the special function key "k2" on the HP9826. ANALYSIS will then be loaded into memory from the HP9134A hard disk and executed. Once ANALYSIS is running, the user must press key "k2" to continue or key "k9" to exit back to AUTOST. This extra check allows for accidental execution of ANALYSIS.

4. The number of data points (per channel) to be analyzed should then be entered. As in the Pascal operating system, data input is typed via the computer keyboard and information is accepted by pressing the "ENTER" key.

5. The subject's name or identifier is then entered. This information is printed at the top of the hard copy output produced by the analysis routine. It (the information) is used strictly for identifying the hard copy output.

6. Calibration factors generated from an earlier system calibration procedure can then be loaded from the 8" flexible disk memory. Answering "Y" to the calibration factor question prompts the user to enter the calibration file name. Upon receipt of a valid calibration file name, ANALYSIS reads in the calibration factors. (NOTE: The calibration file must have been crunched by CAPCRUNCH prior to its use in the analysis routine. See Appendix V for more details.)

7. Should an improper file name be entered, an error message will be displayed and the routine will be halted. (This is the case for the BASIC operating system in general.) Should an error condition exist, pressing "PAUSE" followed by the "RUN" key will restart the routine.

8. Binary data file names are then entered for the

CO_2, O_2, flow and temperature data. Once these names are entered, the analysis routine loads the binary data from the 8" flexible disk memory. (NOTE: These data files must have been crunched by DAPCRUNCH prior to their use in the analysis routine. See Appendix VI for more details.)

9. The user then selects the sampling frequency at which the data were collected. A default value of 50 Hz is available as the majority of the previous work was conducted at the 50 Hz sampling frequency. Answering "Y" to changing the sampling frequency causes ANALYSIS to prompt the user for the new sampling frequency.

10. Breath-by-breath or fixed time delays are then selected. Selecting "B" allows for breath-by-breath determination of the gas mass spectrometer time delay (see section 4.5 Data Analysis and Display Software for more details). Selecting "F" causes the current mass spectrometer time delay to be displayed on the HP9826 CRT and the option to alter this delay is made. The user can either alter this fixed delay or use the current delay throughout the remainder of the analysis routine.

11. Provided the necessary information has been collected, gas volumes may (if the user desires) be corrected to BTPS/STPD conditions. Answering "Y" to this question causes a table of water vapor pressures (VAP) to be loaded into memory and further prompting of the operator for the barometric pressure (torr), relative

humidity (%), and body temperature (deg C). This information should have been collected at data collection time.

12. The user is then asked if a plot of the data is desired. Answering "Y" to this question causes the number of data points available for plotting to be displayed. The user then enters the point within this array of data at which to start plotting and the point at which to stop plotting. This option allows the user to expand the time axes on the plots for better definition of the respiratory data.

13. Once these starting and ending points are entered, the maximum and minimum values of the points to be plotted are displayed. Again, this information can be used to determine whether or not any of the four data acquisition channels saturated during the data collection process.

14. The plotted data are then routed either to the CRT or the HP9872C plotter. (Entering "CRT" routes the plot to the HP9826 CRT and "PLOTTER" cause the plot to be plotted on the HP9872C.) If the HP9872C plotter is desired, press the "CHART LOAD" key on the plotter. This releases the electrostatic charge on the plotter surface so plotter paper can be applied. Place an 11" x 16.5" piece of plotter paper on the plotting surface so the long edge (16.5") is along the bottom edge of the plotter and the left edge is flush with the left edge of the

plotting surface. Pressing "CHART HOLD" on the plotter applies the charge to the plotter surface and holds the plotter paper in place. Plotting limits P1 (lower left) and P2 (upper right) should be set by moving the plotting arm to the desired limit using the arrow keys (up, down, left, and right arrows), pressing the "ENTER" key, followed by "P1" or "P2" depending upon which limit is being set. The HP9872C is now ready for plotting. It should be mentioned that once the plot is completed on the HP9826 CRT, an image of that plot can be dumped to the HP2673A thermal printer by simply holding down on the "SHIFT" key and pressing the "DUMP GRAPHICS" key.

15. Once the plot is completed, the analysis routine is in a paused mode. This allows the operator to observe the completed plot and possibly prepare for additional plots. When ready, the user presses the "CONTINUE" key and a question to redo the plot is made.

16. Should the user answer "Y" to this question, ANALYSIS returns to the question concerning the point at which to start plotting and the routine repeats as described. Answering "N" to the redo graphics question causes printed output of the data analysis to begin on a breath-by-breath basis. This option requires calibration files to have been both crunched and previously selected for use in step 6 of this appendix.

17. Upon completion of the hard copy output, ANALYSIS loads the autostart routine from the 5.25"

floppy (HP9826 Data Analysis Routines diskette) and AUTOST is initiated. The user can then perform additional data analysis if desired (see step #3 above).

APPENDIX VIII

CAP.CODE

General Description

CAP.CODE is a Pascal routine which calibrates the system transducers, namely the Perkin-Elmer gas mass spectrometer, the Fleisch/Godart pneumotach assembly, and the respiratory temperature thermocouple. Following is a list of summarized features of DAP.CODE.

1. This routine assumes that the ASCII calibration file created by CAP.CODE is to be stored on hard disk volume #13 (":HP9895,702,2" in the BASIC operating system).

2. 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.

3. Two external 68000 assembly language routines are utilized by CAP.CODE to handle the high speed requirements needed to monitor the STS (STatuS) signal from the DAM. (See Appendix X for more details.)

4. CAP.CODE'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.

5. Procedure DATA_COLLECT is used by CAP.CODE to sample the necessary channels for calibration purposes. It (DATA_COLLECT) is identical to the DATA_COLLECT procedure used by DAP.CODE.

6. Procedure GASCAL is designed to calibrate the Perkin-Elmer gas mass spectrometer for both the CO_2 (channel A) and O_2 (channel B) DAM channels.

7. 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.

8. Procedure TEMPCAL is designed to calibrate the thermocouple for measuring respiratory temperature. TEMPCAL determines a 2nd order equation for converting binary DAM figures into actual temperature values.

9. CAP.CODE converts all the calibration factors to ASCII units and stores these units in a single ASCII data file (see File Structure section for more details.)

Calculations

Following are the important calculations that are made by CAP.CODE.

1. DAM status word

The DAM status word (16 return bits from the DAM to the HP9826 computer) is organized as follows.

DAM Status Bit	Description
0	8253 clk1 output for system timing
1	STS signal from DAM, goes high then low when conversion is complete
2-13	12 bits digital value from the AD574 A/D
14-15	Not used

As is obvious from the preceding bit organization, the equations

```
R4:=IOSTATUS(12,3);      {Read 16-bit status word}
R4:=R4 DIV 4;            {Shift result right 2 bits}
R4:=BINAND(Mask,R4);     {Mask off all but 12 bits}
```

alter the 16-bit value stored in R4 (a variable). R4 is shifted two bits to the right (DIV 4) and bits 14 and 15 are masked off (BINAND(Mask,R4)) to yield the 12-bit value from the A/D.

2. DAM control word

The DAM control word (16 bits to the DAM from the HP9826 computer) is organized as follows.

DAM Control Bit	Description
0	D0 and BCD for 8253 timer control
1	D1 and M0 for 8253 timer control
2	D2 and M1 for 8253 timer control
3	D3 and M2 for 8253 timer control
4	D4 and RL0 for 8253 timer control
5	D5 and RL1 for 8253 timer control
6	D6 and SC0 for 8253 timer control
7	D7 and SC1 for 8253 timer control
8	8253 timer select, low = selected
9	R/C for AD574 A/D, low = start conversion
10	MA0 for AD7503 multiplexer
11	MA1 for AD7503 multiplexer, A1 for 8253 timer
12	MA2 for AD7503 multiplexer, A0 for 8253 timer
13	S/H control for S/H amplifiers, high = hold
14-15	not used

For a detailed explanation of the 8253 timer controls, see the 1980 Intel Component Data Catalog [19]. For the multiplexer [23] controls MA0, MA1, and MA2 the following table is helpful.

MA2	MA1	MA0	Channel/Switch
0	0	0	Channel F, Switch 1
0	0	1	Channel A, Switch 2
0	1	0	Channel E, Switch 3
0	1	1	Channel B, Switch 4
1	0	0	Channel H, Switch 5
1	0	1	Channel C, Switch 6
1	1	0	Channel G, Switch 7
1	1	1	Channel D, Switch 8

As can be seen from the previous bit specifications, to select channel A, with the S/H amplifiers in the hold mode and the A/D convert signal high, the following code is necessary.

```
R6:=BINAND(15360,Chna);   {AND 15360 with 15359}
R6:=BINCMP(R6);           {Compliment result}
IOCONTROL(12,3,R6);       {Write bit pattern to DAM}
```

Similar calculations are performed throughout the procedure DATA_COLLECT.

3. Average binary values

Throughout the calibration procedures GASCAL, FLOWCAL, and TEMPCAL compute average binary values for certain constant calibration points (i.e. for the O_2 channel 21% and 12.9% O_2 levels are used). Following are the calculations used throughout the mentioned procedures for determining these averages.

```
Tot_zero:=LINE3^[1];            {SUM UP Sam POINTS}
FOR I:=2 TO Sam DO
  BEGIN
     Tot_zero:=Tot_zero+LINE3^[I];
  END;
Bin_zero_flow:=Tot_zero DIV Sam;{COMPUTE AVERAGE}
```

4. Inspiratory flow calibration factor

   The inspiratory flow calibration factor (Insp_flow_cal) is determined by integrating the inspiratory side of the flow calibration signal generated by the Harvard pump (of known stroke volume), dividing by the number of breaths included in the integration, and dividing the result into the stroke volume of the Harvard pump. This results in a calibration factor having units of liters per second per binary value. (NOTE: Insp_flow_cal is a positive quantity even though inspiratory flow is considered negative flow.)

5. Expiratory flow calibration factor

   The expiratory flow calibration factor (Expr_flow_cal) is determined by integrating the expiratory side of the flow calibration signal and performing those divisions mentioned in selection 4 above. Expr_flow_cal also has units of liters per second per binary value.

6. Determining the 2nd order temperature coefficients

A set of 2nd order temperature coefficients are determined for converting binary temperature data to actual temperatures in degrees C. Using the method of Least Squares as described by Agnew and Knapp [24] the 2nd order (Ta), 1st order (Tb), and constant coefficients (Tc) are computed from the three calibration temperatures used.

File Structure

One serial ASCII calibration file is created by CAP.CODE. The calibration file name supplied by the user will have the ASCII extension (.ASC) placed on it by the Pascal operating system. Because of the name conversion process necessary between the Pascal and BASIC systems, Pascal file names in excess of 9 characters are not recommended. (NOTE: The ASCII calibration files will appear on the ":HP9895,702,2" hard disk and are purged immediately following execution of the crunch routine CAPCRUNCH.)

Record #	Contents
1	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)
	Insp_flow_cal (25 ASCII bytes)
	Expr_flow_cal (25 ASCII bytes)
	Time_delay (25 ASCII bytes)
	S (4 ASCII bytes)
	O1 (25 ASCII bytes)
	Ta (25 ASCII bytes)
	Tb (25 ASCII bytes)
	Tc (25 ASCII bytes)
	Date (25 ASCII bytes)

Variable List

A — INTEGER variable used as a pointer into the flow signal array Line3 during flow signal integration.

Aire — REAL value containing the amount of air expired for the current breath.

Airi — REAL value containing the amount of air inspired for the current breath.

Avco2h	INTEGER variable containing the average binary value read for 7% CO_2.
Avco2l	INTEGER variable containing the average binary value read for 0% CO_2.
Avo2h	INTEGER variable containing the average binary value read for 21% O_2.
Avo2l	INTEGER variable containing the average binary value read for 12.9% O_2.
Avole	REAL value equal to the average volume of air expired by the Harvard pump.
Avoli	REAL value equal to the average volume of air inspired by the Harvard pump.
B	INTEGER variable equal in value to the variable Bin_zero_flow. "B" was selected because of its shorter name length.
Bin_temp	REAL matrix containing temperature data for determining the 2nd order curve fit coefficients.
Bin_temp_inv	REAL matrix equal to the inverse of the matrix Bin_temp.

Bin_zero_flow INTEGER value equal to the average binary value read from the flow channel for zero flow.

Cal STRING variable containing the user defined name for the calibration file.

Ch REAL variable containing the actual fractional concentration read from the mass spectrometer for 7% CO_2.

Cl REAL variable containing the actual fractional concentration read from the mass spectrometer for 0% CO_2.

Co2_cal REAL value used to convert the binary data collected from the CO_2 channel into fractional concentration values.

Co2_dc_offset INTEGER value equal to the average binary value read from the CO_2 channel for 0% CO_2.

Date STRING variable containing the date of calibration.

Del INTEGER variable used in the delay loop that allows the S/H amplifiers time to track the input signals before the hold command is given.

Expr_btps	REAL variable used to correct expiratory flow values to BTPS conditions. Expr_btps is equal to unity in CAP.CODE.
Expr_flow_cal	REAL variable used to convert expiratory flow binary data points from channel C to flow units of liters per second.
F	TEXT variable containing the Pascal name associated with the ASCII files created by DAP.CODE.
Fl	INTEGER variable used by the procedure CLKSET to set the LSB of clock 1 in the 8253 timer for proper sampling rate.
Flow_cal	REAL value used to multiply binary data collected from the flow channel to obtain units of liters per second. For CAP.CODE, Flow_cal is always equal to unity.
Fm	INTEGER variable used by the procedure CLKSET to set the MSB of clock 1 in the 8253 timer for the proper sampling rate.
Fname	STRING variable containing the calibration file name as it will appear in the Pascal operating system.

I	INTEGER value used as a loop counter and array pointer.
Insp_btps	REAL variable used to correct inspiratory flow values to BTPS conditions. Insp_btps is equal to unity in CAP.CODE.
Insp_flow_cal	REAL variable used to convert inspiratory flow binary data points from channel C to flow units of liters per second.
Line1	24000 point data string containing the BCD values acquired from the CO_2 channel of the DAM. Access of this external data string is made through the pointer "I".
Line2	24000 point data string containing the BCD values acquired from the O_2 channel of the Dam. Access of this external data string is made through the pointer "I".
Line3	24000 point data string containing the BCD values acquired from the flow channel of the DAM. Access of this external data string is made through the pointer "I".

Line4	24000 point data string containing the BCD values acquired from the temperature channel of the DAM. Access of this external data string is made through pointer "I".
Norm_a	REAL variable equal to the determinant of the temperature matrix Bin_temp[3,3].
No_breaths	INTEGER value representing the number of breaths generated by the Harvard pump during the flow calibration procedure.
NSTRING	Four byte STRING variable containing the ASCII representation of the previously converted BCD value.
O2_cal	REAL variable used to convert the binary data collected from O_2 channel to fractional concentration values.
O2_dc_offset	INTEGER variable equal to the average binary value read from the O_2 channel for 12.9% O_2.
O1	REAL variable containing the O_2 concentration read from the mass spectrometer for 12.9% O_2.

Q	STRING variable containing the answer to a question asked by CAP.CODE. Typically this answer is either a "Y" or "N".
R4	INTEGER variable read from the DAM's status register. (See Calculations section above for more details.)
R6	INTEGER variable written to the DAM's control register. (See Calculations section above for more details.)
RSTRING	STRING variable containing the ASCII representation of those REAL calibration factors stored by CAP.CODE.
S	INTEGER value representing the DAM sampling frequency in Hz.
Sam	INTEGER variable representing the number of samples per channel.
T	REAL variable equal to the reciprocal of the sampling frequency (S).
Ta	REAL variable representing the 2nd order coefficient for converting binary temperature data to units of degrees C.

Tb	REAL variable representing the 1st order coefficient for converting binary temperature data to units of degrees C.
Tc	REAL variable representing the constant coefficient for converting binary temperature data to units of degrees C.
TEMP	INTEGER value used by the STRWRITE function in converting the acquired BCD data to ASCII.
Tlow	REAL variable containing the actual low temperature water bath in degrees C.
Tlow_bin	REAL variable equal to the average binary value read for the low temperature water bath.
Tmid	REAL variable containing the actual middle temperature water bath in degrees C.
Tmid_bin	REAL variable equal to the average binary value read for the middle temperature water bath.
Thigh	REAL variable containing the actual high temperature water bath in degrees C.

Thigh_bin	REAL variable equal to the average binary value read for the high temperature water bath.
Tot_temp	INTEGER variable containing the sum of the binary temperature values during temperature calibration.
Tot_vol_expr	REAL value containing the sum of the expired volume during flow calibration.
Tot_vol_insp	REAL value containing the sum of the inspired volume during flow calibration.
Tot_zero	INTEGER variable containing the sum of all the binary data points collected from channel C (the flow channel).
X	INTEGER used by the procedure CLKSET to set clock 1 in the 8253 timer for proper sampling rate.
Z	INTEGER variable used as a pointer into the flow signal array Line3 during flow signal integration.

```
$SYSPROG ON$
$LINES 64$
$REF 40$     {ALLOCATE ROOM FOR REFERENCE TABLE}
PROGRAM CAP(INPUT,OUTPUT);
{*******************************************************************************

        SYSTEM CALIBRATION ROUTINE

        PASCAL REV 2.1 SOURCE FILENAME:  CAP.TEXT

        DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

        KANSAS STATE UNIVERSITY


        REVISION        DATE                    PROGRAMMER
        --------        ----                    ----------
          1.0           JUNE 28, 1984           LOREN E. RIBLETT, JR.

 *******************************************************************************

        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 STS BIT
                          ON THE DAM GOES HIGH, THEN LOW
                BIT_HI -  68000 ASSEMBLY MODULE THAT WAITS UNTIL THE STS 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 OF
                               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

********************************************************************************

        NOTE 1:  THIS ROUTINE ASSUMES THAT THE ASCII CALIBRATION FILE CREATED
                 BY CAP IS TO BE STORED ON HARD DISK VOLUME #13 (":HP9895,702,
                 2" IN THE BASIC OPERATING SYSTEM).

        NOTE 2:  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 3: TWO EXTERNAL 68000 ASSEMBLY LANGUAGE ROUTINES (BIT_TST AND BIT_HI) ARE UTILIZED BY CAP TO HANDLE THE HIGH SPEED REQUIREMENTS NEEDED TO MONITOR THE STS(STATUS) SIGNAL FROM THE DAM.

NOTE 4: CAP´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 5: PROCEDURE DATA_COLLECT IS USED BY CAP TO SAMPLE THE NECESSARY CHANNELS FOR CALIBRATION PURPOSES. IT (DATA_COLLECT) IS IDENTICAL TO THE DATA_COLLECT PROCEDURE USED BY DAP.CODE.

NOTE 6: PROCEDURE CASCAL IS DESIGNED TO CALIBRATE THE PERKIN-ELMER GAS MASS SPECTROMETER FOR BOTH THE CO2 (CHANNEL A) AND O2 (CHANNEL B) DAM CHANNELS.

NOTE 7: 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.

NOTE 8: 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 9: CAP CONVERTS ALL THE CALIBRATION FACTORS TO ASCII UNITS AND STORES THESE UNITS IN A SINGLE ASCII DATA FILE (SEE EXTERNAL PROGRAM DOCUMENTATION FOR MORE DETAILS).

```
*******************************************************************************}
{
*** LOAD NECESSARY LIBRARY MODULES
}
IMPORT IODECLARATIONS,GENERAL_0,
       IOCOMASM;
{
*** SET PROGRAM CONSTANTS
}
CONST Time_delay=400.0;     {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:INTEGER;
    TEMP,Co2_dc_offset:INTEGER;
    02_dc_offset:INTEGER;
    Tot_zero,Bin_zero_flow:INTEGER;
    GPINT[7077890]:INTEGER;
    01,Co2_cal,02_cal:REAL;
    Insp_flow_cal:REAL;
    Expr_flow_cal:REAL;
    Ta,Tb,Tc:REAL;
    NSTRING: STRING [4];
    Q: STRING[1];
    Cal: STRING[6];
    Date: STRING[25];
    Fname: STRING[14];
    RSTRING: STRING[25];
    F: TEXT;
    Line1: PT1;
    Line2: PT2;
    Line3: PT3;
    Line4: PT4;
{
*** DECLARE EXTERNAL 68000 ASSEMBLY MODULES
}
PROCEDURE BIT_TST;EXTERNAL;     {WAITS UNTIL STS BIT GOES HIGH, THEN LOW}
PROCEDURE BIT_HI;EXTERNAL;     {WAITS UNTIL STS BIT IS LOW}
{
*** DECLARE PROCEDURE TO SET DAM CLOCK
}
PROCEDURE CLKSET(VAR S:INTEGER);     {PASS SAMPLING FREQUENCY (S)}
   VAR X,Fm,Fl:INTEGER;
   BEGIN
      {
      *** HAVE USER ENTER THE SAMPLING FREQUENCY
      }
      WRITELN('ENTER SAMPLING FREQUENCY:  ');
      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;
                     Fl:=X-256*Fm;
                   END
              ELSE
                   BEGIN
                     Fm:=0;
                     Fl:=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 PAUSING PROGRAM OPERATION
}
PROCEDURE HOLD_UP;
  BEGIN
      WRITELN('PRESS ENTER TO CONTINUE.');     {DISPLAY PROMPT ON CRT}
      READLN;      {WAIT UNTIL 'ENTER' IS PRESSED}
  END; {HOLD_UP END}
{
*** DECLARE PROCEDURE FOR COLLECTING FOUR CHANNELS OF DAM DATA
}
PROCEDURE DATA_COLLECT(VAR Sam:INTEGER;
                       VAR LINE1:PT1;VAR LINE2:PT2;
                       VAR LINE3:PT3;VAR LINE4:PT4);
  VAR I,R6,Del,R4:INTEGER;
  CONST Chna=15359;Chnb=13311;     {BIT PATTERNS NECESSARY TO SET THE CHANNEL MU
LTIPLEXER}
  CONST Chnc=11263;Chnd=9215;
```

```
CONST Mask=4095;      {MASKS OFF ALL BUT 12 DATA BITS IN STATUS WORD}
{
*** MAIN LOOP FOR FOUR CHANNEL DATA ACQUISITION
}
BEGIN {DATA_COLLECT}
    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
            }
            Del:=15;
            WHILE Del>0 DO
                BEGIN
                    Del:=Del-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 STS 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 STS 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 STS 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 STS 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 STS 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,
                 Sam:INTEGER; VAR LINE1:PT1;
                 VAR LINE2:PT2; VAR LINE3:PT3;
                 VAR LINE4:PT4);
  VAR I,Avco2l,Avo2h,Avco2h,Avo2l:INTEGER;
      Cl,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.');
     WRITELN;
```

```
{
*** OBTAIN ACTUAL O2 AND CO2 CONCENTRATIONS FROM GMS FRONT PANEL
}
WRITELN('ENTER ACTUAL CO2 CONCENTRATION.');
READLN(Cl);
WRITELN('ENTER ACTUAL O2 CONCENTRATION.');
READLN(Oh);
{
*** FOLLOWING A CARRIAGE RETURN, TAKE 1000 DATA POINTS ON CO2 AND O2
*** CHANNELS
}
HOLD_UP;
DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE AVERAGE BINARY VALUES FOR O2 AND CO2 CHANNELS
}
Avco2l:=LINE1^[1];
Avo2h:=LINE2^[1];
FOR I:=2 TO Sam DO
   BEGIN
     Avco2l:=Avco2l+LINE1^[I];
     Avo2h:=Avo2h+LINE2^[I];
   END;
Avco2l:=Avco2l DIV 1000;
Co2_dc_offset:=Avco2l;
Avo2h:=Avo2h DIV 1000;
{
*** DISPLAY AVERAGES ON HP9826 CRT
}
WRITELN('CO2 DC OFFSET =',Co2_dc_offset);
WRITELN('Value read for 0% Co2 was...',Avco2l);
WRITELN('Value read for 21% O2 was...',Avo2h);
{
*** INSTRUCT THE USER TO CONNECT GMS PROBE TO 7% CO2 AND 13% O2
}
WRITELN('CONNECT THE GMS PROBE TO 7% CO2 AND 13% O2.');
WRITELN;
{
*** OBTAIN ACTUAL O2 AND CO2 CONCENTRATIONS FROM GMS FRONT PANEL
}
WRITELN('ENTER THE ACTUAL VALUE FOR THE CO2 CONCENTRATION.');
READLN(Ch);
WRITELN('ENTER THE ACTUAL VALUE FOR THE O2 CONCENTRATION.');
READLN(Ol);
{
*** TAKE 1000 DATA POINTS ON CO2 AND O2 CHANNELS
}
HOLD_UP;
DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE AVERAGE BINARY VALUES FOR O2 AND CO2 CHANNELS
}
Avco2h:=LINE1^[1];
Avo2l:=LINE2^[1];
FOR I:=2 TO Sam DO
```

```
        BEGIN
          Avco2h:=Avco2h+LINE1^[I];
          Avo21:=Avo21+LINE2^[I];
        END;
      Avco2h:=Avco2h DIV 1000;
      Avo21:=Avo21 DIV 1000;
      02_dc_offset:=Avo21;
      {
      *** DISPLAY AVERAGES ON HP9826 CRT
      }
      WRITELN('02 DC OFFSET =',02_dc_offset);
      WRITELN('Average value read for 7% CO2 was...',Avco2h);
      WRITELN('Average value read for 13% 02 was...',Avo21);
      {
      *** COMPUTE CO2 AND 02 CALIBRATION FACTORS FROM THESE AVERAGES
      }
      Co2_cal:=ROUND((Ch-Cl)/(Avco2h-Avco21)*10000000)/10000000;
      02_cal:=ROUND((Oh-Ol)/(Avo2h-Avo21)*10000000)/10000000;
      {
      *** DISPLAY CO2 AND 02 CALIBRATION FACTORS ON 9826 CRT
      }
      WRITELN('CO2 CALIBRATION FACTOR =',Co2_cal);
      WRITELN('02 CALIBRATION FACTOR =',02_cal);
  END; {GASCAL}
{
*** DECLARE PROCEDURE FOR CALIBRATING THE FLEISCH/GODART FLOW APPARATUS
}
PROCEDURE FLOWCAL(VAR Insp_flow_cal,
                  Expr_flow_cal:REAL;
                  VAR Bin_zero_flow,Sam:INTEGER;
                  VAR LINE1:PT1; VAR LINE2:PT2;
                  VAR LINE3:PT3; VAR LINE4:PT4);
  LABEL 1,2,3,4,5,6,7,8;
  VAR I,Tot_zero,No_breaths,A,Z,B:INTEGER;
      T,Tot_vol_insp,Tot_vol_exp,Flow_cal:REAL;
      Insp_btps,Expr_btps,Airi,Aire:REAL;
      Avoli,Avole:REAL;
  {
  *** BEGIN FLOW CALIBRATION
  }
  BEGIN {FLOWCAL}
       {
       *** INSTRUCT USER TO APPLY ZERO FLOW TO THE PNEUMOTACHOMETER
       }
       WRITELN('CONNECT ZERO FLOW TO PNEUMOTACHOMETER.');
       WRITELN;
       {
       *** FOLLOWING A CARRIAGE RETURN, GO COLLECT 1000 FLOW DATA POINTS
       }
       HOLD_UP;
       DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
       {
       *** AVERAGE THE 1000 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 1000;
{
*** DISPLAY BINARY ZERO FLOW VALUE ON 9826 CRT
}
WRITELN('Average binary value for zero flow =',Bin_zero_flow);
{
*** INSTRUCT USER TO CONNECT PNEUMOTACHOGRAPH TO HARVARD PUMP
}
WRITELN('CONNECT PUMP FLOW TO THE PNEUMOTACHOGRAPH.');
WRITELN;
{
*** FOLLOWING A CARRIAGE RETURN, GO COLLECT 4000 DATA POINTS OF PUMP FLOW
}
HOLD_UP;
Sam:=4000;
WRITELN('NOW COLLECTING DATA... please wait patiently.');
DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
{
*** INSTRUCT USER DATA COLLECTION IS COMPLETE AND BEGIN INTEGRATION
*** OF FLOW SIGNAL TO DETERMINE INSPIRATORY AND EXPIRATORY FLOW CALIBRATI
ON
}
WRITELN('DATA COLLECTION COMPLETE... turn off the pump.');
{
*** INITIALIZE NECESSARY VARIABLES FOR FLOW SIGNAL INTEGRATION
}
No_breaths:=0;
T:=1/S;
A:=1;
Z:=1;
Tot_vol_insp:=0;
Tot_vol_exp:=0;
Flow_cal:=1;
Insp_flow_cal:=1;
Expr_flow_cal:=1;
B:=Bin_zero_flow;
{
*** LOOK FOR FIRST INSPIRATION IN FLOW SIGNAL
}
WHILE ((LINE3^[A]-B)<0) OR
   ((LINE3^[A+1]-B)>=0) OR
   ((LINE3^[A+2]-B)>=0) OR
   ((LINE3^[A+3]-B)>=0) OR
   ((LINE3^[A+4]-B)>=0) DO
   BEGIN
     A:=A+1;
   END;
{
*** ADJUST FLOW INDEX A AS NEEDED TO BEGINNING OF INSPIRATION
}
IF ((LINE3^[A]-B)<>0) THEN
```

```
            BEGIN
              A:=A+1;
            END;
         {
         *** ADJUST ADDITIONAL ROUTINE VARIABLES
         }
         Z:=A;
         Insp_btps:=1;
         Expr_btps:=1;
         {
         *** PRINT HEADER ON CRT FOR BREATH-BY-BREATH INTEGRATION DISPLAY
         }
         WRITELN('BREATH       AIR          AIR');
         WRITELN('NUMBER    INSPIRED     EXPIRED');
         WRITELN('           (LITERS)     (LITERS)');
         WRITELN('------------------------------');
{
*** MAKE SURE FLOW INDEX IS AT BEGINNING OF INSPIRATION
}
8:       WHILE ((LINE3^[A]-B)>0) OR
            ((LINE3^[A+1]-B)>=0) OR
            ((LINE3^[A+2]-B)>=0) OR
            ((LINE3^[A+3]-B)>=0) OR
            ((LINE3^[A+4]-B)>=0) OR
            ((LINE3^[A+5]-B)>=0) DO
            BEGIN
              A:=A+1;
              Z:=Z+1;
              IF A> Sam-10 THEN GOTO 1; {Goon}
            END;
         {
         *** COMPUTE 1/2 OF FIRST TRAPEZOIDAL AREA OF INSPIRATION
         }
         Flow_cal:=Insp_flow_cal;
         Airi:=0.5*(LINE3^[A]-B)*Flow_cal*Insp_btps;
{
*** SUM UP ENTIRE INSPIRATION TRAPEZOIDS
}
4:       A:=A+1;
         Z:=Z+1;
         IF Z>Sam THEN GOTO 1; {Goon}
         IF LINE3^[A]-B=0 THEN GOTO 2; {B}
         IF LINE3^[A]-B>0 THEN GOTO 3; {Decri}
         Airi:=Airi+(LINE3^[A]-B)*Flow_cal*Insp_btps;
         GOTO 4; {A_label}
{
*** SUBTRACT OFF 1/2 OF LAST TRAPEZOIDAL AREA OF INSPIRATION
}
3:       A:=A-1;
         Z:=Z-1;
         Airi:=Airi-0.5*(LINE3^[A]-B)*Flow_cal*Insp_btps;
         A:=A+1;
         Z:=Z+1;
{
*** LOOP UNTIL BEGINNING OF EXPIRATION IS FOUND
```

```
}
2:    IF A>Sam-10 THEN GOTO 1; {Goon}
      WHILE (LINE3^[A]-B<0) OR
            (LINE3^[A+1]-B<=0) OR
            (LINE3^[A+2]-B<=0) OR
            (LINE3^[A+3]-B<=0) OR
            (LINE3^[A+4]-B<=0) OR
            (LINE3^[A+5]-B<=0) DO
         BEGIN
           A:=A+1;
           Z:=Z+1;
           IF A>Sam-10 THEN GOTO 1; {Goon}
         END;
      {
      *** COMPUTE 1/2 OF FIRST TRAPEZOIDAL AREA OF EXPIRATION
      }
      Flow_cal:=Expr_flow_cal;
      Aire:=0.5*(LINE3^[A]-B)*Flow_cal*Expr_btps;
{
*** SUM UP ENTIRE EXPIRATION TRAPEZOIDS
}
7:    A:=A+1;
      Z:=Z+1;
      IF Z>Sam THEN GOTO 1; {Goon}
      IF LINE3^[A]-B=0 THEN GOTO 5; {C}
      IF LINE3^[A]-B<0 THEN GOTO 6; {Decre}
      Aire:=Aire+(LINE3^[A]-B)*Flow_cal*Expr_btps;
      GOTO 7; {F}
{
*** SUBTRACT OFF 1/2 OF LAST TRAPEZOIDAL AREA OF EXPIRATION
}
6:    A:=A-1;
      Z:=Z-1;
      Aire:=Aire-0.5*(LINE3^[A]-B)*Flow_cal*Expr_btps;
      A:=A+1;
      Z:=Z+1;
{
*** COMPUTE AIR INSPIRED AND AIR EXPIRED FOR THIS BREATH
}
5:    Airi:=Airi*T;
      Aire:=Aire*T;
      {
      *** BUMP THE NUMBER OF BREATHS COUNT
      }
      No_breaths:=No_breaths+1;
      {
      *** UPDATE TOTAL VOLUMES INSPIRED AND EXPIRED
      }
      Tot_vol_insp:=Tot_vol_insp+Airi;
      Tot_vol_exp:=Tot_vol_exp+Aire;
      {
      *** DISPLAY AIR INSPIRED AND AIR EXPIRED ON CRT FOR THIS BREATH
      }
      WRITELN(No_breaths,'  ',Airi,'  ',Aire);
      {
```

```
        *** LOOP UNTIL FLOW DATA ARRAY IS EXHAUSTED
        }
        GOTO 8; {New_inspire}
   {
   *** COMPUTE AVERAGE VOLUMES INSPIRED AND EXPIRED (PER BREATH BASIS)
   }
   1:   Avoli:=Tot_vol_insp/No_breaths;
        Avole:=Tot_vol_exp/No_breaths;
        {
        *** COMPUTE INSPIRATORY AND EXPIRATORY FLOW CALIBRATION FACTORS
        }
        Insp_flow_cal:=ROUND(0.647/(-1*Avoli)*10000000)/10000000;
        Expr_flow_cal:=ROUND(0.647/Avole*10000000)/10000000;
        {
        *** DISPLAY FLOW CALIBRATIONS ON HP9826 CRT
        }
        WRITELN('BINARY ZERO FLOW =',Bin_zero_flow);
        WRITELN('INSPIRATORY FLOW CALIBRATION FACTOR =',Insp_flow_cal);
        WRITELN('EXPIRATORY FLOW CALIBRATION FACTOR =',Expr_flow_cal);
  END; {FLOWCAL}
{
*** DECLARE PROCEDURE FOR CALIBRATION OF TEMPERATURE TRANSDUCER
}
PROCEDURE TEMPCAL(VAR Ta,Tb,Tc:REAL;
                  VAR Sam:INTEGER;
                  VAR LINE1:PT1; VAR LINE2:PT2;
                  VAR LINE3:PT3; VAR LINE4:PT4);
  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 ACTUAL WATER BATH TEMPERATURE (deg C).');
        READLN(Tlow);
        {
        *** GO TAKE 1000 DATA POINTS ON THE TEMPERATURE CHANNEL
        }
        Sam:=1000;
        DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
        {
        *** COMPUTE THE AVERAGE VALUE FOR THE 1000 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,' deg C');
WRITELN('IS:  ',Tlow_bin);
{
*** 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 ACTUAL WATER BATH TEMPERATURE (deg C).');
READLN(Tmid);
{
*** GO TAKE 1000 DATA POINTS ON THE TEMPERATURE CHANNEL
}
DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE THE AVERAGE VALUE FOR THE 1000 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,' deg C');
WRITELN('IS:  ',Tmid_bin);
{
*** 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 ACTUAL WATER BATH TEMPERATURE (deg C).');
READLN(Thigh);
{
*** GO TAKE 1000 DATA POINTS ON THE TEMPERATURE CHANNEL
}
DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
{
*** COMPUTE THE AVERAGE VALUE FOR THE 1000 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,' deg C');
WRITELN('IS:  ',Thigh_bin);
{
*** 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 (CAP)
}
BEGIN {CAP 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 1000 POINTS PER CHANNEL
  }
  Sam:=1000;
  {
  *** CALIBRATE FRACTIONAL CONCENTRATIONS SIGNALS IF DESIRED
  }
  WRITELN('CALIBRATE FRACTIONAL CONCENTRATION SIGNAL? (Y/N)');
  READLN(Q);
  IF (Q='Y') OR (Q='y') THEN
     BEGIN
       GASCAL(O1,Co2_cal,O2_cal,Co2_dc_offset,
              O2_dc_offset,Sam,LINE1,LINE2,
```

```
            LINE3,LINE4);
   END;
{
*** CALIBRATE THE FLOW SIGNAL IF DESIRED
}
WRITELN('CALIBRATE THE FLOW SIGNAL? (Y/N)');
READLN(Q);
IF (Q='Y') OR (Q='y') THEN
   BEGIN
     FLOWCAL(Insp_flow_cal,Expr_flow_cal,
             Bin_zero_flow,Sam,LINE1,LINE2,
             LINE3,LINE4);
   END;
{
*** CALIBRATE THE TEMPERATURE SIGNAL IF DESIRED
}
WRITELN('CALIBRATE FLOW TEMPERATURE SIGNAL? (Y/N)');
READLN(Q);
IF (Q='Y') OR (Q='y') THEN
   BEGIN
     TEMPCAL(Ta,Tb,Tc,Sam,LINE1,LINE2,
             LINE3,LINE4);
   END;
{
*** STORE THE CALIBRATION DATA IN AN ASCII FILE IF DESIRED
}
WRITELN('STORE CALIBRATION FACTORS ON DISK? (Y/N)');
READLN(Q);
IF (Q='Y') OR (Q='y') THEN
   BEGIN
     {
     *** GET CALIBRATION FILE NAME
     }
     WRITELN('ENTER THE FILE NAME FOR CALIBRATION FACTORS.');
     READLN(Cal);
     {
     *** GET TODAYS DATE
     }
     WRITELN('ENTER TODAYS DATE, FORMAT:  Month/Day/Year');
     READLN(Date);
     {
     *** CONSTRUCT ASCII FILE NAME AS IT SHOULD APPEAR IN THE PASCAL
     *** FILE DIRECTORY (.ASC EXTENSION).
     }
     Fname:='#13:.ASC';
     STRINSERT(Cal,Fname,5);
     {
     *** CREATE OR REWRITE ASCII FILE ON VOLUME #13 (":HP9895,702,2")
     }
     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,Insp_flow_cal);
WRITELN(F,RSTRING);
{
*** WRITE Expr_flow_cal TO FILE
}
STRWRITE(RSTRING,1,TEMP,Expr_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);
        {
        *** CLOSE AND COMPACT THE CALIBRATION FILE
        }
        CLOSE(F,'CRUNCH');
      END;
END. {CAP END}
```

APPENDIX IX

DAP.CODE

General Description

DAP.CODE is a Pascal routine which performs all the necessary actions to acquire, convert (to ASCII), and store four channels (CO_2, O_2, flow, and temperature) of 12-bit binary data from the DAM. Following is a list of summarized features of DAP.CODE.

1. This routine assumes that the ASCII files created by DAP.CODE are to be stored on hard disk volume #13 (":HP9895,702,2" in the BASIC operating system).

2. 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.

3. A maximum of 24000 data points per channel is allowed with the current version of DAP.CODE. This value may have to be reduced if substantial additions to DAP.CODE's program length is required.

4. Two external 68000 assembly language routines are utilized by DAP.CODE to handle the high speed requirements needed to monitor the STS (STatuS) signal from the DAM. (See Appendix X for more details.)

5. DAP.CODE'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.

6. Once the requisite number of samples per channel are collected, procedure MAXMIN determines the maximum and minimum for each of the four data acquisition channels.

7. Procedure DATA_STORAGE converts the 12-bit binary values to 4-byte ASCII units and stores these units in four ASCII data files (see File Structure section for more details.)

Calculations

Following are the important calculations that are made by DAP.CODE.

1. DAM status word

The DAM status word (16 return bits from the DAM to the HP9826 computer) is organized as follows.

DAM Status Bit	Description
0	8253 clk1 output for system timing
1	STS signal from DAM, goes high then low when conversion is complete
2-13	12 bits digital value from the AD574 A/D
14-15	Not used

As is obvious from the preceding bit organization, the equations

```
R4:=IOSTATUS(12,3);      {Read 16 bit status word}
R4:=R4 DIV 4;            {Shift result right 2 bits}
R4:=BINAND(Mask,R4);     {Mask off all but 12 bits}
```

alter the 16-bit value stored in R4 (a variable). R4 is shifted two bits to the right (DIV 4) and bits 14 and 15 are masked off (BINAND(Mask,R4)) to yield the 12-bit value from the A/D.

2. DAM control word

The DAM control word (16 bits to the DAM from the HP9826 computer) is organized as follows.

DAM Control Bit	Description
0	D0 and BCD for 8253 timer control
1	D1 and M0 for 8253 timer control
2	D2 and M1 for 8253 timer control
3	D3 and M2 for 8253 timer control
4	D4 and RL0 for 8253 timer control
5	D5 and RL1 for 8253 timer control
6	D6 and SC0 for 8253 timer control
7	D7 and SC1 for 8253 timer control
8	8253 timer select, low = selected
9	R/C for AD574 A/D, low = start conversion
10	MA0 for AD7503 multiplexer
11	MA1 for AD7503 multiplexer, A1 for 8253 timer
12	MA2 for AD7503 multiplexer, A0 for 8253 timer
13	S/H control for S/H amplifiers, high = hold
14-15	not used

For a detailed explanation of the 8253 timer controls, see the 1980 Intel Component Data Catalog [19]. For the multiplexer [23] controls MA0, MA1, and MA2 the following table is helpful.

MA2	MA1	MA0	Channel/Switch
0	0	0	Channel F, Switch 1
0	0	1	Channel A, Switch 2
0	1	0	Channel E, Switch 3
0	1	1	Channel B, Switch 4
1	0	0	Channel H, Switch 5
1	0	1	Channel C, Switch 6
1	1	0	Channel G, Switch 7
1	1	1	Channel D, Switch 8

As can be seen from the previous bit specifications, to elect channel A, with the S/H amplifiers in the hold mode and the A/D convert signal high, the following code is necessary.

```
R6:=BINAND(15360,Chna);   {AND 15360 with 15359}
R6:=BINCMP(R6);           {Compliment result}
IOCONTROL(12,3,R6);       {Write bit pattern to DAM}
```

Similar calculations are performed throughout the procedure DATA_COLLECT.

File Structure

Four serial ASCII data files are created by DAP.CODE and each are organized into single record files. These files are always named MONSTER1.ASC, MONTER2.ASC, MONSTER3.ASC, AND MONSTER4.ASC for the CO_2, O_2, flow, and temperature data files respectively. Following is the organization of these files.

(NOTE: These files will appear on the ":HP9895,702,2" hard disk and are purged immediately following execution of the crunch routine DAPCRUNCH.)

File	Record #	Contents
MONSTER1.ASC	1	CO_2 channel maximum (4 ASCII bytes) CO_2 channel minimum (4 ASCII bytes) n CO_2 channel data points (4 ASCII bytes per point)
MONSTER2.ASC	1	O_2 channel maximum (4 ASCII bytes) O_2 channel minimum (4 ASCII bytes) n O_2 channel data points (4 ASCII bytes per point)
MONSTER3.ASC	1	Flow channel maximum (4 ASCII bytes) Flow channel minimum (4 ASCII bytes) n flow channel data points (4 ASCII bytes per point)
MONSTER4.ASC	1	Temperature channel maximum (4 ASCII bytes) Temperature channel minimum (4 ASCII bytes) n temperature channel data points (4 ASCII bytes per point)

Variable List

Co2max	INTEGER variable representing the maximum acquired BCD value on the CO_2 channel.
Co2min	INTEGER variable representing the minimum acquired BCD value on the CO_2 channel.
Del	INTEGER variable used in the delay loop that allows the S/H amplifiers time to track the input signals before the hold command is given.
F	TEXT variable containing the Pascal name associated with the ASCII files created by DAP.CODE.
Fl	INTEGER variable used by the procedure CLKSET to set the LSB of clock 1 in the 8253 timer for proper sampling rate.
Fm	INTEGER variable used by the procedure CLKSET to set the MSB of clock 1 in the 8253 timer for proper sampling rate.
I	INTEGER variable used as a loop counter and array pointer.

Line1	24000 point data string containing the BCD values acquired from the CO_2 channel of the DAM. Access of this external data string is made through the pointer "I".
Line2	24000 point data string containing the BCD values acquired from the O_2 channel of the DAM. Access of this external data string is made through the pointer "I".
Line3	24000 point data string containing the BCD values acquired from the flow channel of the DAM. Access of this external data string is made through the pointer "I".
Line4	24000 point data string containing the BCD values acquired from the temperature channel of the DAM. Access of this external data string is made through pointer "I".
NSTRING	Four byte STRING variable containing the ASCII representation of the previously converted BCD value.
O2max	INTEGER variable representing the maximum acquired BCD value on the O_2 channel.

O2min	INTEGER variable representing the minimum acquired BCD value on the O_2 channel.
R4	INTEGER variable read from the DAM's status register. (See Calculations section above for more details.)
R6	INTEGER variable written to the DAM's control register. (See Calculations section above for more details.)
S	INTEGER variable representing the DAM sampling frequency in Hz.
Sam	INTEGER variable representing the number of samples per channel.
TEMP	INTEGER variable used by the STRWRITE function in converting the acquired BCD data to ASCII.
Tmax	INTEGER variable representing the maximum acquired BCD value on the temperature channel.
Tmin	INTEGER variable representing the minimum acquired BCD value on the temperature channel.

Vmax	INTEGER variable representing the maximum acquired BCD value on the flow channel.
Vmin	INTEGER variable representing the minimum acquired BCD value on the flow channel.
X	INTEGER variable used by the procedure CLKSET to set clock 1 in the 8253 timer for proper sampling rate.

```
$SYSPROG ON$
$LINES 64$
PROGRAM DAP(INPUT,OUTPUT);
{****************************************************************************

        DATA ACQUISTION AND STORAGE ROUTINE

        PASCAL REV 2.1 SOURCE FILENAME:  DAP.TEXT

        DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

        KANSAS STATE UNIVERSITY

        REVISION        DATE                    PROGRAMMER
        --------        ----                    ----------
          1.0           JUNE 28, 1984           LOREN E. RIBLETT, JR.

*****************************************************************************

        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 STS BIT
                          ON THE DAM GOES HIGH, THEN LOW
                BIT_HI -  68000 ASSEMBLY MODULE THAT WAITS UNTIL THE STS 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

*****************************************************************************

        NOTE 1:  THIS ROUTINE ASSUMES THAT THE ASCII FILES CREATED BY DAP
                 ARE TO BE STORED ON HARD DISK VOLUME #13 (":HP9895,702,2"
                 IN THE BASIC OPERATING SYSTEM).

        NOTE 2:  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 3:  A MAXIMUM OF 24000 DATA POINTS PER CHANNEL IS ALLOWED WITH
                 THE CURRENT VERSION OF DAP.  THIS VALUE MAY HAVE TO BE
                 REDUCED IF SUBSTANTIAL ADDITIONS TO DAP'S PROGRAM LENGTH
                 IS REQUIRED.

        NOTE 4:  TWO EXTERNAL 68000 ASSEMBLY LANGUAGE ROUTINES ARE UTILIZED
                 BY DAP TO HANDLE THE HIGH SPEED REQUIREMENTS NEEDED TO
                 MONITOR THE Status (STS) SIGNAL FROM THE DAM.

        NOTE 5:  AT PRESENT, 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.

*******************************************************************************}
{
*** 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:INTEGER;
    Co2max,O2max,Vmax,Tmax:INTEGER;
    Line1: PT1;
    Line2: PT2;
    Line3: PT3;
    Line4: PT4;
{
*** DECLARE EXTERNAL 68000 ASSEMBLY MODULES
}
PROCEDURE BIT_TST;EXTERNAL;     {WAITS UNTIL STS BIT GOES HIGH, THEN LOW}
PROCEDURE BIT_HI;EXTERNAL;     {WAITS UNTIL STS BIT IS LOW}
{
*** 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
}
WRITELN('ENTER SAMPLING FREQUENCY:  ');
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 TO DETERMINE MAXIMUM AND MINIMUM VALUES FOR
*** CO2, O2, FLOW, AND TEMPERATURE DATA ARRAYS
}
PROCEDURE MAXMIN(VAR Sam,Co2max,Co2min,O2max,
       O2min,Vmax,Vmin,Tmax,Tmin:INTEGER;
       VAR Line1:PT1; VAR Line2:PT2;
       VAR Line3:PT3; VAR Line4:PT4);
  VAR I:INTEGER;
  BEGIN {MAX/MIN ROUTINE}
     {
     *** SET INITIAL MAX/MIN VALUES TO FIRST DATA POINTS IN ARRAYS
     }
     Co2max:=Line1^[1];
     Co2min:=Line1^[1];
     O2max:=Line2^[1];
     O2min:=Line2^[1];
     Vmax:=Line3^[1];
     Vmin:=Line3^[1];
     Tmax:=Line4^[1];
     Tmin:=Line4^[1];
     {
     *** STEP THROUGH REMAINDER OF DATA ARRAYS TO FIND TRUE MAX/MIN VALUES
     }
     FOR I:=2 TO Sam DO
        BEGIN
        {
        *** CHECK FOR NEW CO2 CHANNEL MAXIMUM
        }
        IF Co2max<Line1^[I] THEN
           BEGIN
              Co2max:=Line1^[I];
           END;
        {
        *** CHECK FOR NEW CO2 CHANNEL MINIMUM
        }
        IF Co2min>Line1^[I] THEN
           BEGIN
              Co2min:=Line1^[I];
           END;
        {
        *** CHECK FOR NEW O2 CHANNEL MAXIMUM
        }
        IF O2max<Line2^[I] THEN
           BEGIN
              O2max:=Line2^[I];
           END;
        {
        *** CHECK FOR NEW O2 CHANNEL MINIMUM
        }
        IF O2min>Line2^[I] THEN
           BEGIN
              O2min:=Line2^[I];
           END;
```

```
        {
        *** CHECK FOR NEW FLOW CHANNEL MAXIMUM
        }
        IF Vmax<Line3^[I] THEN
           BEGIN
              Vmax:=Line3^[I];
           END;
        {
        *** CHECK FOR NEW FLOW CHANNEL MINIMUM
        }
        IF Vmin>Line3^[I] THEN
           BEGIN
              Vmin:=Line3^[I];
           END;
        {
        *** CHECK FOR NEW TEMPERATURE CHANNEL MAXIMUM
        }
        IF Tmax<Line4^[I] THEN
           BEGIN
              Tmax:=Line4^[I];
           END;
        {
        *** CHECK FOR NEW TEMPERATURE CHANNEL MINIMUM
        }
        IF Tmin>Line4^[I] THEN
           BEGIN
              Tmin:=Line4^[I];
           END;
        END;
  END; {MAXMIN END}
{
*** DECLARE PROCEDURE TO CREATE FOUR ASCII DATA FILES FOR CO2, O2, FLOW,
*** AND TEMPERATURE DATA
}
PROCEDURE DATA_STORAGE(VAR Sam,Co2max,Co2min,
       O2max,O2min,Vmax,Vmin,Tmax,Tmin:INTEGER;
       VAR Line1:PT1; VAR Line2:PT2;
       VAR Line3:PT3; VAR Line4:PT4);
  VAR TEMP,I:INTEGER;
      NSTRING: STRING [4];
      F: TEXT;
  BEGIN {DATA_STORAGE}
     {
     *** BEGIN CREATING ASCII CO2 FILE, FILENAME = MONSTER1.ASC
     }
     REWRITE(F,'#13:MONSTER1.ASC');     {CREATE OR REWRITE FILE ON UNIT #13}
     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('CHANNEL #1:  ',NSTRING);     {DISPLAY VALUE ON CRT}
        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
   }
   REWRITE(F,'#13:MONSTER2.ASC'};     {CREATE OR REWRITE FILE ON UNIT #13}
   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('CHANNEL #2:  ',NSTRING);     {DISPLAY VALUE ON CRT}
        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
   }
   REWRITE(F,'#13:MONSTER3.ASC'};     {CREATE OR REWRITE FILE ON UNIT #13}
   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 ASCI
I}
        WRITELN('CHANNEL #3:  ',NSTRING);     {DISPLAY VALUE ON CRT}
        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
   }
   REWRITE(F,'#13:MONSTER4.ASC'};     {CREATE OR REWRITE FILE ON UNIT #13}
   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 TEMPERATURE VALUE
TO ASCII}
            WRITELN('CHANNEL #4:  ',NSTRING);     {DISPLAY VALUE ON CRT}
            WRITELN(F,NSTRING);     {WRITE TEMPERATURE VALUE TO FILE}
          END;
      CLOSE(F,'CRUNCH');     {CLOSE AND COMPACT ASCII FLOW FILE}
  END; {DATA_STORAGE}
{
*** DECLARE PROCEDURE FOR PAUSING PROGRAM OPERATION
}
PROCEDURE HOLD_UP;
  BEGIN
      WRITELN('PRESS ENTER TO CONTINUE.');     {DISPLAY PROMPT ON CRT}
      READLN;     {WAIT UNTIL 'ENTER' IS PRESSED}
  END; {HOLD_UP END}
{
*** DECLARE PROCEDURE FOR COLLECTING FOUR CHANNELS OF DAM DATA
}
PROCEDURE DATA_COLLECT(VAR Sam:INTEGER;
                       VAR LINE1:PT1;VAR LINE2:PT2;
                       VAR LINE3:PT3;VAR LINE4:PT4);
  VAR I,R6,Del,R4:INTEGER;
  CONST Chna=15359;Chnb=13311;     {BIT PATTERNS NECESSARY TO SET THE CHANNEL MU
LTIPLEXER}
  CONST Chnc=11263;Chnd=9215;
  CONST Mask=4095;     {MASKS OFF ALL BUT 12 DATA BITS IN STATUS WORD}
  BEGIN {DATA_COLLECT}
      {
      *** MAIN LOOP FOR FOUR CHANNEL DATA ACQUISTION
      }
      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
            }
            Del:=15;
            WHILE Del>0 DO
               BEGIN
                  Del:=Del-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);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** SELECT CHANNEL A ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINAND(7168,Chna);
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** WAIT FOR STS 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 STS 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);
R6:=BINAND(7168,Chnc);
{
*** SELECT CHANNEL C ON MULTIPLEXER AND CONVERT SIGNAL HIGH
}
R6:=BINCMP(R6);
IOCONTROL(12,3,R6);
{
*** WAIT FOR STS 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 STS 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 STS SIGNAL TO GO HIGH, THEN LOW
```

```
              }
              BIT_TST;
           {
           *** LOOP BACK UNTIL ALL POINTS ARE COLLECTED
           }
           END;
    END; {DATA_COLLECT}
  {
  *** BEGIN MAIN DATA ACQUISITION PROGRAM (DAP)
  }
  BEGIN {DAP 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 TO DESIRED FREQUENCY
    }
    CLKSET(S};
    {
    *** DETERMINE NUMBER OF DATA POINTS PER CHANNEL TO ACQUIRE
    }
    Sam:=24001;
    WHILE Sam>24000 DO
       BEGIN
         WRITELN('ENTER NUMBER OF SAMPLES [24000 MAX]:  ');
         READLN(Sam};
       END;
    HOLD_UP;
    {
    *** GO COLLECT THE FOUR CHANNELS WORTH OF DATA
    }
    DATA_COLLECT(Sam,LINE1,LINE2,LINE3,LINE4);
    {
    *** DETERMINE THE MAXIMUM AND MINIMUM VALUES FOR EACH OF THE FOUR CHANNELS
    }
    MAXMIN(Sam,Co2max,Co2min,O2max,O2min,
         Vmax,Vmin,Tmax,Tmin,Line1,Line2,
         Line3,Line4};
    {
    *** DISPLAY THESE MAX/MIN VALUES ON THE CRT
    }
    WRITELN('Co2max = ',Co2max);
    WRITELN('Co2min = ',Co2min};
    WRITELN('O2max  = ',O2max);
    WRITELN('O2min  = ',O2min);
    WRITELN('Vmax   = ',Vmax};
    WRITELN('Vmin   = ',Vmin);
    WRITELN('Tmax   = ',Tmax);
    WRITELN('Tmin   = ',Tmin);
    HOLD_UP;
    {
    *** GO STORE THE FOUR CHANNELS OF DATA AS ASCII FILES
    }
    DATA_STORAGE(Sam,Co2max,Co2min,O2max,O2min,
```

```
        Vmax,Vmin,Tmax,Tmin,Line1,Line2,
        Line3,Line4);
END. {DAP END}
```

APPENDIX X

BIT.CODE

General Description

Module BIT.CODE contains two 68000 assembly language routines that monitor bit 1 of the GPIO status register (STS available from AD574 A/D) to determine when data from the 12-bit A/D are valid. This code was written in assembly language as a similar Pascal routine was found to execute too slowly. Following is a list of the summarized features of BIT.CODE.

1. Routine BIT_TST within BIT.CODE simply continues to loop on itself until bit 1 of the GPIO status register first goes high and then low. Following completion of the looping process BIT_TST returns back to the calling routine.

2. Routine BIT_HI within BIT.CODE loops on itself until bit 1 of the GPIO status register goes low. Following completion of the looping process BIT_HI returns back to the calling routine.

3. Both of these routines (BIT_TST and BIT_HI) assume that the GPIO interface is connected to the HP9826 computer at select code #12. This insures proper addressing of the STS signal. See Calculations section below for more on the GPIO addressing.

4. BIT.CODE as it presently exists is contained in the Pascal system library.

Calculations

Following are the important calculations that are made by DAP.

1. Absolute address of GPIO interface

Refering to the GPIO section of the PASCAL 2.1 PROCEDURE LIBRARY MANUAL [12], the base address of the GPIO is determined as follows. The HP9826 has 24-bit addressing capability. To address external I/O interfaces bits 20 through 23 (bit 0 being the LSB) must be 0,1,1, and 0 respectively. Bits 16 through 20 corresponds to the interface select code. The GPIO interface currently used has an interface select code of 12. Therefore, bits 16 through 20 must be 0,0,1,1,0 respectively. The bottom 16 bits of the GPIO base address are all zeros. Combining the mentioned bits, the base address of the interface is 7077888. Once the base address of the GPIO interface was located, the system debugger [25] was used to locate the actual address of 16 return bits from the DAM (register #3 within the GPIO). Using the system debugger, the MSB of the 16 return bits from the DAM are located at 7077892 and the LSB is located at 7077893.

2. Determining the status of the STS bit

The STS bit (bit 1 of the GPIO status

register) is polled by simply ANDing the LSB of GPIO status register with 2. Then, depending upon whether or not the result is zero or non-zero one can monitor the status of the STS bit. Thus the code

```
ANDI.B #2,D0
BEQ    BIT_LOW
```

will loop back to BIT_LOW so long as bit 1 is low. For the case when a loop is desired when bit 1 is high

```
ANDI.B #2,D0
BNE    BIT_HI
```

will work.

```
*****************************************************************************
*
*       BIT TEST MODULE FOR BIT 1 OF GPIO STATUS REGISTER
*
*       SOURCE FILENAME:  BIT.TEXT
*
*       DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
*
*       KANSAS STATE UNIVERSITY
*
*       REVISION        DATE                    PROGRAMMER
*       --------        ----                    ----------
*         1.0           JUNE 29, 1984           LOREN E. RIBLETT, JR.
*
*****************************************************************************
*
*       PURPOSE
*
*               THIS MODULE MONITORS BIT 1 OF THE GPIO STATUS REGISTER
*               (STS SIGNAL FROM AD574 A/D) TO DETERMINE WHEN VALID DATA
*               FROM THE 12-BIT A/D IS AVAILABLE.
*
*       ROUTINE(S) CALLED BY THIS MODULE
*
*               NONE
*
*****************************************************************************
*
*       NOTE 1:  ROUTINE BIT_TST WITHIN BIT.CODE SIMPLY CONTINUES TO LOOP
*                ON ITSELF UNTIL BIT 1 OF THE GPIO STATUS REGISTER FIRST
*                GOES HIGH THEN LOW.  FOLLOWING COMPLETION OF THE LOOPING
*                PROCESS BIT_TST RETURNS BACK TO THE CALLING ROUTINE.
*
*       NOTE 2:  ROUTINE BIT_HI WITHIN BIT.CODE LOOPS ON ITSELF UNTIL BIT 1
*                OF THE GPIO STATUS REGISTER GOES LOW.  FOLLOWING COMPLETION
*                OF THE LOOPING PROCESS BIT_HI RETURNS BACK TO THE CALLING
*                ROUTINE.
*
*       NOTE 3:  BOTH OF THESE ROUTINES (BIT_TST AND BIT_HI) ASSUME THAT THE
*                GPIO INTERFACE IS CONNECTED TO THE HP9826 COMPUTER AT SELECT
*                CODE #12.  THIS INSURES PROPER ADDRESSING OF THE STS SIGNAL.
*
*       NOTE 4:  BIT.CODE AS IT PRESENTLY EXISTS IS CONTAINED IN THE PASCAL
*                SYSTEM LIBRARY.
*
*****************************************************************************
        SPC     1
        MNAME   BIT                     *DECLARE MODULE NAME
        DEF     BIT_TST                 *DEFINE ENTRY POINTS INTO MODULE
        DEF     BIT_HI
        DEF     BIT_BIT
        RORG    0                       *DEFINE ORIGIN OF PROGRAM
GPINT   EQU     7077893                 *ADDRESS OF LSB OF GPIO STATUS REGISTER
        SPC     2
BIT_TST EQU     *                       *BEGIN BIT_TST PROCEDURE
```

```
BIT_LOW MOVE.B GPINT,D0          *LOOP UNTIL BIT 1 OF GPIO IS HIGH
        ANDI.B #2,D0
        BEQ    BIT_LOW
BIT_HI  MOVE.B GPINT,D0          *LOOP UNTIL BIT 1 OF GPIO IS LOW
        ANDI.B #2,D0
        BNE    BIT_HI
BIT_BIT RTS                      *RETURN BACK TO CALLING ROUTINE
        END                      *END OF BIT MODULE
```

APPENDIX XI

AUTOST

General Description

AUTOST is an HP BASIC auto start routine which allows the user to select (from a menu) those BASIC programs that currently exist for the human respiratory research. Features of AUTOST include:

1. Because this is an HP AUTOST routine, it should appear only on the 5.25" floppy used in the mass storage unit ":INTERNAL".

2. The program is initiated by placing the floppy in the mentioned drive and applying power to the HP9826.

3. AUTOST is recalled upon completion of the called routines CAPCRUNCH, DAPCRUNCH, and ANALYSIS.

4. The various functions AUTOST is capable of performing are assigned to special function keys. Key assignments include:

A. k0 - loads and executes the BASIC routine CAPCRUNCH.

B. k1 - loads and executes the BASIC routine DAPCRUNCH.

C. k2 - loads and executes the BASIC routine ANALYSIS.

D. k5 - changes the mass storage unit specifier.

E. k6 - loads and executes a user selected routine.

F. k7 - performs a catalog listing for the current mass storage device.

G. k9 - causes AUTOST to end.

These key assignments allow for single key stroke command entry. Also, little if any knowledge concerning where various programs are stored is required as AUTOST automatically loads and executes the desired routines.

Variable List

A — An integer representing the desired mass storage device as specified in the mass storage menu.

Dev$(5)[15] — A string array consisting of 5 elements, 15 bytes long containing the 5 mass storage unit specifiers available for use by the respiratory routines.

I — An integer used strictly as a FOR/NEXT loop counter.

Lo$ — A string variable representing the name of the user defined program to load and execute.

```
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 PROGRAMS.
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  !
280  !************************************************************************
290  !
300  !       NOTE 1:  Because this is an AUTOST routine, it should appear
310  !                only on the 5.25" floppy used in the mass storage unit
320  !                ":INTERNAL".
330  !
340  !       NOTE 2:  The program is initiated by placing the floppy in the
350  !                mentioned drive and applying power to the HP9826.
360  !
370  !       NOTE 3:  This program is recalled upon completion of the called
380  !                routines.
390  !
400  !************************************************************************
410  !
420  !*** DECLARATIONS
430  !
440  DIM Dev$(5)[15]
450  !
460  !*** ASSIGN SPECIAL FUNCTION KEYS
470  !
480  ON KEY 0 LABEL "CCRUNCH" GOTO Ccrunch
490  ON KEY 1 LABEL "DCRUNCH" GOTO Dcrunch
500  ON KEY 2 LABEL "ANALYSIS" GOTO Analysis
510  ON KEY 5 LABEL "New MSI" GOTO Msi
520  ON KEY 6 LABEL "  LOAD" GOTO Load
530  ON KEY 7 LABEL "  CAT" GOTO Cat
540  ON KEY 9 LABEL "  EXIT" GOTO Exit
550  GOTO 480
```

```
560 !
570 !*** CALL TO DAPCRUNCH
580 !
590 Dcrunch:!
600          OFF KEY
610          MASS STORAGE IS ":HP9895,702,2"
620          LOAD "DAPCRUNCH",1
630 !
640 !*** CALL TO CAPCRUNCH
650 !
660 Ccrunch:!
670          OFF KEY
680          MASS STORAGE IS ":HP9895,702,2"
690          LOAD "CAPCRUNCH",1
700 !
710 !*** CALL TO ANALYSIS
720 !
730 Analysis:!
740           OFF KEY
750           MASS STORAGE IS ":HP9895,702,2"
760           LOAD "ANALYSIS",1
770 !
780 !*** CATALOG REQUEST
790 !
800 Cat:!
810      OFF KEY
820      CAT
830      GOTO 480
840 !
850 !*** NEW MASS STORAGE REQUEST
860 !
870 Msi:!
880      OFF KEY
890      PRINT CHR$(12)
900      PRINT "Select the device you wish to use"
910      Dev$(1)=":INTERNAL"
920      Dev$(2)=":HP9895,700,0"
930      Dev$(3)=":HP9895,702,1"
940      Dev$(4)=":HP9895,702,2"
950      Dev$(5)=":HP9895,702,3"
960      FOR I=1 TO 5
970      PRINT USING "3X,D,29A,15A";I," -is the mass storage device ",Dev$(I)
980      NEXT I
990      INPUT "ENTER the desired device´s corresponding number.",A
1000     IF A<1 OR A>5 THEN Msi
1010     MASS STORAGE IS Dev$(A)
1020     PRINT CHR$(12)
1030     GOTO 480
1040!
1050!*** LOAD PROGRAM REQUEST
1060!
1070 Load:!
1080      OFF KEY
1090      PRINT "ENTER THE NAME OF THE PROGRAM YOU WISH TO LOAD"
1100      INPUT Lo$
```

```
1110      LOAD Lo$
1120      GOTO 480
1130!
1140!*** PROGRAM TERMINATION REQUEST
1150!
1160 Exit:!
1170      OFF KEY
1180      END
```

APPENDIX XII

CAPCRUNCH

General Description

CAPCRUNCH is an HP BASIC routine which converts the calibration file created by the Pascal routine CAP.CODE from ASCII to binary, allowing for more efficient data storage. To give a more complete description of CAPCRUNCH, a list of summarized features is given below.

1. This routine assumes that the ASCII files to be converted are stored on hard disk ":HP9895,702,2".

2. Converted files (binary) are stored on the 8" floppy ":HP9895,700,0". Care should be exercised by the user to insure that the 8" floppy has adequate room for the converted files. See Appendix I for more details.

3. Calibration file names should not exceed 9 characters in length. Calibration file names are actually established in CAP.CODE.

4. The ASCII calibration files are purged (deleted) following the conversion process.

5. The auto start routine AUTOST is called following completion of CAPCRUNCH.

6. The various functions CAPCRUNCH is capable of performing are assigned to special function keys. Key assignments include:

A. k0 - allows CAPCRUNCH to crunch a calibration file.

B. k9 - causes CAPCRUNCH to exit and AUTOST to be called.

These key assignments allow for single key stroke command entry. Also, should the user enter CAPCRUNCH by accidentally pressing "k0" in AUTOST, pressing "k9" in CAPCRUNCH allows the operator to exit without attempting to crunch any calibration files.

Calculations

Only two calculations are performed in CAPCRUNCH. One relates to changing file names from Pascal to BASIC format and the other determines the binary data file size.

1. Pascal to BASIC file name conversion

   File names listed in the Pascal operating system directory as "FNAME.ASC" appear as "FNAMEA___" in the BASIC operating system. In other words, the file name along with the first letter of the extension (.ASC) are combined and the "_" character then fills the file name to 10 characters. As can be seen from this renaming scheme, file names longer than nine characters in the Pascal operating system are not recommended. As far as the program user is concerned, the ASCII calibration file is given the name "FNAME" and once the crunch on the calibration file is complete, the binary data file stored on the HP9895A 8" floppy disk will appear as "FNAME" in both the Pascal and BASIC operating system directories.

2. Binary data file size determination

Binary data file sizes are determined realizing that 8 bytes of mass storage is required to store a single real variable and 1 byte per character is required for string variables. Thus, for the 13 calibration constants (real variables) and 25 character date, the number of mass storage bytes needed is

bytes = (13 * 8) + 25

File Structure

The serial ASCII calibration files created by CAP.CODE are organized into single record files using the following format. (NOTE: These files will appear on the ":HP9895,702,2" hard disk and are purged immediately following the crunch procedure.)

Record #	Contents
1	Co2_dc_offset$ (4 bytes)
	O2_dc_offset$ (4 bytes)
	Bin_zero_flow$ (4 bytes)
	Co2_cal$ (25 bytes)
	O2_cal$ (25 bytes)
	Insp_flow_cal$ (25 bytes)
	Expr_flow_cal$ (25 bytes)
	Time_delay$ (25 bytes)
	S$ (4 bytes)
	O1$ (25 bytes)
	Ta$ (25 bytes)
	Tb$ (25 bytes)
	Tc$ (25 bytes)
	Date$ (25 bytes)

The serial BDAT (Binary DATa) calibration files created by CAPCRUNCH are organized into single record files using the following format. (NOTE: The following file is created on the ":HP9895,700,0" 8" flexible disk.)

Record #	Contents
1	Co2_dc_offset (INTEGER)
	O2_dc_offset (INTEGER)
	Bin_zero_flow (INTEGER)
	Co2_cal (REAL)
	O2_cal (REAL)
	Insp_flow_cal (REAL)
	Expr_flow_cal (REAL)
	Time_delay (REAL)
	S (INTEGER)
	O1 (REAL)
	Ta (REAL)
	Tb (REAL)
	Tc (REAL)
	Date$ (Character string, 25 bytes)

Once the calibration files appear in the format shown above they are compatible with ANALYSIS (the breath-by-breath analysis routine) and can be used in converting the BCD data collected by the DAM to real world units.

Variable List

Bin_zero_flow$[4] ASCII representation of the average BCD value of 100 samples obtained from the DAM flow channel with zero flow connected to the pneumotach.

Co2_cal$[25]	ASCII representation of the calibration factor for the CO_2 channel (channel A of the DAM). Multiplying digital CO_2 data by this factor yields fractional CO_2 concentration units.
Co2_dc_offset$[4]	ASCII representation of the average BCD value of 1000 samples obtained from the CO_2 channel with the mass spectrometer probe connected to room air.
Date$[25]	Character string containing the date that a particular calibration file was created.
Expr_flow_cal$[25]	ASCII representation of the expiratory flow calibration factor (channel C of the DAM). Multiplying digital expiratory flow data by this factor yields flow units of l/s.
Insp_flow_cal$[25]	ASCII representation of the inspiratory flow calibration factor (channel C of the DAM). Multiplying digital inspiratory flow data by this factor yields flow units of l/s.
N$[10]	Character string containing the name of the calibration file as it was named by the user in the Pascal operating system.

N1$[10]	Character string containing the calibration file name as it appears in the BASIC operating system (having been created in the Pascal operating system).
O2_dc_offset$[4]	ASCII representation of the average BCD value of 1000 samples obtained from the O_2 channel with the mass spectrometer probe connected to 13% O_2.
O2_cal$[25]	ASCII representation of the calibration factor for the O_2 channel (channel B of the DAM). Multiplying digital O_2 data by this factor yields fractional O_2 concentration units.
O1$[25]	ASCII representation of the actual value displayed by the mass spectrometer when the sampling probe is connected to 13% O_2.
S$[4]	ASCII representation of the sampling frequency for channels A, B, C, and D of the DAM.
Ta$[25]	ASCII representation of the 2nd order polynomial coefficient used to convert digital temperature data to units of degrees C.

Tb$[25]	ASCII representation of the 1st order polynomial coefficient used to convert digital temperature data to units of degrees C.
Tc$[25]	ASCII representation of the constant term used to convert digital temperature data to units of degrees C.
Time_delay$[25]	ASCII representation of the mass spectrometer time delay in milliseconds.

```
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  !
140  !************************************************************************
150  !
160  !       PURPOSE
170  !
180  !               THIS ROUTINE CONVERTS THE CALIBRATION FILE CREATED BY
190  !               THE PASCAL ROUTINE "CAP.CODE" FROM ASCII TO BINARY,
200  !               ALLOWING FOR MORE EFFICIENT DATA STORAGE.
210  !
220  !       ROUTINE(S) CALLED BY THIS ROUTINE
230  !
240  !               AUTOST - USER PROGRAM ACCESS ROUTINE
250  !
260  !************************************************************************
270  !
280  !       NOTE 1:  This routine assumes that the ASCII files to be converted
290  !                are stored on  ":HP9895,702,2".
300  !
310  !       NOTE 2:  Converted files (binary) will be stored on the 8" floppy
320  !                ":HP9895,700,0".
330  !
340  !       NOTE 3:  File names in excess of 9 characters should not be used.
350  !
360  !       NOTE 4:  ASCII files are purged following the conversion process.
370  !
380  !       NOTE 5:  AUTOST is called following completion of CAPCRUNCH.
390  !
400  !************************************************************************
410  !
420  !*** ASSIGN SPECIAL FUNCTION KEYS
430  !
440  ON KEY 0 LABEL "CCRUNCH" GOTO 500
450  ON KEY 9 LABEL "  EXIT" GOTO Done
460  GOTO 440
470  !
480  !*** MAKE DECLARATIONS
490  !
500  OFF KEY
510  DIM Co2_dc_offset$[4],02_dc_offset$[4],Bin_zero_flow$[4]
520  DIM Co2_cal$[25],02_cal$[25],Insp_flow_cal$[25]
530  DIM Expr_flow_cal$[25],Time_delay$[25],S$[4]
540  DIM 01$[25],Ta$[25],Tb$[25],Tc$[25],Date$[25]
550  DIM N$[10],N1$[10]
```

```
560    !
570    !*** GET FILE NAME TO CRUNCH
580    !
590    MASS STORAGE IS ":HP9895,702,2"
600    BEEP
610    INPUT "ENTER NAME OF CALIBRATION FILE TO CRUNCH:  ",N$
620    !
630    !*** ALTER FILE NAME TO REFLECT PASCAL TO BASIC NAME CHANGES
640    !
650    N1$=N$&"A"
660 Fill: IF LEN(N1$)>=10 THEN GOTO Full
670       N1$=N1$&"_"
680       GOTO Fill
690 !
700 !*** READ IN ASCII VALUES FROM CALIBRATION FILE
710 !
720 Full:  ASSIGN @Path1 TO N1$
730        ENTER @Path1;Co2_dc_offset$,02_dc_offset$
740        ENTER @Path1;Bin_zero_flow$,Co2_cal$
750        ENTER @Path1;02_cal$,Insp_flow_cal$
760        ENTER @Path1;Expr_flow_cal$,Time_delay$,S$
770        ENTER @Path1;01$,Ta$,Tb$,Tc$,Date$
780        !
790        !*** CREATE BINARY FILE ON 8" FLOPPY
800        !
810        MASS STORAGE IS ":HP9895,700,0"
820        CREATE BDAT N$,1,8*13+25
830        !
840        !*** WRITE BINARY CALIBRATION CONSTANTS TO FILE
850        !
860        ASSIGN @Path1 TO N$
870        OUTPUT @Path1;VAL(Co2_dc_offset$),VAL(02_dc_offset$)
880        OUTPUT @Path1;VAL(Bin_zero_flow$),VAL(Co2_cal$)
890        OUTPUT @Path1;VAL(02_cal$),VAL(Insp_flow_cal$)
900        OUTPUT @Path1;VAL(Expr_flow_cal$),VAL(Time_delay$)
910        OUTPUT @Path1;VAL(S$),VAL(01$),VAL(Ta$)
920        OUTPUT @Path1;VAL(Tb$),VAL(Tc$),Date$
930        MASS STORAGE IS ":HP9895,702,2"
940        !
950        !*** DELETE OLD ASCII CALIBRATION FILE
960        !
970        PURGE N1$
980        PRINT "CALIBRATION FILE COMPACTION COMPLETE."
990 !
1000!*** RETURN TO AUTOST
1010!
1020 Done:  OFF KEY
1030        MASS STORAGE IS ":INTERNAL"
1040        LOAD "AUTOST",1
1050        END
```

APPENDIX XIII

DAPCRUNCH

General Description

DAPCRUNCH is an HP BASIC routine which converts the four ASCII files of DAM data from the Pascal routine DAP.CODE (namely the CO_2, O_2, flow, and temperature data files) from ASCII to binary, allowing for more efficient data storage. To give a more complete description of CAPCRUNCH, a list of summarized features is given below.

1. This routine assumes that the ASCII files to be converted are stored on the ":HP9895,702,2" hard disk. These files must be named "MONSTER1A_", "MONSTER2A_", "MONSTER3A_", AND "MONSTER4A_" in the BASIC operating system (in the Pascal operating system these files would appear as "MONSTER1.ASC", "MONSTER2.ASC", etc.) See Appendix XII for more information on Pascal to BASIC file name conversion. The data from MONSTER1A_ are assumed to be CO_2 data (channel A), MONSTER2A_ are O_2 data (channel B), MONSTER3A_ are flow data (channel C), and MONSTER4A_ are temperature data (channel D).

2. Converted files (binary) are stored on the 8" floppy ":HP9895,700,0". Care should be exercised by the user to insure that the 8" floppy has adequate room for the converted files. See Appendix I for more details.

3. Binary data file names in excess of 10 characters should not be used.

4. The ASCII data files are purged (deleted) following the conversion process.

5. The auto start routine AUTOST is called following completion of DAPCRUNCH.

6. The various functions DAPCRUNCH is capable of performing are assigned to special function keys. Key assignments include:

A. k1 - allows DAPCRUNCH to crunch four ASCII data files.

B. k9 - causes DAPCRUNCH to exit and AUTOST to be called.

These key assignments allow for single key stroke command entry. Also, should the user enter DAPCRUNCH by accidentally pressing "k1" in AUTOST, pressing "k9" in DAPCRUNCH allows the operator to exit without attempting to crunch any data files.

Calculations

Only one type of calculation is performed in DAPCRUNCH. It (the calculation) is related to the determination of the binary data file size for the four DAM data files.

1. Binary data file size determination

Binary data file sizes are determined realizing that 2 bytes of mass storage is required to store a single integer variable. Thus, for n DAM data points (integers) plus the maximum and minimum data points within the n data points (see File

Structure section that follows) the number of mass storage bytes needed is

bytes = (n * 2) + 4

File Structure

The four serial ASCII data files created by DAP.CODE are organized into single record files using the following format. (NOTE: These files will appear on the ":HP9895,702,2" hard disk and are purged immediately following the crunch procedure.)

File	Record #	Contents
CO_2	1	CO_2 channel maximum (4 ASCII bytes) CO_2 channel minimum (4 ASCII bytes) n CO_2 channel data points (4 ASCII bytes per point)
O_2	1	O_2 channel maximum (4 ASCII bytes) O_2 channel minimum (4 ASCII bytes) n O_2 channel data points (4 ASCII bytes per point)
Flow	1	Flow channel maximum (4 ASCII bytes) Flow channel minimum (4 ASCII bytes) n flow channel data points (4 ASCII bytes per point)
Temperature	1	Temperature channel maximum (4 ASCII bytes) Temperature channel minimum (4 ASCII bytes) n temperature channel data points (4 bytes per point)

The serial BDAT (Binary DATa) data files created by DAPCRUNCH are organized into single record files using the following format. (NOTE: The following files are created on the ":HP9895,700,0" 8" flexible disk.)

File	Record #	Contents
CO_2	1	CO_2 channel maximum (2 bytes)
		CO_2 channel minimum (2 bytes)
		n CO_2 channel data points (2 bytes per point)
O_2	1	O_2 channel maximum (2 bytes)
		O_2 channel minimum (2 bytes)
		n O_2 channel data points (2 bytes per point)
Flow	1	Flow channel maximum (2 bytes)
		Flow channel minimum (2 bytes)
		n flow channel data points (2 bytes per point)
Temperature	1	Temperature channel maximum (2 bytes)
		Temperature channel minimum (2 bytes)
		n temperature channel data points (2 bytes per point)

Once the data files appear in the format shown above they are compatible with ANALYSIS (the breath-by-breath analysis routine) and can be used by ANALYSIS for plotting or breath-by-breath analysis.

Variable List

Num	INTEGER variable representing the number of data points per file to crunch. This number is supplied to DAPCRUNCH by the user.
A$(1:Num+2)[4]	ASCII array containing the 4 byte channel data from any one of the four DAM channel files plus the maximum and minimum digital values for that particular channel (i.e., A$(1)= CO_2 maximum, A$(2)= CO_2 minimum, and A$(3...Num+2) = ASCII data points 1 thru Num).
Line(Num+2)	INTEGER array variable containing those values converted from A$ (see above). This array is eventually written to respective binary data files.

```
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   !
140   !************************************************************************
150   !
160   !        PURPOSE
170   !                THIS ROUTINE CONVERTS FOUR ASCII FILES OF DAM DATA (NAMELY
180   !                THE CO2, O2, FLOW, AND TEMPERATURE DATA FILES) CREATED
190   !                BY "DAP.CODE" TO FOUR FILES OF BINARY DATA, ALLOWING FOR
200   !                MORE EFFICIENT DATA STORAGE.
210   !
220   !        ROUTINE(S) CALLED BY THIS ROUTINE
230   !
240   !                AUTOST - USER PROGRAM ACCESS ROUTINE
250   !
260   !************************************************************************
270   !
280   !        NOTE 1:  This routine assumes that the ASCII files to be converted
290   !                 are stored on ":HP9895,702,2".  These files must be named
300   !                 MONSTER1A_, 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   !*** SPECIAL FUNCTION KEY DEFINITIONS
490   !
500   OPTION BASE 1
510   ON KEY 1 LABEL "DCRUNCH" GOTO 570
520   ON KEY 9 LABEL "  EXIT" GOTO Done
530   GOTO 510
540   !
550   !*** GET NUMBER OF DATA POINTS AND MAKE APPROPRIATE DIMENSIONS
```

```
560   !
570   OFF KEY
580   BEEP
590   INPUT "ENTER NUMBER OF DATA POINTS TO CRUNCH:  ",Num
600   ALLOCATE A$(1:Num+2)[4]
610   ALLOCATE INTEGER Line(Num+2)
620   !
630   !*** READ ASCII DATA FILE MONSTER1A_
640   !
650   MASS STORAGE IS ":HP9895,702,2"
660   ASSIGN @Path1 TO "MONSTER1A_"
670   ENTER @Path1;A$(*)
680   !
690   !*** CONVERT ASCII DATA TO BINARY
700   !
710   FOR I=1 TO Num+2
720   Line(I)=VAL(A$(I))
730   NEXT I
740   !
750   !*** CREATE BINARY DATA FILE FOR CO2 DATA
760   !
770   BEEP
780   INPUT "ENTER NAME OF CO2 BINARY DATA FILE",C1$
790   MASS STORAGE IS ":HP9895,700,0"
800   CREATE BDAT C1$,1,2*Num+4
810   ASSIGN @Path1 TO C1$
820   ON END @Path1 GOTO 870
830   !
840   !*** WRITE BINARY DATA TO CO2 FILE AND DELETE ASCII CO2 FILE
850   !
860   OUTPUT @Path1;Line(*)
870   MASS STORAGE IS ":HP9895,702,2"
880   PURGE "MONSTER1A_"
890   !
900   !*** READ ASCII DATA FILE MONSTER2A_
910   !
920   ASSIGN @Path1 TO "MONSTER2A_"
930   ENTER @Path1;A$(*)
940   !
950   !*** CONVERT ASCII DATA TO BINARY
960   !
970   FOR I=1 TO Num+2
980   Line(I)=VAL(A$(I))
990   NEXT I
1000  !
1010  !*** CREATE BINARY DATA FILE FOR O2 DATA
1020  !
1030  BEEP
1040  INPUT "ENTER NAME OF O2 BINARY DATA FILE:  ",O$
1050  MASS STORAGE IS ":HP9895,700,0"
1060  CREATE BDAT O$,1,2*Num+4
1070  ASSIGN @Path1 TO O$
1080  ON END @Path1 GOTO 1130
1090  !
1100  !*** WRITE BINARY DATA TO O2 FILE AND DELETE ASCII O2 FILE
```

```
1110  !
1120  OUTPUT @Path1;Line(*)
1130  MASS STORAGE IS ":HP9895,702,2"
1140  PURGE "MONSTER2A_"
1150  !
1160  !*** READ ASCII DATA FILE MONSTER3A_
1170  !
1180  ASSIGN @Path1 TO "MONSTER3A_"
1190  ENTER @Path1;A$(*)
1200  !
1210  !*** CONVERT ASCII DATA TO BINARY
1220  !
1230  FOR I=1 TO Num+2
1240  Line(I)=VAL(A$(I))
1250  NEXT I
1260  !
1270  !*** CREATE BINARY DATA FILE FOR FLOW DATA
1280  !
1290  BEEP
1300  INPUT "ENTER NAME OF FLOW BINARY DATA FILE:  ",V$
1310  MASS STORAGE IS ":HP9895,700,0"
1320  CREATE BDAT V$,1,2*Num+4
1330  ASSIGN @Path1 TO V$
1340  ON END @Path1 GOTO 1390
1350  !
1360  !*** WRITE BINARY DATA TO FLOW FILE AND DELETE ASCII FLOW FILE
1370  !
1380  OUTPUT @Path1;Line(*)
1390  MASS STORAGE IS ":HP9895,702,2"
1400  PURGE "MONSTER3A_"
1410  !
1420  !*** READ ASCII DATA FILE MONSTER4A_
1430  !
1440  ASSIGN @Path1 TO "MONSTER4A_"
1450  ENTER @Path1;A$(*)
1460  !
1470  !*** CONVERT ASCII DATA TO BINARY
1480  !
1490  FOR I=1 TO Num+2
1500  Line(I)=VAL(A$(I))
1510  NEXT I
1520  !
1530  !*** CREATE BINARY DATA FILE FOR TEMPERATURE DATA
1540  !
1550  BEEP
1560  INPUT "ENTER NAME OF TEMPERATURE BINARY DATA FILE:  ",T$
1570  MASS STORAGE IS ":HP9895,700,0"
1580  CREATE BDAT T$,1,2*Num+4
1590  ASSIGN @Path1 TO T$
1600  ON END @Path1 GOTO 1650
1610  !
1620  !*** WRITE BINARY DATA TO TEMPERATURE FILE AND DELETE ASCII FILE
1630  !
1640  OUTPUT @Path1;Line(*)
1650  MASS STORAGE IS ":HP9895,702,2"
```

```
1660  PURGE "MONSTER4A_"
1670  !
1680  !*** RETURN TO AUTOST
1690  !
1700  PRINT "DATA FILE COMPACTION COMPLETE."
1710 Done: OFF KEY
1720       MASS STORAGE IS ":INTERNAL"
1730       LOAD "AUTOST",1
1740       END
```

APPENDIX XIV

ANALYSIS

General Description

ANALYSIS is an HP BASIC routine which, in addition to containing the code necessary to plot any window (section) of the collected data, performs breath- by-breath analysis on the data collected from the DAM. Following is a list of summarized features of ANALYSIS.

1. This routine assumes that the binary calibration files created by CAPCRUNCH are stored on the HP9895A 8" flexible disk ":HP9895,700,0" (volume #7 in the Pascal operating system).

2. ANALYSIS assumes that the four binary data files created by DAPCRUNCH are also stored on the HP9895A 8" flexible disk ":HP9895,700,0" (volume #7 in the Pascal operating system).

3. Gas mass spectrometer time delay values can be determined on a breath-by-breath basis or a fixed time delay can be selected for analysis of the acquired data. Should an extreme breath-by-breath time delay be calculated, an average time delay is substituted for the computed time delay.

4. Respiratory volumes can be corrected to STPD/BTPS conditions provided the barometric pressure (torr), relative humidity (%), and body temperature (deg

C) is supplied. Using this information along with point-by-point temperature correction (using channel D, the respiratory temperature channel), the inspiratory and expiratory gas volumes are corrected.

5. Any window of data from the DAM data may be plotted either on the HP9826 CRT or HP9872C plotter. All four channels of data are plotted, the CO_2 and O_2 channels being plotted with a time delay equal to the average time delay entered by the user. Data plotted on the HP9826 CRT may also be dumped to the HP2673A thermal printer.

6. Once breath-by-breath analysis of the DAM data begins, information concerning each breath is printed on the DECwriter printer. See section 4.5 Data Analysis and Display Software for an example of the hard copy output.

7. Once the data arrays are exhausted summary data for the entire run is computed and printed. File names as well as critical calibration parameters are also printed.

Calculations

Following are the important calculations that are made by ANALYSIS.

1. Vapor pressure at given temperature

In order to convert volumes (or flows) to STPD/BTPS conditions a data file named "VAP" is read by ANALYSIS. VAP contains water vapor pressures from 20.0 to 44.9 deg C in 0.1 deg C increments. Thus the equation

```
Ph2o_body=Vap((Body_temp-20)*10)
```

will determine the vapor pressure of water at the given body temperature.

2. Positive to negative transition of flow signal

   To locate positive to negative transition of the flow signal (beginning of inspiration) the array index must point to a flow value that is less than or equal to the binary zero flow and the next five flow points must be less than binary zero flow. Thus, the statements

```
IF (Line3(A)-B>0) OR (Line3(A+1)-B>=0) OR
        (Line3(A+2)-B>=0) THEN 2960
IF (Line3(A+3)-B<0) AND (Line3(A+4)-b<0) AND
        (Line3(A+5)-B<0) THEN Decre
```

   will not allow the program to exit from the current expiration calculations until the mentioned flow criterion is met.

3. Negative to positive transition of flow signal

   To locate negative to positive transition of the flow signal (beginning of expiration) the array index must point to a flow value that is greater than or equal to the binary zero flow and the next five flow points must be greater than binary zero flow. Thus, the statements

```
IF (Line3(A)-B<0) OR (Line3(A+1)-B<=0) OR
        (Line3(A+2)-B<=0) THEN 2490
IF (Line3(A+3)-B>0) AND (Line3(A+4)-B>0) AND
        (Line3(A+5)-B>0) THEN Decri
```

does not allow the program to exit from the current inspiration calculations until the mentioned flow criterion is met.

4. Bad breath-by-breath time delays

If the breath-by-breath time delay subroutine (Bbb_time_delay) calculates a mass spectrometer time delay less than 330 ms or greater than 560 ms ANALYSIS ignores the computed time delay and substitutes a running average of the previously computed time delays that fall within the mentioned time limits. The running average (Avg_time_delay) is computed as follows.

```
Time_delay_sum=Time_delay_sum+Time_delay
Time_delay_cnt=Time_delay_cnt+1
Avg_time_delay=Time_delay_sum/Time_delay_cnt
```

This section of code is not executed if the computed time delay falls outside the 330-560 ms range and the substitution

Time_delay=Avg_time_delay

is made.

5. Point-by-point temperature calculation

For successful conversion to STPD/BTPS the temperature channel data (channel D) must be converted to units of degrees C. Using the 2nd order coefficients (Ta, Tb, and Tc) determined by the calibration routine, this conversion is made using the following equation.

Temp=Ta*Line4(A)^2 + Tb*Line4(A) + Tc

6. Inspiratory BTPS conversion constant

For conversion of inspiratory data to BTPS conditions the following conversion constant is necessary.

Insp_btps=(Pb-Rel_humid*Ph2o_insp)/(Pb-Ph2o_body)
*(273+Body_temp)/(273+Temp)

7. Inspiratory STPD conversion constant

For conversion of inspiratory data to STPD conditions the following conversion constant is necessary.

```
Insp_stpd=(Pb-Rel_humid*Ph2o_insp)/760*273/
          (273+Temp)
```

8. Inspiratory volumes of air, CO_2, and O_2

The trapezoidal rule for integration is used to compute inspiratory gas volumes from the flow signal (channel C). Following are the three general equations used for this integration process.

```
Airi=Airi+(Line3(A)-B)*Flow_cal*Inps_btps*T
Co2i=Co2i+(Line1(Z)-Co2_dc_offset)*(Line3(A)-B)
     *Flow_cal*Co2_cal*Insp_stpd*T
O2i=O2i+((Line2(Z)-O2_dc_offset)*O2_cal+O1)
   *Insp_stpd*(Line3(A)-B)*Flow_cal*T
```

As is consistent with the trapezoidal rule only half of the first and last trapezoid areas are included in the mentioned summing process.

9. Time of inspiration

Inspiratory time is simply determined by multiplying the number of points acquired during inspiration by the sampling period of the DAM. Following is the equation used for this calculation.

```
Insp_time=(A-Insp_count)*T
```

10. Expiratory BTPS conversion constant

For conversion of expiratory data to BTPS conditions the following conversion constant is necessary.

```
Expr_btps=(Pb-Ph2o_expr)/(Pb-Ph2o_body)
          *(273+Body_temp)/(273+Temp)
```

11. Expiratory STPD conversion constant

For conversion of expiratory data to STPD conditions the following conversion constant is necessary.

```
Expr_stpd=(Pb-Ph2o_expr)/760*273/(273+Temp)
```

12. Expiratory volumes of air, CO_2, and O_2

The trapezoidal rule for integration is used to compute expiratory gas volumes from the flow signal (channel C). Following are the three general equations used for this integration process.

```
Aire=Aire+(Line3(A)-B)*Flow_cal*Expr_btps*T
Co2e=Co2e+(Line1(Z)-Co2_dc_offset)*(Line3(A)-B)
     *Flow_cal*Co2_cal*Expr_stpd*T
O2e=O2e+((Line2(Z)-O2_dc_offset)*O2_cal+O1)
    *Expr_stpd*(Line3(A)-B)*Flow_cal*T
```

As is consistent with the trapezoidal rule only half of the first and last trapezoid areas are included in the mentioned summing process.

13. Time of expiration

Expiratory time is simply determined by multiplying the number of points acquired during expiration by the sampling period of the DAM. Following is the equation used for this calculation.

Exp_time=(A-Exp_count)*T

14. O_2 consumed for this breath

Oxygen consumption for a particular breath is found simply by adding the O_2 inspired to the O_2 expired value. A negative value for O_2 consumption simply indicates oxygen is being consumed. Following is the equation for determining O_2 consumption.

O2cons=O2i+O2e

15. CO_2 produced for this breath

CO_2 production for a particular breath is found by adding the CO_2 inspired to the CO_2 expired value. A positive value for CO_2 production indicates CO_2 is being produced. Following is the equation for determining CO_2 production.

```
Co2prod=Co2i+Co2e
```

16. Total inspiratory volumes

Total inspiratory volumes are determined by adding the inspiratory volumes for the individual breaths. Following are those equations used for calculating these inspiratory volumes.

```
Tot_vol_insp=Tot_vol_insp+Airi
Tot_o2_insp=Tot_o2_insp+O2i
Tot_co2_insp=Tot_co2_insp+Co2i
```

17. Total expiratory volumes

Total expiratory volumes are determined by adding the expiratory volumes for the individual breaths. Following are those equations used for calculating these expiratory volumes.

```
Tot_vol_exp=Tot_vol_exp+Aire
Tot_o2_exp=Tot_o2_exp+O2e
Tot_co2_exp=Tot_co2_exp+Co2e
```

18. Total O_2 consumption and CO_2 production

Total O_2 consumption and CO_2 production values are determined by adding consumption and production values for the individual breaths. Following are those equations.

Tot_o2_cons=Tot_o2_cons+O2cons

Tot_co2_prod=Tot_co2_prod+Co2prod

19. Total time of inspiration

Total time of inspiration is found by multiplying the total number of inspiratory data points by the period of sampling used. The equation to calculate total inspiratory time follows.

Tot_time_insp=Tot_insp_points*T

20. Total time of expiration

Total time of expiration is found by multiplying the total number of expiratory data points by the period of sampling used. The equation to calculate total expiratory time follows.

Tot_time_exp=Tot_exp_points*T

21. Total time of respiration

Total respiration time is found by multiplying the total number of data points analyzed by the period of sampling used. The equation to calculate total respiration time follows.

Tot_time_resp=(Final_index-Init_index)*T

22. Inspiratory minute volume

Inspiratory minute volume is the total volume of air inspired per minute. The equation to calculate inspiratory minute volume follows.

```
Minvoli=Tot_vol_insp*60/Tot_time_resp
```

23. Expiratory minute volume

Expiratory minute volume is the total volume of air expired per minute. The equation to calculate expiratory minute volume follows.

```
Minvole=Tot_vol_exp*60/Tot_time_resp
```

24. Average inspiratory volume

Average inspiratory volume is the total amount of air inspired divided by the number of breaths taken. The equation to calculate average inspiratory volume follows.

```
Avoli=Tot_vol_insp/No_breaths
```

25. Average expiratory volume

Average expiratory volume is the total amount of air expired divided by the number of breaths taken. The equation to calculate average expiratory volume follows.

Avole=Tot_vol_exp/No_breaths

26. Respiratory frequency

The respiratory frequency is found by dividing the total number of breaths by the total time of respiration. Following is the equation used to determine respiratory frequency.

Respf=No_breaths*60/Tot_time_resp

27. Average inspired O_2 (CO_2) per breath

The average O_2 (CO_2) inspired per breath is found simply by dividing total O_2 (CO_2) inspired for the entire trial by the total number of breaths. The equation to calculate average inspired O_2 (CO_2) per breath follows.

O2i_tidal=Tot_o2_insp/No_breaths

Co2i_tidal=Tot_co2_insp/No_breaths

28. Average expired O_2 (CO_2) per breath

The average O_2 (CO_2) expired per breath is found by dividing total O_2 (CO_2) expired for the entire trial by the total number of breaths. The equation to calculate average expired O_2 (CO_2) per breath follows.

O2e_tidal=Tot_o2_exp/No_breaths

```
Co2e_tidal=Tot_co2_exp/No_breaths
```

29. Average O_2 consumed per breath

Average O_2 consumed per breath is found by dividing the total O_2 consumed for the entire trial by the number of breaths taken during the trial. Following is the equation used to calculate average O_2 consumed per breath.

```
Avo2cons=Tot_o2_cons/No_breaths
```

30. Average CO_2 produced per breath

Average CO_2 produced per breath is found by dividing the total CO_2 produced for the entire trial by the number of breaths taken during the trial. Following is the equation used to determine average CO_2 produced per breath.

```
Avco2prod=Tot_co2_prod/No_breaths
```

31. O_2 consumed per minute

O_2 consumed per minute is found by dividing the total O_2 consumed for the entire trial by the total time of respiration. The equation for calculating O_2 consumed per minute follows.

```
V_dot_o2=Tot_o2_cons/Tot_time_resp*60
```

32. CO_2 produced per minute

CO_2 produced per minute is found by dividing the total CO_2 produced for the entire trial by the total time of respiration. The equation for calculating CO_2 produced per minute follows.

```
V_dot_co2=Tot_co2_prod/Tot_time_resp*60
```

33. Respiratory quotient

Respiratory quotient (R) is found as the rate at which CO_2 is produced divided by the rate at which O_2 is consumed. The equation used for calculating R follows.

```
R=ABS(V_dot_co2/V_dot_o2)
```

34. Mass spectrometer time delay

See section 4.5 Data Analysis and Display Software for details.

File Structure

The serial BDAT (Binary DATa) calibration files used by ANALYSIS are organized into single record files using the following format. (NOTE: The following file is located on the ":HP9895,700,0" 8" flexible disk.)

Record #	Contents
1	Co2_dc_offset (INTEGER)
	O2_dc_offset (INTEGER)
	Bin_zero_flow (INTEGER)
	Co2_cal (REAL)
	O2_cal (REAL)
	Insp_flow_cal (REAL)
	Expr_flow_cal (REAL)
	Time_delay (REAL)
	S (INTEGER)
	O1 (REAL)
	Ta (REAL)
	Tb (REAL)
	Tc (REAL)
	Date$ (STRING, 25 bytes)

The four serial BDAT (Binary DATa) data files used by ANALYSIS are organized into single record files using the following format. (NOTE: The following file is located on the ":HP9895,700,0" 8" flexible disk.)

File	Record #	Contents
CO_2	1	CO_2 channel maximum (INTEGER) CO_2 channel minimum (INTEGER) n CO_2 channel data points (INTEGERS)
O_2	1	O_2 channel maximum (INTEGER) O_2 channel minimum (INTEGER) n O_2 channel data points (INTEGERS)
Flow	1	Flow channel maximum (INTEGER) Flow channel minimum (INTEGER) n flow channel data points (INTEGERS)
Temperature	1	Temperature channel maximum (INTEGER) Temperature channel minimum (INTEGER) n temperature channel data points (INTEGERS)

ANALYSIS utilizes water vapor pressure for conversion of gas volumes to various conditions (i.e. to STPD or BTPS conditions). The serial binary data file "VAP" contains a water vapor pressure table for temperatures from 20.0 deg C to 44.9 deg C in 0.1 deg C increments. Following is the organization of "VAP".

File	Record #	Contents
VAP	1	Vapor pressure at 20 deg C (REAL)
		Vapor pressure at 20.1 deg C (REAL)
		Vapor pressure at 20.2 deg C (REAL)
		.
		.
		.
		Vapor pressure at 44.9 deg C (REAL)

Variable List

A	INTEGER variable used as a pointer into the flow and temperature signal arrays (Line3 and Line4) during signal integration.
A$[3]	STRING constant set equal to the string "Air" which is used in the hard copy output display table heading.
Adiff	REAL variable representing the absolute value of the difference between Asum and Bsum. By minimizing Adiff, the breath-by-breath time delay can be determined.
Aire	REAL variable equal to the amount of air expired for the current breath in liters.
Airi	REAL variable equal to the amount of air inspired for the current breath in liters.
Asum	REAL variable containing the area above the CO_2 signal based upon the integration limits Beg_pt and Beg_intg as defined in the breath-by-breath time delay subroutine.

Avco2prod	REAL value equal to the average CO_2 produced on a per breath basis.
Avg_time_delay	REAL value equal to the average of valid time delays as determined by the breath-by-breath time delay subroutine.
Avo2cons	REAL value equal to the average O_2 consumed on a per breath basis.
Avole	REAL variable containing the average expiratory volume for the entire trial in liters.
Avoli	REAL variable containing the average inspiratory volume for the entire trial in liters.
B	INTEGER variable equal to Bin_zero_flow. Used simply to reduce length of calculation involving the binary zero flow value.
B$[2]	STRING constant set equal the string "O2" which is used in the hard copy output display table heading.
Beg_pt	INTEGER array pointer for the CO_2 signal which points to where integration above the CO_2 signal begins.

Best_index	INTEGER array pointer which points to that location in the CO_2 and O_2 signal arrays corresponding to the beginning of inspiration. By also knowing where inspiration begins, the breath-by-breath time delay can be determined.
Best_match	REAL variable containing the smallest difference in the area above and area below the fractional CO_2 signal. Used by the breath-by-breath time delay subroutine.
Bin_zero_flow	INTEGER value equal to the average binary value read from the flow channel for zero flow.
Body_temp	REAL variable representing the body temperature of the subject in deg C.
Bsum	REAL variable containing the area below the CO_2 signal based on the integration limits Beg_intg and End_pt as defined in the breath-by-breath time delay subroutine.
C	INTEGER variable equal to Co2_dc_offset. Used simply to reduce length of calculation involving the CO_2 offset value.

C$[3]	STRING constant set equal to the string "CO2" which is used in the hard copy output display table heading.
C1$[10]	STRING variable containing the name of the CO_2 signal file name.
Cal$[10]	STRING variable containing the name of the calibration factors file.
Cal_flag	INTEGER flag which equals zero when correction to STPD/BTPS conditions is requested. Cal_ flag equals one otherwise.
Cmax	INTEGER variable representing the maximum acquired BCD value on the CO_2 channel.
Cmin	INTEGER variable representing the minimum acquired BCD value on the CO_2 channel.
Co2_cal	REAL value used to convert binary data collected from channel A of the DAM (CO_2 channel) to fractional concentration values.
Co2_dc_offset	INTEGER value equal to the average binary value read from the CO_2 channel for 0% CO_2.

Co2e — REAL variable containing the amount of CO_2 expired for the current breath in liters.

Co2i — REAL variable containing the amount of CO_2 inspired for the current breath in liters.

Co2e_tidal — REAL variable equal to the average CO_2 expired by the subject on a per breath basis.

Co2i_tidal — REAL variable equal to the average CO_2 inspired by the subject on a per breath basis.

Co2prod — REAL variable equal to the amount of CO_2 produced for the current breath in liters.

Correct$[1] — STRING variable equal to "Y" or "y" when correction to STPD/BTPS conditions are requested.

Ctmax — INTEGER variable representing the maximum data value on the temperature channel in degrees C.

Ctmin — INTEGER variable representing the minimum data value on the temperature channel in degrees C.

D$[8] — STRING constant set equal to the string "Inspired" which is used in the hard copy output display table heading.

Date$[18]	STRING variable containing the creation date of the calibration factors file.
E$[7]	STRING constant set equal to the string "Expired" which is used in the hard copy output display table heading.
End	INTEGER variable containing the point in the DAM data strings at which plotting or analysis is to end.
End_pt	INTEGER array pointer for the CO_2 signal which points to where integration of the CO_2 signal ends. Points to same location as the pointer Min_index.
Exp_count	INTEGER array pointer where expiration first begins within the flow signal. This pointer is used along with the end of expiration point to determine time of expiration for any given breath.
Expr_btps	REAL variable used to scale expiratory total gas volumes to BTPS conditions.

Expr_flow_cal	REAL variable containing the factor necessary to convert expiratory data collected from channel C of the DAM (flow channel) to values having flow units of liters per second.
Expr_stpd	REAL variable used to scale expiratory CO_2 and O_2 gas volumes to STPD conditions.
Exp_time	REAL variable containing the time for the current expiration in seconds.
F$[8]	STRING constant set equal to the string "(liters)" which is used in the hard copy output display table heading.
Final_index	INTEGER pointer into the flow signal indicating where analysis of the respiratory data ended. Final_index is used with Init_index to determine total respiratory time.
Flow_cal	REAL value used in converting DAM flow data into units of liters per second. Flow_cal equals Insp_flow_cal during periods of inspiration and equals Expr_flow_cal during periods of expiration.
Fmax	INTEGER variable representing the maximum acquired BCD value on the flow channel.

Fmin	INTEGER variable representing the minimum acquired BCD value on the flow channel.
G$[4]	STRING constant set equal to the string "BTPS" which is used in the hard copy output display table heading.
Good_exp_count	INTEGER variable containing the number of good expirations analyzed. A good expiration is defined to be an expiration greater than 500 ml.
Good_insp_count	INTEGER variable containing the number of good inspirations analyzed. A good inspiration is an inspiration greater than 500 ml.
H$[4]	STRING constant set equal to the string "STPD" which is used in the hard copy output display table heading.
I	REAL loop counter.
Incr	REAL variable used as the step value in plotter routine FOR/NEXT loops.
Init_index	INTEGER pointer into the flow signal indicating where analysis of the respiratory data begins. Init_index is used with Final_index to determine total respiratory time.

Insp_btps	REAL variable used to scale inspiratory total gas volumes to BTPS conditions.
Insp_count	INTEGER array pointer where inspiration first begins within the flow signal. This pointer is used along with the end of inspiration point to determine time of inspiration for any given breath.
Insp_flow_cal	REAL variable containing factor necessary to convert inspiratory data collected from channel C of the DAM (flow channel) to values having flow units of liters per second.
Insp_stpd	REAL variable used to scale inspiratory CO_2 and O_2 gas volumes to STPD conditions.
Insp_time	REAL variable containing the time of inspiration in seconds.
Line1	24000 point data string containing the BCD values acquired from the CO_2 channel of the DAM.
Line2	24000 point data string containing the BCD values acquired from the O_2 channel of the DAM.

Line3	24000 point data string containing the BCD values acquired from the flow channel of the DAM.
Line4	24000 point data string containing the BCD values acquired from the temperature channel of the DAM.
Max_index	INTEGER pointer into the CO_2 data string where the maximum CO_2 fraction is observed in the current breath. Used to determine the GMS time delay value.
Mid_index	INTEGER pointer into the CO_2 data string where 1/2 of the maximum CO_2 fraction is observed in the current breath. Used to determine the GMS time delay value.
Min_index	INTEGER pointer into the CO_2 data string where the minimum CO_2 fraction is observed in the current breath. Used to determine the GMS time delay value.
Minvole	REAL variable containing the expiratory minute volume of the subject in liters per minute.
Minvoli	REAL variable containing the inspiratory minute volume of the subject in liters per minute.

No_breaths	INTEGER variable representing the number of breaths analyzed during the analysis procedure.
No_points	INTEGER variable containing the total number of data points to be analyzed.
O$[10]	STRING variable containing the name of the O_2 signal file.
O2_cal	REAL variable used to convert binary data collected from channel B of the DAM (O_2 channel) to fractional concentration units.
O2cons	REAL variable equal to the amount of O_2 consumed for the current breath in liters.
O2_dc_offset	INTEGER variable equal to the average binary value read from the O_2 channel for 12.9% O_2.
O2e	REAL variable containing the amount of expired O_2 for the current breath in liters.
O2i	REAL variable containing the amount of inspired O_2 for the current breath in liters.
O2e_tidal	REAL variable equal to the average O_2 expired by the subject on a per breath basis.

O2i_tidal — REAL variable equal to the average O_2 inspired by the subject on a per breath basis.

Offset — INTEGER variable added to the CO_2 and O_2 array pointers for the plotting of time aligned signals. Offset is directly related to the Time_delay parameter.

Ol — REAL variable containing the actual O_2 concentration read from the mass spectrometer for 12.9% O_2.

Omax — INTEGER variable representing the maximum acquired BCD value on the O_2 channel.

Omin — INTEGER variable representing the minimum acquired BCD value on the O_2 channel.

P — INTEGER variable containing the total number of data points to be analyzed or plotted. (Same as No_points.)

Pb — REAL variable containing the barometric pressure in torr.

Ph2o_body — REAL variable representing the vapor pressure of water at body temperature in torr.

Ph2o_expr	REAL variable representing the vapor pressure of water at the expiratory temperature in torr.
Ph2o_insp	REAL variable representing the vapor pressure of water at the inspiratory temperature in torr.
Q$[1]	STRING variable containing the answer to a question asked by ANALYSIS. Typically this answer is either a "Y" or "N".
R	REAL variable equal to the respiratory quotient. See equations sections for more details.
Rel_humid	REAL variable containing the relative humidity in fractional form.
Respf	REAL variable equal to the respiratory frequency of the subject in breaths per minute.
S	INTEGER variable representing the DAM sampling frequency in Hz.
Start	INTEGER variable containing the point in the DAM data strings at which plotting or analysis is to begin.
T	REAL variable equal to the reciprocal of the sampling frequency (S).

T$[10]	STRING variable containing the flow temperature file name.
Ta	REAL variable containing the 2nd order temperature coefficient for converting DAM temperature data to units of degrees C.
Tb	REAL variable containing the 1st order temperature coefficient for converting DAM temperature data to units of degrees C.
Tc	REAL variable containing the constant temperature coefficient for converting DAM temperature data to units of degrees C.
Temp	REAL variable equal to the incremental respiratory temperature at the present analysis point in degrees C.
Temp_a	INTEGER variable used by the breath-by-breath time delay routine to preserve "A", the flow and temperature signal pointer.
Temp_z	INTEGER variable used by the breath-by-breath time delay routine to preserve "Z", the fractional CO_2 and O_2 signal pointer.

Time_delay	INTEGER variable representing the gas mass spectrometer time delay in msec.
Time_delay_cnt	INTEGER variable containing the number of valid time delays computed by the breath-by-breath time delay subroutine. This count is used to calculate an average time delay of the valid time delays.
Time_delay_flag	INTEGER flag which equals zero when fixed time delays are requested, equal to 1 when variable time delays are requested, and equal to 2 when variable time delays are requested but the computed time delay was invalid.
Time_delay_sum	REAL variable containing the sum of all valid time delays computed by the breath-by-breath time delay subroutine. This sum is used to calculate an average time delay of the valid time delays.
Tmax	INTEGER variable representing the maximum acquired BCD value on the temperature channel.
Tmin	INTEGER variable representing the minimum acquired BCD value on the temperature channel.

Tot_co2_exp — REAL variable containing the total expired CO_2 volume during the experiment in liters.

Tot_co2_insp — REAL variable equal to the total inspired CO_2 volume during the experiment in liters.

Tot_co2_prod — REAL variable containing the total CO_2 volume produced during the experiment in liters.

Tot_exp_points — INTEGER variable representing the number of flow signal points considered to be expiratory points. This number is used along with the sampling period to determine the total time of expiration.

Tot_insp_points — INTEGER variable representing the number of flow signal points considered to be inspiratory points. This number is used along with the sampling period to determine the total time of inspiration.

Tot_o2_cons — REAL variable equal to the total O_2 volume consumed during the experiment in liters.

Tot_o2_exp — REAL variable containing the total expired O_2 volume during the experiment in liters.

Tot_o2_insp	REAL variable equal to the total inspired O_2 volume during the experiment in liters.
Tot_time_exp	REAL variable containing the total time during which expiration occurred in seconds.
Tot_time_insp	REAL variable containing the total time during which inspiration occurred in seconds.
Tot_time_resp	REAL variable containing the total time during which respiration occurred in seconds.
Tot_vol_exp	REAL variable containing the total expired gas during the trial in liters.
Tot_vol_insp	REAL variable containing the total inspired gas during the trial in liters.
V$[10]	STRING variable containing the flow signal file name.
Vap	REAL array containing the water vapor pressure values from 20.0 to 44.9 deg C in 0.1 deg C increments.
V_dot_co2	REAL variable equal to the average rate at which CO_2 is produced in liters per minute.

V_dot_o2	REAL variable equal to the average rate at which O_2 is consumed in liters per minute.
Wye	INTEGER FOR/NEXT loop counter used by the breath-by-breath time delay subroutine.
Z	INTEGER variable used as a pointer into the CO_2 signal array (Line1) and O_2 signal array (Line2) during signal integration.

```
10   !************************************************************************
20   !
30   !        BREATH-BY-BREATH RESPIRATORY ANALYSIS/PLOTTING ROUTINE
40   !
50   !        HP BASIC FILENAME:  ANALYSIS
60   !
70   !        DEPARTMENT OF ELECTRICAL 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  !                THIS ROUTINE PERFORMS ALL ANALYSIS THAT IS CURRENTLY
180  !                DONE ON THE RESPIRATORY DATA.  RESULTS OF THIS ANALYSIS
190  !                ARE PRESENTED IN BOTH TABULAR AND GRAPHICAL FORMS.
200  !
210  !        ROUTINE(S) CALLED
220  !
230  !                AUTOST - USER PROGRAM ACCESS ROUTINE
240  !
250  !************************************************************************
260  !
270  !        NOTE 1:  THIS ROUTINE ASSUMES THAT THE BINARY CALIBRATION FILES
280  !                 CREATED BY "CAPCRUNCH" ARE STORED ON THE HP9895A 8"
290  !                 FLEXIBLE DISK ":HP9895,700,0" (VOLUME #7 IN THE PASCAL
300  !                 OPERATING SYSTEM).
310  !
320  !        NOTE 2:  ANALYSIS ASSUMES THAT THE FOUR BINARY DATA FILES CREATED
330  !                 BY "DAPCRUNCH" ARE ALSO STORED ON THE HP9895A 8" FLEXIBLE
340  !                 DISK ":HP9895,700,0" (VOLUME #7 IN THE PASCAL OPERATING
350  !                 SYSTEM).
360  !
370  !        NOTE 3:  GAS MASS SPECTROMETER TIME DELAY VALUES CAN BE DETERMINED
380  !                 ON A BREATH-BY-BREATH BASIS OR A FIXED TIME DELAY CAN BE
390  !                 SELECTED FOR ANALYSIS OF THE ACQUIRED DATA.  SHOULD AN
400  !                 EXTREME BREATH-BY-BREATH TIME BE CALCULATED, AN AVERAGE
410  !                 TIME DELAY IS SUBSTITUTED FOR THE COMPUTED TIME DELAY.
420  !
430  !        NOTE 4:  RESPIRATORY VOLUMES CAN BE CORRECTED TO STPD/BTPS CON-
440  !                 DITIONS PROVIDED THE BAROMETRIC PRESSURE (TORR), RELATIVE
450  !                 HUMIDITY (%), AND BODY TEMPERATURE (DEG C) IS SUPPLIED.
460  !                 USING THIS INFORMATION ALONG WITH POINT-BY-POINT TEMP-
470  !                 ERATURE CORRECTION (USING CHANNEL D, THE RESPIRATORY
480  !                 TEMPERATURE CHANNEL), THE INSPIRATORY AND EXPIRATORY
490  !                 GAS VOLUMES ARE CORRECTED.
500  !
510  !        NOTE 5:  ANY WINDOW OF DATA FROM THE DAM DATA MAY BE PLOTTED
520  !                 EITHER ON THE HP9826 CRT OR HP9872C PLOTTER.  ALL FOUR
530  !                 CHANNELS OF DATA ARE PLOTTED, THE CO2 AND O2 CHANNELS
540  !                 BEING PLOTTED WITH A TIME DELAY EQUAL TO THE AVERAGE
550  !                 TIME DELAY ENTERED BY THE USER.  DATA PLOTTED ON THE
```

```
560  !                  HP9826 CRT MAY ALSO BE DUMPED TO THE HP2673A THERMAL
570  !                  PRINTER.
580  !
590  !         NOTE 6:  ONCE BREATH-BY-BREATH ANALYSIS OF THE DAM BEGINS,
600  !                  INFORMATION CONCERNING EACH BREATH IS PRINTED ON THE
610  !                  DECwriter PRINTER.
620  !
630  !         NOTE 7:  ONCE THE DATA ARRAYS ARE EXHAUSTED SUMMARY DATA FOR THE
640  !                  ENTIRE RUN IS COMPUTED AND PRINTED.  FILE NAMES AS WELL
650  !                  AS CRITICAL CALIBRATION PARAMETERS ARE ALSO PRINTED.
660  !
670  !************************************************************************
680  !
690  !*** SPECIAL FUNCTION KEY DECLARATION
700  !
710  OPTION BASE 1
720  ON KEY 2 LABEL "ANALYSIS" GOTO 780
730  ON KEY 9 LABEL "  EXIT" GOTO Done
740  GOTO 720
750  !
760  !*** CALL ANALYSIS SUBROUTINE Andata
770  !
780  OFF KEY
790  GOSUB Andata
800  !
810  !*** SIGNAL END OF ROUTINE AND RETURN TO AUTOST
820  !
830  PRINTER IS 1
840  DISP "PROGRAM RUN COMPLETE"
850 Done:  OFF KEY
860        MASS STORAGE IS ":INTERNAL"
870        LOAD "AUTOST",1
880        STOP
890 !
900 !*** BEGINNING OF ANALYSIS SUBROUTINE
910 !
920 !*** GET NUMBER OF POINTS TO ANALYZE AND MAKE APPROPRIATE DIMENSIONS
930 !
940 Andata: BEEP
950   INPUT "ENTER THE TOTAL NUMBER OF POINTS TO BE ANALYZED",P
960   No_points=P
970   ALLOCATE INTEGER Line1(P),Line2(P),Line3(P),Line4(P)
980   INTEGER Cmax,Cmin,Omax,Omin,Fmax,Fmin,Tmax,Tmin
990   INTEGER Co2_dc_offset,02_dc_offset,Bin_zero_flow,S
1000  !
1010  !*** GET SUBJECT INFORMATION FOR PRINTOUTS
1020  !
1030  BEEP
1040  INPUT "ENTER THE SUBJECT´S NAME OR IDENTIFIER",Name$
1050  Q$=""
1060  !
1070  !*** LOAD BINARY CALIBRATION FILE IF REQUESTED
1080  !
1090  BEEP
1100  INPUT "LOAD CALIBRATION FACTORS FROM DISK ? (Y/N)",Q$
```

```
1110  IF Q$="Y" OR Q$="y" THEN GOSUB Rtcal
1120  !
1130  !*** LOAD FOUR CHANNELS OF BINARY DATA
1140  !
1150  GOSUB Rtdata
1160  !
1170  !*** CHOOSE APPROPRIATE SAMPLING FREQUENCY
1180  !
1190 Analyze: S=50
1200  Q$=""
1210  BEEP
1220  INPUT "CHANGE SAMPLING FREQUENCY FROM 50 HZ.? (Y/N)",Q$
1230  IF Q$<>"Y" AND Q$<>"y" THEN GOTO 1290
1240  INPUT "ENTER DESIRED SAMPLING FREQUENCY (HZ.)",S
1250  Q$=""
1260  !
1270  !*** CHOOSE B-BY-B TIME DELAYS OR A FIXED GMS TIME DELAY
1280  !
1290  BEEP
1300  INPUT "B-BY-B TIME DELAY OR FIXED TIME DELAY (B/F) ?",Q$
1310  IF Q$<>"B" THEN 1430
1320  !
1330  !*** FOR B-BY-B DELAYS, CHOOSE STARTING VALUE FOR AVERAGE DELAY
1340  !
1350  BEEP
1360  INPUT "AVERAGE TIME DELAY FOR BAD BREATH PROBLEMS (msec)?",Time_delay_sum
1370  Avg_time_delay=Time_delay_sum
1380  Time_delay_flag=1      !Time_delay_flag=1 FOR B-BY-B TIME DELAYS
1390  GOTO 1530
1400  !
1410  !*** FOR FIXED TIME DELAY, ENTER DESIRED TIME DELAY
1420  !
1430  Time_delay_flag=0      !Time_delay_flag=1 FOR FIXED TIME DELAYS
1440  PRINT "CURRENT TIME DELAY IS ";Time_delay;" msec"
1450  Q$=""
1460  BEEP
1470  INPUT "CHANGE TIME DELAY? (Y/N)",Q$
1480  IF Q$<>"Y" AND Q$<>"y" THEN GOTO 1530
1490  INPUT "ENTER DESIRED TIME DELAY (msec.)",Time_delay
1500  !
1510  !*** SEE IF STPD/BTPS CONVERSION IS DESIRED
1520  !
1530  Correct$=""
1540  BEEP
1550  Cal_flag=0
1560  INPUT "CORRECT SIGNALS TO STPD AND BTPS? (Y/N)",Correct$
1570  IF Correct$<>"Y" AND Correct$<>"y" THEN Cal_flag=1
1580  IF Correct$<>"Y" AND Correct$<>"y" THEN 1840
1590  !
1600  !*** IF STPD/BTPS DESIRED, LOAD IN WATER VAPOR PRESSURE TABLE
1610  !
1620  DISP "READING WATER VAPOR PRESSURES"
1630  DIM Vap(250)
1640  ASSIGN @Path1 TO "VAP"
1650  ON END @Path1 GOTO 1710
```

```
1660  ENTER @Path1;Vap(*)
1670  !
1680  !*** FOR STPD/BTPS, ENTER BAROMETRIC PRESSURE, RELATIVE HUMIDITY,
1690  !*** AND BODY TEMPERATURE
1700  !
1710  DISP ""
1720  BEEP
1730  INPUT "ENTER THE BAROMETRIC PRESSURE (torr)",Pb
1740  BEEP
1750  INPUT "ENTER THE RELATIVE HUMIDITY (in %)",Rel_humid
1760  Rel_humid=Rel_humid/100
1770  BEEP
1780  INPUT "ENTER THE BODY TEMPERATURE (deg C)",Body_temp
1790  Ph2o_body=Vap((Body_temp-20)*10)     !Vap CONTAINS WATER VAPOR
1800  !                                     PRESSURES FROM 20.0 C TO 44.9 C
1810  !
1820  !*** SEE IF PLOT OF DATA IS DESIRED
1830  !
1840 Q$=""
1850 BEEP
1860 INPUT "WOULD YOU LIKE A PLOT OF THE DATA?(Y/N)",Q$
1870 !
1880 !*** IF SO, GO PLOT THE DATA
1890 !
1900 IF Q$="Y" OR Q$="y" THEN GOSUB Dtplot
1910  Q$=""
1920!
1930!*** INITIALIZE NECESSARY ANALYSIS PARAMETERS
1940!
1950 Vol_compare: IF 01=0 THEN 01=.11
1960  No_breaths=0
1970  Good_insp_count=0
1980  Good_exp_count=0
1990  Time_delay_cnt=1
2000  T=1/S
2010  BEEP
2020  INPUT "ENTER STARTING POINT TO ANALYZE.",Start
2030  IF Start>No_points-50 THEN GOTO 2010
2040  A=Start
2050  Z=Start
2060  BEEP
2070  INPUT "ENTER ENDING POINT TO ANALYZE.",End
2080  IF End>No_points THEN GOTO 2060
2090  Tot_vol_insp=0
2100  Tot_vol_exp=0
2110  Tot_o2_insp=0
2120  Tot_co2_insp=0
2130  Tot_o2_exp=0
2140  Tot_co2_exp=0
2150  Tot_o2_cons=0
2160  Tot_co2_prod=0
2170  Tot_insp_points=0
2180  Tot_exp_points=0
2190 !
2200 !*** LOCATE FIRST INSPIRATION IN FLOW SIGNAL
```

```
2210 !
2220 First_inspire: Flow_cal=Insp_flow_cal
2230  B=Bin_zero_flow
2240 !
2250 !*** CHECK FOR FIRST POSITIVE TO NEGATIVE TRANSITION OF FLOW SIGNAL
2260 !
2270 IF (Line3(A)-B<0) OR (Line3(A+1)-B>=0) OR (Line3(A+2)-B>=0) THEN 2290
2280 IF (Line3(A+3)-B<0) AND (Line3(A+4)-B<0) THEN Start
2290  A=A+1
2300  GOTO First_inspire
2310 !
2320 !*** ONCE FIRST INSPIRATION FOUND, ADJUST ADDITIONAL ANALYSIS VARIABLES
2330 !
2340 Start: IF Line3(A)-Bin_zero_flow<>0 THEN A=A+1
2350  IF Cal_flag=1 THEN X=1
2360  IF Cal_flag<>1 THEN 2410
2370  Insp_btps=1
2380  Expr_btps=1
2390  Insp_stpd=1
2400  Expr_stpd=1
2410 Headings: A$="Air"
2420           B$="02"
2430           C$="CO2"
2440           D$="Inspired"
2450           E$="Expired"
2460           F$="(liters)"
2470           G$="BTPS"
2480           H$="STPD"
2490 !
2500 !*** GO PRINT DATA TABLE HEADER ON PRINTER LISTING
2510 !
2520 GOSUB Hard_copy_head
2530 !
2540 !*** LOCATE NEXT INSPIRATION IN FLOW DATA
2550 !
2560 New_inspire:  ! Check first for glitches
2570 E:Insp_count=A
2580  !
2590  !*** IF B-BY-B TIME DELAY, GO TO SUBROUTINE TO DETERMINE THE DELAY
2600  !
2610  IF Time_delay_flag=0 THEN 2800
2620  GOSUB Bbb_time_delay
2630  !
2640  !*** IF B-BY-B TIME DELAY OUTSIDE LIMITS, SUBSTITUTE AVERAGE DELAY
2650  !
2660  IF (Time_delay>560) OR (Time_delay<330) THEN Bad_time_delay
2670  !
2680  !*** OTHERWISE, USE THE B-BY-B DELAY AND UPDATE RUNNING AVERAGE
2690  !
2700  Time_delay_flag=1
2710  Time_delay_sum=Time_delay_sum+Time_delay
2720  Time_delay_cnt=Time_delay_cnt+1
2730  Avg_time_delay=Time_delay_sum/Time_delay_cnt
2740  GOTO 2800
2750 Bad_time_delay:  Time_delay=Avg_time_delay
```

```
2760  Time_delay_flag=2
2770  !
2780  !*** ADJUST CO2 AND O2 INDEX (Z) FOR PROPER POINT SELECTION
2790  !
2800  Z=A+INT(Time_delay/1000*S+.5)
2810 Flow_cal=Insp_flow_cal
2820  !
2830  !*** PREPARE FOR STPD/BTPS CONVERSION IF REQUESTED
2840  !
2850  IF Cal_flag=1 THEN 2930
2860  Temp=Ta*Line4(A)^2+Tb*Line4(A)+Tc
2870  Ph2o_insp=Vap((Temp-20)*10)
2880  Insp_btps=(Pb-Rel_humid*Ph2o_insp)/(Pb-Ph2o_body)*(273+Body_temp)/(273+Tem
p)
2890  Insp_stpd=(Pb-Rel_humid*Ph2o_insp)/760*273/(273+Temp)
2900  !
2910  !*** COMPUTE HALF THE AREA FOR THE FIRST TRAPEZOIDAL AREA
2920  !
2930  Airi=.5*(Line3(A)-B)*Flow_cal*Insp_btps
2940  Co2i=.5*(Line3(A)-B)*(Line1(Z)-Co2_dc_offset)*Co2_cal*Insp_stpd*Flow_cal
2950  O2i=.5*(Line3(A)-B)*Flow_cal*((Line2(Z)-O2_dc_offset)*O2_cal+O1)*Insp_stpd
2960  !
2970  !*** BUMP ARRAY POINTERS TO NEXT TRAPEZOIDAL AREA
2980  !
2990 A_label:  A=A+1
3000  Z=Z+1
3010  !
3020  !*** MAKE SURE ENOUGH DATA POINTS REMAIN FOR THIS BREATH
3030  !
3040  IF Z>End-50 THEN Goon
3050  !
3060  !*** BRANCH IF END OF INSPIRATION
3070  !
3080  IF (Line3(A)-B<0) OR (Line3(A+1)-B<=0) OR (Line3(A+2)<=0) THEN 3130
3090  IF (Line3(A+3)-B>0) AND (Line3(A+4)-B>0) AND (Line3(A+5)-B>0) THEN Decri
3100  !
3110  !*** PREPARE FOR STPD/BTPS CONVERSION IF REQUESTED
3120  !
3130  IF Cal_flag=1 THEN 3210
3140  Temp=Ta*Line4(A)^2+Tb*Line4(A)+Tc
3150  Ph2o_insp=Vap((Temp-20)*10)
3160  Insp_btps=(Pb-Rel_humid*Ph2o_insp)/(Pb-Ph2o_body)*(273+Body_temp)/(273+Tem
p)
3170  Insp_stpd=(Pb-Rel_humid*Ph2o_insp)/760*273/(273+Temp)
3180  !
3190  !*** SUM UP INSPIRATORY VOLUME, INSPIRED CO2 AND O2 FOR THIS BREATH
3200  !
3210  Airi=Airi+(Line3(A)-B)*Flow_cal*Insp_btps
3220  Co2i=Co2i+(Line1(Z)-Co2_dc_offset)*(Line3(A)-B)*Flow_cal*Co2_cal*Insp_stpd
3230  O2i=O2i+((Line2(Z)-O2_dc_offset)*O2_cal+O1)*Insp_stpd*(Line3(A)-B)*Flow_ca
l
3240  !
3250  !*** LOOP UNTIL END OF INSPIRATION
3260  !
3270  GOTO A_label
```

```
3280  !
3290  !*** SET ARRAY POINTERS BACK ONE TO REFLECT END OF INSPIRATION
3300  !
3310 Decri:  A=A-1
3320  Z=Z-1
3330  !
3340  !*** PREPARE FOR STPD/BTPS CONVERSION IF REQUESTED
3350  !
3360  IF Cal_flag=1 THEN 3440
3370  Temp=Ta*Line4(A)^2+Tb*Line4(A)+Tc
3380  Ph2o_insp=Vap((Temp-20)*10)
3390  Insp_btps=(Pb-Rel_humid*Ph2o_insp)/(Pb-Ph2o_body)*(273+Body_temp)/(273+Tem
p)
3400  Insp_stpd=(Pb-Rel_humid*Ph2o_insp)/760*273/(273+Temp)
3410  !
3420  !*** SUBTRACT OFF 1/2 OF THE LAST TRAPEZOIDAL AREA
3430  !
3440  Airi=Airi-.5*(Line3(A)-B)*Flow_cal*Insp_btps
3450  Co2i=Co2i-.5*(Line3(A)-B)*(Line1(Z)-Co2_dc_offset)*Flow_cal*Co2_cal*Insp_s
tpd
3460  O2i=O2i-.5*(Line3(A)-B)*Flow_cal*((Line2(Z)-O2_dc_offset)*O2_cal+O1)*Insp_
stpd
3470  !
3480  !*** ADJUST ARRAY POINTER TO POINT TO START OF EXPIRATION
3490  !
3500  A=A+1
3510  Z=Z+1
3520 !
3530 !*** COMPUTE TIME OF INSPIRATION
3540 !
3550 B:Insp_time=(A-Insp_count)*T     !Time of inspiration in seconds
3560  !
3570  !*** BEGIN COMPUTATIONS ON EXPIRATION
3580  !
3590 New_expire:Flow_cal=Expr_flow_cal
3600  Exp_count=A
3610  !
3620  !*** PREPARE FOR STPD/BTPS CONVERSION IF REQUESTED
3630  !
3640  IF Cal_flag=1 THEN 3720
3650  Temp=Ta*Line4(A)^2+Tb*Line4(A)+Tc
3660  Ph2o_expr=Vap((Temp-20)*10)
3670  Expr_btps=(Pb-Ph2o_expr)/(Pb-Ph2o_body)*(273+Body_temp)/(273+Temp)
3680  Expr_stpd=(Pb-Ph2o_expr)/760*273/(273+Temp)
3690  !
3700  !*** TAKE ONLY 1/2 OF THE FIRST TRAPEZOIDAL AREA (EXPIRATION)
3710  !
3720  Aire=.5*(Line3(A)-B)*Flow_cal*Expr_btps
3730  Co2e=.5*(Line3(A)-B)*(Line1(Z)-Co2_dc_offset)*Co2_cal*Expr_stpd*Flow_cal
3740  O2e=.5*(Line3(A)-B)*Flow_cal*((Line2(Z)-O2_dc_offset)*O2_cal+O1)*Expr_stpd
3750  !
3760  !*** ADJUST ARRAY POINTERS FOR NEXT TRAPEZOIDAL AREA
3770  !
3780 F: A=A+1
3790  Z=Z+1
```

```
3800  !
3810  !*** MAKE SURE ADEQUATE POINTS EXIST FOR EXPIRATION CALCULATIONS
3820  !
3830  IF Z>End-50 THEN Goon
3840  !
3850  !*** BRANCH IF END OF EXPIRATION
3860  !
3870  IF (Line3(A)-B>0) OR (Line3(A+1)-B>=0) OR (Line3(A+2)-B>=0) THEN 3920
3880  IF (Line3(A+3)-B<0) AND (Line3(A+4)-B<0) AND (Line3(A+5)-B<0) THEN Decre
3890  !
3900  !*** PREPARE FOR STPD/BTPS CONVERSION IF REQUESTED
3910  !
3920  IF Cal_flag=1 THEN 4000
3930  Temp=Ta*Line4(A)^2+Tb*Line4(A)+Tc
3940  Ph2o_expr=Vap((Temp-20)*10+1)
3950  Expr_btps=(Pb-Ph2o_expr)/(Pb-Ph2o_body)*(273+Body_temp)/(273+Temp)
3960  Expr_stpd=(Pb-Ph2o_expr)/760*273/(273+Temp)
3970  !
3980  !*** SUM UP EXPIRATORY VOLUME, EXPIRED CO2 AND O2 FOR THIS BREATH
3990  !
4000  Aire=Aire+(Line3(A)-B)*Flow_cal*Expr_btps
4010  Co2e=Co2e+(Line3(A)-B)*(Line1(Z)-Co2_dc_offset)*Flow_cal*Co2_cal*Expr_stpd
4020  O2e=O2e+(Line3(A)-B)*Flow_cal*((Line2(Z)-O2_dc_offset)*O2_cal+O1)*Expr_stp
d
4030  !
4040  !*** LOOP UNTIL END OF EXPIRATION
4050  !
4060  GOTO F
4070  !
4080  !*** SET ARRAY POINTERS BACK ONE TO REFLECT END OF EXPIRATION
4090  !
4100 Decre: A=A-1
4110  Z=Z-1
4120  !
4130  !*** PREPARE FOR STPD/BTPS CONVERSION IF REQUESTED
4140  !
4150  IF Cal_flag=1 THEN 4230
4160  Temp=Ta*Line4(A)^2+Tb*Line4(A)+Tc
4170  Ph2o_expr=Vap((Temp-20)*10)
4180  Expr_btps=(Pb-Ph2o_expr)/(Pb-Ph2o_body)*(273+Body_temp)/(273+Temp)
4190  Expr_stpd=(Pb-Ph2o_expr)/760*273/(273+Temp)
4200  !
4210  !*** SUBTRACT OFF 1/2 OF THE LAST TRAPEZOIDAL AREA (EXPIRATION)
4220  !
4230  Aire=Aire-.5*(Line3(A)-B)*Flow_cal*Expr_btps
4240  Co2e=Co2e-.5*(Line3(A)-B)*(Line1(Z)-Co2_dc_offset)*Flow_cal*Co2_cal*Expr_s
tpd
4250  O2e=O2e-.5*(Line3(A)-B)*Flow_cal*((Line2(Z)-O2_dc_offset)*O2_cal+O1)*Expr_
stpd
4260  !
4270  !*** BUMP ARRAY POINTERS TO START OF NEXT INSPIRATION
4280  !
4290  A=A+1
4300  Z=Z+1
4310  !
```

```
4320  !*** BEGIN CALCULATIONS FOR THIS PARTICULAR BREATH
4330  !
4340 C: Airi=Airi*T     !INSPIRATORY VOLUME FOR THIS BREATH (LITERS)
4350  Co2i=Co2i*T     !CO2 INSPIRED FOR THIS BREATH (LITERS)
4360  O2i=O2i*T     !O2 INSPIRED FOR THIS BREATH (LITERS)
4370  Aire=Aire*T     !EXPIRATORY VOLUME FOR THIS BREATH (LITERS)
4380  Co2e=Co2e*T     !CO2 EXPIRED FOR THIS BREATH (LITERS)
4390  O2e=O2e*T     !O2 EXPIRED FOR THIS BREATH (LITERS)
4400  O2cons=O2i+O2e     !CONSUMED O2 FOR THIS BREATH (LITERS)
4410  Co2prod=Co2i+Co2e     !CO2 PRODUCED FOR THIS BREATH (LITERS)
4420  No_breaths=No_breaths+1     !TOTAL NUMBER OF BREATHS ANALYZED
4430  IF No_breaths=1 THEN Init_index=Insp_count
4440  Exp_time=(A-Exp_count)*T     !TIME FOR CURRENT EXPIRATION (SECONDS)
4450  Final_index=A-1
4460  !
4470  !*** GO PRINT CALCULATED VALUES FOR THIS BREATH
4480  !
4490  GOSUB Hard_output
4500  !
4510  !*** KEEP TRACK OF GOOD INSPIRATIONS AND EXPIRATIONS
4520  !
4530  IF ABS(Airi)>.50 THEN Good_insp_count=Good_insp_count+1
4540  IF ABS(Aire)>.50 THEN Good_exp_count=Good_exp_count+1
4550  !
4560  !*** ADJUST RUNNING TOTAL VALUES FOR ENTIRE TRIAL
4570  !
4580  Tot_vol_insp=Tot_vol_insp+Airi     !TOTAL INSPIRED GAS DURING EXPERIMENT (
LITERS)
4590  Tot_vol_exp=Tot_vol_exp+Aire     !TOTAL EXPIRED GAS DURING EXPERIMENT (LIT
ERS)
4600  Tot_o2_insp=Tot_o2_insp+O2i     !TOTAL INSPIRED O2 DURING EXPERIMENT (LITE
RS)
4610  Tot_o2_exp=Tot_o2_exp+O2e     !TOTAL EXPIRED O2 DURING EXPERIMENT (LITERS)
4620  Tot_co2_insp=Tot_co2_insp+Co2i     !TOTAL INSPIRED CO2 DURING EXPERIMENT (
LITERS)
4630  Tot_co2_exp=Tot_co2_exp+Co2e     !TOTAL EXPIRED CO2 DURING EXPERIMENT (LIT
ERS)
4640  Tot_o2_cons=Tot_o2_cons+O2cons     !TOTAL CONSUMED O2 DURING EXPERIMENT (L
ITERS)
4650  Tot_co2_prod=Tot_co2_prod+Co2prod     !TOTAL PRODUCED CO2 DURING EXPERIMEN
T (LITERS)
4660  Tot_insp_points=Tot_insp_points+Insp_time/T     !# OF INSPIRATORY VOLUME D
ATA POINTS USED
4670  Tot_exp_points=Tot_exp_points+(A-Exp_count)     !# OF EXPIRATORY VOLUME DA
TA POINTS USED
4680  !
4690  !*** GO FIND ANOTHER BREATH TO PROCESS
4700  !
4710  GOTO New_inspire
4720  !
4730  !*** DATA ARRAYS ARE EXHAUSTED, COMPUTE FINAL TRIAL AVERAGES
4740  !
4750 Goon:Tot_time_insp=Tot_insp_points*T     !TOTAL INSPIRATORY TIME (SECONDS)
4760  Tot_time_exp=Tot_exp_points*T     !TOTAL EXPIRATORY TIME (SECONDS)
4770  Tot_time_resp=(Final_index-Init_index)*T     !TOTAL RESPIRATORY TIME (SECO
```

```
NDS)
4780  Minvoli=Tot_vol_insp*60/Tot_time_resp     !INSPIRATORY MINUTE VOLUME (VI-D
OT, LITERS/MIN)
4790  Minvole=Tot_vol_exp*60/Tot_time_resp     !EXPIRATORY MINUTE VOLUME (VE-DOT
, LITERS/MIN)
4800  No_breaths=(Good_insp_count+Good_exp_count)/2
4810  Avoli=Tot_vol_insp/No_breaths     !AVERAGE INSPIRATORY VOLUME (LITERS)
4820  Avole=Tot_vol_exp/No_breaths     !AVERAGE EXPIRATORY VOLUME (LITERS)
4830  Respf=No_breaths*60/Tot_time_resp   !RESPIRATORY FREQUENCY (BREATHS/MIN)
4840  O2i_tidal=Tot_o2_insp/No_breaths    !AVERAGE INSPIRED O2/BREATH (LITERS)
4850  O2e_tidal=Tot_o2_exp/No_breaths    !AVERAGE EXPIRED O2/BREATH (LITERS)
4860  Co2i_tidal=Tot_co2_insp/No_breaths     !AVERAGE INSPIRED CO2/BREATH (LITER
S)
4870  Co2e_tidal=Tot_co2_exp/No_breaths     !AVERAGE EXPIRED CO2/BREATH (LITERS)
4880  Avo2cons=Tot_o2_cons/No_breaths    !AVERAGE O2 CONSUMED/BREATH (LITERS)
4890  Avco2prod=Tot_co2_prod/No_breaths
4900  V_dot_o2=Tot_o2_cons/Tot_time_resp*60
4910  V_dot_co2=Tot_co2_prod/Tot_time_resp*60
4920  R=ABS(V_dot_co2/V_dot_o2)
4930  !
4940  !*** GO PRINT MEAN VALUES
4950  !
4960  GOSUB Means
4970  Q$=""
4980  BEEP
4990  INPUT "REDO ANALYSIS? (Y/N)",Q$
5000  IF Q$="Y" OR Q$="y" THEN GOTO Analyze
5010  !
5020  !*** END OF ANALYSIS SUBROUTINE
5030  !
5040  RETURN ! Branch back to the main routine
5050  !
5060  !*** SUBROUTINE TO PRINT STANDARD HEADER TO THE DECwriter II PRINTER
5070  !
5080 Hard_copy_head:  PRINTER IS 9
5090  PRINT "";"SUBJECT IDENTIFIER:  ";Name$
5100  PRINT "";"DATE:  ";Date$;""
5110  PRINT USING "#,2X,K,5X,K,7X,K,8X,K,8X,K,7X,K";"Breath";A$;A$;B$;B$;C$
5120  PRINT USING "#,7X,K,8X,K,7X,K,7X,K,6X,K,5X,K";C$;B$;C$;"Insp";"Expr";"Dela
y"
5130  PRINT
5140  PRINT USING "#,2X,K,4X,K,2X,K,3X,K,2X,K";"Start";D$;E$;D$;E$
5150  PRINT USING "#,3X,K,2X,K,3X,K,2X,K";D$;E$;"Consumed";"Produced"
5160  PRINT USING "#,4X,K,6X,K,6X,K";"TIME";"TIME";"TIME"
5170  PRINT
5180  PRINT USING "#,2X,K,4X,K,2X,K,2X,K,2X,K,2X,K";"Index";F$;F$;F$;F$;F$
5190  PRINT USING "#,2X,K,2X,K,2X,K,3X,K,5X,K,5X,K";F$;F$;F$;"(sec)";"(sec)";"(m
sec)"
5200  PRINT
5210  IF Correct$<>"Y" THEN 5250
5220  PRINT USING "#,13X,K,6X,K,6X,K,6X,K,6X,K";G$;G$;H$;H$;H$
5230  PRINT USING "#,6X,K,6X,K,6X,K";H$;H$;H$
5240  PRINT
5250  PRINT USING "#,K";"-----------------------------------------------------"
5260  PRINT USING "#,K";"-----------------------------------------------------"
```

```
5270  PRINT USING "#,K";"--------------------"
5280  PRINT
5290  RETURN
5300  !
5310  !*** SUBROUTINE TO PRINT A SINGLE LINE OF BREATH-BY-BREATH INFORMATION
5320  !
5330 Hard_output: !
5340 !PRINT USING "#,70X,K,18X,K";"|";"|"
5350  PRINT
5360  PRINT USING "#,2X,DDDDD";Insp_count
5370  PRINT USING "#,5X,DD.DDD,3(4X,DD.DDD)";Airi;Aire;O2i;O2e
5380  PRINT USING "#,4X,D.DDD,5X,D.DDD,3X,K,1X,D.DDD";Co2i;Co2e;"|";O2cons
5390  PRINT USING "#,5X,D.DDD,2X,K";Co2prod;"|"
5400  PRINT USING "#,3X,D.DD,6X,D.DD";Insp_time;Exp_time
5410  IF Time_delay_flag<>2 THEN PRINT USING "#,6X,DDD";Time_delay
5420  IF Time_delay_flag=2 THEN PRINT USING "#,4X,K,DDD,K";"**";Time_delay;"**"
5430  PRINT
5440  RETURN
5450  !
5460  !*** SUBROUTINE TO PRINT TRIAL AVERAGE VALUE INFORMATION
5470  !
5480 Means:  !
5490  IF Correct$<>"Y" THEN H$=""
5500  IF Correct$<>"Y" THEN G$=""
5510 PRINT USING "K,DDD.D,K,4A";"Inspiratory minute volume = ";Minvoli;" liters
per minute ",G$
5520 PRINT USING "K,DDD.D,K,4A";"Expiratory minute volume = ";Minvole;" liters p
er minute ",G$
5530 PRINT USING "K,DDD.4D,K,4A";"Inspiratory tidal volume = ";Avoli;" liters ",
G$
5540  PRINT USING "K,DDD.4D,K,4A";"Expiratory tidal volume = ";Avole;" liters ",
G$
5550  PRINT USING "K,DDDD.D,K";"Respiratory frequency = ";Respf;" breaths per mi
nute"
5560  PRINT USING "K,DDD.3D,K,4A";"Mean O2 inspired = ";O2i_tidal;" liters ",H$
5570  PRINT USING "K,DDD.3D,K,4A";"Mean O2 expired = ";O2e_tidal;" liters ",H$
5580  PRINT USING "K,DDD.3D,K,4A";"Mean CO2 inspired = ";Co2i_tidal;" liters ",H
$
5590  PRINT USING "K,DDD.3D,K,4A";"Mean CO2 expired = ";Co2e_tidal;" liters ",H$
5600  PRINT USING "K,DDD.3D,K,4A";"Mean O2 consumed per breath = ";Avo2cons;" li
ters ",H$
5610  PRINT USING "K,DDD.3D,K,4A";"Mean CO2 produced per breath =";Avco2prod;" l
iters ",H$
5620  PRINT USING "K,DDD.3D,K,4A";"O2 consumed per minute = ";V_dot_o2;" liters
per minute ",H$
5630  PRINT USING "K,DDD.3D,K,4A";"CO2 produced per minute =";V_dot_co2;" liters
 per minute ",H$
5640  PRINT USING "K,DDD.3D";"RESPIRATORY QUOTIENT = ";R
5650  PRINT USING "K,3D.D,K";"Total time of inspiration = ";Tot_time_insp;" sec"
5660  PRINT USING "K,3D.D,K";"Total time of expiration = ";Tot_time_exp;" sec"
5670  PRINT USING "K,3D.D,K";"Total time of respiration = ";Tot_time_resp;" sec"
5680  PRINT USING "K,3D.D";"Number of good inspirations = ";Good_insp_count
5690  PRINT USING "K,3D.D";"Number of good expirations = ";Good_exp_count
5700  PRINT USING "K,3D.D";"Number of good breaths = ";No_breaths
5710  IF Correct$<>"Y" THEN 5760
```

```
5720  PRINT USING "K,DD.D,K";"Relative Humidity = ";Rel_humid*100;"%"
5730  PRINT USING "K,DD.DD,K";"Body Temperature = ";Body_temp;" deg C"
5740  PRINT USING "K,DD.DD,K,DD.3D,K";"PH2O at ";Body_temp;" deg C = ";Ph2o_body
;" torr"
5750  PRINT USING "K,3D.DD,K";"Barometric Pressure = ";Pb;" torr"
5760 PRINT USING "K,4D";"FLOW DC OFFSET = ";Bin_zero_flow
5770 PRINT USING "K,4D";"CO2 DC OFFSET = ";Co2_dc_offset
5780 PRINT USING "K,4D";"O2 DC OFFSET = ";O2_dc_offset
5790  PRINT USING "K,D.4DE";"CO2 CALIBRATION FACTOR = ";Co2_cal
5800 PRINT USING "K,D.4DE";"O2 CALIBRATION FACTOR = ";O2_cal
5810  PRINT USING "K,D.4DE";"INSPIRATORY FLOW CALIBRATION FACTOR = ";Insp_flow_c
al
5820  PRINT USING "K,D.4DE";"EXPIRATORY FLOW CALIBRATION FACTOR = ";Expr_flow_ca
l
5830  PRINT USING "3(K,MD.4DE)";"TEMPERATURE CORRECTION = ";Ta;"X^2 + ";Tb;"X +
";Tc
5840 PRINT "";"SAMPLING FREQUENCY =";S
5850  PRINT "";"FLOW CALIBRATION FILENAME:  ";Cal$
5860  PRINT "";"CO2 DATA FILENAME:  ";C1$
5870  PRINT "";"O2 DATA FILENAME:  ";O$
5880  PRINT "";"FLOW DATA FILENAME:  ";V$
5890  PRINT "";"TEMPERATURE DATA FILENAME:  ";T$
5900 PRINTER IS 1
5910  RETURN
5920  !
5930  !*** SUBROUTINE TO PLOT OUT THE FOUR BINARY DATA SETS
5940  !
5950 Dtplot:  PRINT "                    DATA PLOTTING ROUTINE"
5960  PRINT "THERE ARE ";No_points;" DATA POINTS AVAILABLE"
5970  !
5980  !*** OBTAIN STARTING AND ENDING POINTS TO PLOT
5990  !
6000  BEEP
6010  INPUT "ENTER STARTING POINT TO PLOT.",Start
6020  IF Start<1 OR Start>No_points THEN GOTO 6010
6030  BEEP
6040  INPUT "ENTER ENDING POINT TO PLOT.",End
6050  IF End<=Start OR End>No_points THEN GOTO 6040
6060  !
6070  !*** COMPUTE NUMBER OF POINTS TO PROCESS
6080  !
6090  P=End-Start
6100  !
6110  !*** COMPUTE PLOTTING OFFSET FOR MASS SPECTROMETER DELAYS
6120  !
6130  Offset=INT(Time_delay/1000*S)
6140  !
6150  !*** DETERMINE MAXIMUM AND MINIMUM VALUES FOR THE PLOTTED POINTS
6160  !
6170  Cmax=Line1(Start)
6180  Cmin=Line1(Start)
6190  Omax=Line2(Start)
6200  Omin=Line2(Start)
6210  Fmax=Line3(Start)
6220  Fmin=Line3(Start)
```

```
6230  Tmax=Line4(Start)
6240  Tmin=Line4(Start)
6250  FOR I=Start+1 TO End
6260  IF Cmax<Line1(I) THEN Cmax=Line1(I)
6270  IF Cmin>Line1(I) THEN Cmin=Line1(I)
6280  IF Omax<Line2(I) THEN Omax=Line2(I)
6290  IF Omin>Line2(I) THEN Omin=Line2(I)
6300  IF Fmax<Line3(I) THEN Fmax=Line3(I)
6310  IF Fmin>Line3(I) THEN Fmin=Line3(I)
6320  IF Tmax<Line4(I) THEN Tmax=Line4(I)
6330  IF Tmin>Line4(I) THEN Tmin=Line4(I)
6340  NEXT I
6350  !
6360  !*** DISPLAY THE MAXIMUM AND MINIMUM VALUES ON THE CRT
6370  !
6380  DISP
6390  PRINT "CO2 MAX: ";Cmax;TAB(25);"CO2 MIN: ";Cmin
6400  PRINT "O2 MAX: ";Omax;TAB(25);"O2 MIN: ";Omin
6410  PRINT "FLOW MAX: ";Fmax;TAB(25);"FLOW MIN: ";Fmin
6420  PRINT "TEMP MAX: ";Tmax;TAB(25);"TEMP MIN: ";Tmin
6430  !
6440  !*** ADJUST MAXIMUM AND MINIMUM PLOTTING VALUES FOR SMALL INPUT CHANGES
6450  !
6460  IF Fmax-Fmin>100 THEN 6490
6470  Fmax=4095
6480  Fmin=0
6490  IF Cmax-Cmin>100 THEN 6520
6500  Cmax=4095
6510  Cmin=0
6520  IF Omax-Omin>100 THEN 6550
6530  Omax=4095
6540  Omin=0
6550  IF Tmax-Tmin>100 THEN 6620
6560  Tmax=4095
6570  Tmin=0
6580  !
6590  !*** IF NO TEMPERATURE CALIBRATION DATA IS AVAILABLE, ONLY PLOT BINARY
6600  !*** TEMPERATURE DATA
6610  !
6620  IF (Ta<>0) AND (Tb<>0) THEN 6690
6630  Ta=0
6640  Tc=0
6650  Tb=1
6660  !
6670  !*** ALLOW USER TO SELECT PLOTTING DEVICE
6680  !
6690  BEEP
6700  INPUT "OUTPUT ON PLOTTER OR CRT ? (PLOTTER/CRT)",Q$
6710  !
6720  !*** SET DEFAULT PLOTTING DEVICE TO INTERNAL CRT AND DUMP DEVICE TO
6730  !*** THERMAL PRINTER
6740  !
6750  PLOTTER IS 3,"INTERNAL"
6760  DUMP DEVICE IS 801
6770  IF Q$<>"PLOTTER" THEN GOTO 6880
```

```
6780  !
6790  !*** SET SYSTEM FOR HP9872C PLOTTER
6800  !
6810  PLOTTER IS 705,"HPGL"
6820  PRINTER IS 705
6830  PRINT "VS5;"
6840  PRINTER IS 1
6850  !
6860  !*** INITIALIZE GRAPHICS SYSTEM
6870  !
6880  GRAPHICS ON
6890  GCLEAR
6900  PRINT CHR$(12)
6910  PEN 1
6920  DEG      !SET DEGREES MODE
6930  !
6940  !*** PLOT THE TEMPERATURE DATA ARRAY
6950  !
6960  VIEWPORT 10,120,4,24
6970  Ctmin=Ta*Tmin^2+Tb*Tmin+Tc
6980  Ctmax=Ta*Tmax^2+Tb*Tmax+Tc
6990  WINDOW -P/10,P,Ctmin,Ctmax
7000  LINE TYPE 1      !SET FOR SOLID LINE
7010  CSIZE 2.8
7020  LDIR 0
7030  LORG 2      !SET LABEL ORIGIN TO POSITION 8
7040  AXES P/20,(Ctmax-Ctmin)/10,0,Ctmin
7050  IF (Ctmin=Tmin) AND (Ctmax=Tmax) THEN 7120
7060  Incr=(Ctmax-Ctmin)/5
7070  FOR I=Ctmin+Incr TO Ctmax-Incr STEP Incr
7080  MOVE -P/10,I
7090  LABEL USING 7100;I
7100  IMAGE ZZ.DD
7110  NEXT I
7120  CSIZE 3.3      !SET CHARACTER HEIGHTH TO 3.3 GDU´S
7130  LINE TYPE 1
7140  PEN 2
7150  MOVE 1,Ta*Line4(Start)^2+Tb*Line4(Start)+Tc
7160  FOR I=Start TO End
7170  Temp=Ta*Line4(I)^2+Tb*Line4(I)+Tc
7180  PLOT I-Start+1,Temp
7190  NEXT I
7200  PEN 1
7210  CSIZE 2.5
7220  LINE TYPE 1
7230  !
7240  !*** LABEL THE TEMPERATURE PLOT
7250  !
7260  VIEWPORT 0,120,4,24
7270  WINDOW 0,125,4,24
7280  MOVE 1,14
7290  LORG 5
7300  LDIR 90
7310  LABEL "FLOW TEMP"
7320  MOVE 4,14
```

```
7330  LABEL "DEGREES C"
7340  LDIR 0
7350  VIEWPORT 0,125,24,28
7360  WINDOW 0,10,0,4
7370  MOVE 2,2
7380  LABEL "START POINT =";Start
7390  MOVE 5,2
7400  LABEL "END POINT =";End
7410  MOVE 8,2
7420  LABEL "1 TICK =";DROUND(P/(20*S),3);" SECONDS"
7430  !
7440  !*** PLOT THE FLOW DATA ARRAY
7450  !
7460  VIEWPORT 10,120,28,48
7470  WINDOW -P/10,P,Fmin,Fmax
7480  LINE TYPE 1
7490  CSIZE 2.8
7500  LORG 2
7510  Flow_cal=(Insp_flow_cal+Expr_flow_cal)/2
7520  IF Flow_cal=0 THEN Bin_zero_flow=2048
7530  AXES P/20,(Fmax-Fmin)/10,0,Bin_zero_flow
7540  IF Flow_cal=0 THEN 7640
7550  Incr=(Fmax-Fmin)/5
7560  FOR I=Bin_zero_flow TO Fmax STEP Incr  !LABEL Y-COORDINATE AXES (FLOW)
7570  MOVE -P/10,I
7580  LABEL USING 7100;(I-Bin_zero_flow)*Expr_flow_cal
7590  NEXT I
7600  FOR I=Bin_zero_flow-(Fmax-Fmin)/5 TO Fmin STEP (Fmin-Fmax)/5
7610  MOVE -P/10,I
7620  LABEL USING 7100;(I-Bin_zero_flow)*Insp_flow_cal
7630  NEXT I
7640  LDIR 0
7650  CSIZE 3.3
7660  LINE TYPE 1
7670  PEN 2
7680  MOVE 1,Line3(Start)
7690  FOR I=Start TO End
7700  PLOT I-Start+1,Line3(I)
7710  NEXT I
7720  PEN 1
7730  LINE TYPE 1
7740  CSIZE 2.5
7750  !
7760  !*** LABEL THE FLOW PLOT
7770  !
7780  VIEWPORT 0,120,28,48
7790  WINDOW 0,125,5,30
7800  MOVE 1,18
7810  LORG 5
7820  LDIR 90
7830  LABEL "FLOW [L/S]"
7840  !
7850  !*** PLOT THE O2 DATA ARRAY
7860  !
7870  VIEWPORT 10,120,52,72
```

```
7880  WINDOW -P/10,P,0,Omax
7890  AXES P/20,Omax/10
7900  LDIR 0
7910  CSIZE 2.8
7920  LORG 2
7930  IF O2_cal=0 THEN 7990
7940  IMAGE Z.DDD,X
7950  FOR I=Omax/5 TO Omax-Omax/5 STEP Omax/5
7960  MOVE -P/10,I
7970  LABEL USING 7940;I*O2_cal+O1
7980  NEXT I
7990  LDIR 90
8000  CSIZE 3.3
8010  LINE TYPE 1
8020  PEN 2
8030  MOVE 1,Line2(Start+Offset)-O2_dc_offset
8040  FOR I=Start TO End-Offset
8050  PLOT I-Start+1,Line2(I+Offset)-O2_dc_offset
8060  NEXT I
8070  PEN 1
8080  LINE TYPE 1
8090  !
8100  !*** LABEL THE O2 PLOT
8110  !
8120  VIEWPORT 0,120,52,72
8130  WINDOW 0,125,35,60
8140  MOVE 1,47
8150  LORG 5
8160  CSIZE 2.5
8170  LABEL "FRACTIONAL O2"
8180  MOVE 4,47
8190  LABEL "CONCENTRATION"
8200  LDIR 0
8210  LORG 6
8220  !
8230  !*** PLOT THE CO2 DATA ARRAY
8240  !
8250  VIEWPORT 10,120,76,96
8260  WINDOW -P/10,P,Cmin,Cmax
8270  AXES P/20,(Cmax-Cmin)/10,0,Cmin
8280  LDIR 0
8290  CSIZE 2.8
8300  LORG 2
8310  IF Co2_cal=0 THEN 8370
8320  Incr=(Cmax-Cmin)/5
8330  FOR I=Cmin+Incr TO Cmax-Incr STEP Incr
8340  MOVE -P/10,I
8350  LABEL USING 7940;(I-Co2_dc_offset)*Co2_cal
8360  NEXT I
8370  CSIZE 3.3
8380  LDIR 90
8390  LINE TYPE 1
8400  PEN 2
8410  MOVE 1,Line1(Start+Offset)
8420  FOR I=Start TO End-Offset
```

```
8430  PLOT I-Start+1,Line1(I+Offset)
8440  NEXT I
8450  PEN 1
8460  !
8470  !*** LABEL THE CO2 PLOT
8480  !
8490  VIEWPORT 0,120,76,96
8500  WINDOW 0,125,65,90
8510  LINE TYPE 1
8520  LORG 5
8530  MOVE 1,78
8540  CSIZE 2.3
8550  LABEL "FRACTIONAL CO2"
8560  MOVE 4,78
8570  LABEL "CONCENTRATION"
8580  Q$=""
8590  !
8600  !*** PUT PEN AWAY AND PAUSE FOR USER TO OBSERVE PLOT
8610  !
8620  PEN 0
8630  PAUSE
8640  !
8650  !*** ONCE PAUSE IS COMPLETE, PROMPT THE USER FOR REDO OF GRAPHICS
8660  !
8670  GRAPHICS OFF
8680  Q$=""
8690  BEEP
8700  INPUT "REDO GRAPHICS? (Y/N)",Q$
8710  !
8720  !*** IF DESIRED, GO START PLOTTING SUBROUTINE OVER
8730  !
8740  IF Q$="Y" OR Q$="y" THEN GOTO Dtplot
8750  !
8760  !*** OTHERWISE, RETURN BACK TO BEGIN BREATH-BY-BREATH ANALYSIS
8770  !
8780  RETURN
8790  !
8800  !*** SUBROUTINE FOR RETRIEVING FOUR CHANNELS OF BINARY DATA FROM STORAGE
8810  !
8820  !
8830  !*** GET THE NAMES OF THE FOUR FILES
8840  !
8850 Rtdata:  BEEP
8860  INPUT "ENTER THE CO2 SIGNAL FILE NAME",C1$
8870  BEEP
8880  INPUT "ENTER THE O2 SIGNAL FILE NAME",O$
8890  BEEP
8900  INPUT "ENTER THE FLOW SIGNAL FILE NAME",V$
8910  BEEP
8920  INPUT "ENTER THE FLOW TEMPERATURE SIGNAL FILE NAME",T$
8930  !
8940  !*** SELECT PROPER MASS STORAGE UNIT AND OPEN THE FILES
8950  !
8960  MASS STORAGE IS ":HP9895,700,0"
8970  ASSIGN @Path1 TO C1$
```

```
8980  ASSIGN @Path2 TO O$
8990  ASSIGN @Path3 TO V$
9000  ASSIGN @Path4 TO T$
9010  !
9020  !*** TELL PROGRAM WHEN TO QUIT READING THE FILES
9030  !
9040  ON END @Path1 GOTO 9160
9050  ON END @Path2 GOTO 9210
9060  ON END @Path3 GOTO 9260
9070  ON END @Path4 GOTO 9310
9080  !
9090  !*** READ THE CO2 DATA FILE
9100  !
9110  ENTER @Path1;Cmax,Cmin
9120  ENTER @Path1;Line1(*)
9130  !
9140  !*** READ THE O2 DATA FILE
9150  !
9160  ENTER @Path2;Omax,Omin
9170  ENTER @Path2;Line2(*)
9180  !
9190  !*** READ THE FLOW DATA FILE
9200  !
9210  ENTER @Path3;Fmax,Fmin
9220  ENTER @Path3;Line3(*)
9230  !
9240  !*** READ THE TEMPERATURE DATA FILE
9250  !
9260  ENTER @Path4;Tmax,Tmin
9270  ENTER @Path4;Line4(*)
9280  !
9290  !*** SET MASS STORAGE UNIT BACK TO INTERNAL FLOPPY
9300  !
9310  MASS STORAGE IS ":INTERNAL"
9320  !
9330  !*** RETURN BACK TO ANALYSIS ROUTINE
9340  !
9350  RETURN
9360  !
9370  !*** SUBROUTINE FOR RETRIEVING CALIBRATION FACTORS FROM MASS STORAGE
9380  !
9390  !
9400  !*** GET CALIBRATION FILE NAME AND ASSIGN PROPER MASS STORAGE UNIT
9410  !
9420 Rtcal:  BEEP
9430  INPUT "ENTER CALIBRATION FILE FILENAME",Cal$
9440  MASS STORAGE IS ":HP9895,700,0"
9450  !
9460  !*** OPEN CALBRATION FILE AND TELL PROGRAM WHEN TO STOP READING FILE
9470  !
9480  ASSIGN @Path1 TO Cal$
9490  ON END @Path1 GOTO 9590
9500  !
9510  !*** READ PARAMETERS IN CALIBRATION FILE
9520  !
```

```
9530 ENTER @Path1;Co2_dc_offset,02_dc_offset,Bin_zero_flow,Co2_cal,02_cal
9540 ENTER @Path1;Insp_flow_cal,Expr_flow_cal,Time_delay,S,01,Ta,Tb,Tc,Date$
9550 !
9560 !*** SET MASS STORAGE UNIT BACK TO INTERNAL DRIVE AND RETURN BACK TO
9570 !*** ANALYSIS ROUTINE
9580 !
9590 MASS STORAGE IS ":INTERNAL"
9600 RETURN
9610 !
9620 !*** SUBROUTINE TO DETERMINE MASS SPECTROMETER TIME DELAY ON A BREATH-
9630 !*** BY-BREATH BASIS
9640 !
9650 Bbb_time_delay:  Temp_a=A
9660 Temp_z=Z
9670 B=Bin_zero_flow      !SET B TO BINARY ZERO FLOW VALUE
9680 C=Co2_dc_offset      !SET C TO BINARY ZERO CO2 VALUE
9690 !
9700 !*** BEGINNING AT POINT ON FLOW SIGNAL CORRESPONDING TO ZERO FLOW,
9710 !*** LOCATE PEAK END EXPIRED CO2 VALUE
9720 !
9730 Hunt_max:  Z=A        !CO2 INDEX CORRESPONDING TO ZERO FLOW
9740 IF No_points-Z<50 THEN Bomb_out      !MAKE SURE 150 POINTS FOLLOW ZERO CROS
SING
9750 Co2max=Line1(Z)-C     !SET INITIAL CO2MAX LEVEL TO FIRST CO2 VALUE
9760 Max_index=Z
9770 FOR Wye=Z TO Z+.75*S   ! SEARCH AHEAD FOR THE MAX END EXPIRED FCO2 VALUE
9780 IF Line1(Wye)-C>Co2max THEN Max_index=Wye
9790 IF Line1(Wye)-C>Co2max THEN Co2max=Line1(Wye)-C
9800 NEXT Wye
9810 !
9820 !*** FIND THE MIDDLE INDEX (THAT POINT CORRESPONDING TO 50% OF THE MAXIMUM
9830 !*** END EXPIRED CO2 VALUE)
9840 !
9850 Mid_index=Max_index
9860 FOR Wye=Max_index TO Z+.75*S
9870 IF Line1(Wye)-C>.5*Co2max THEN Next_wye
9880 Mid_index=Wye ! INDEX OF THE 50% DOWN POINT ON FCO2 CURVE
9890 GOTO Set_limits
9900 Next_wye:  NEXT Wye
9910 !
9920 !*** FIND THE MINIMUM INDEX (THAT POINT CORRESPONDING TO THE MINIMUM END
9930 !*** EXPIRED CO2 VALUE
9940 !
9950 Set_limits:  Min_index=Mid_index
9960 Co2min=Line1(Mid_index)-C
9970 FOR Wye=Mid_index TO Mid_index+Mid_index-Max_index
9980 IF Line1(Wye)-C>Co2min THEN Next_y
9990 Min_index=Wye
10000 Co2min=Line1(Wye)-C
10010 Next_y: NEXT Wye
10020 !
10030 !*** INITIALIZE INDEXES FOR START OF INTEGRATION OF CO2 SIGNAL
10040 !
10050 Best_match=1.E+50
10060 Beg_pt=Max_index
```

```
10070 End_pt=Min_index
10080 !
10090 !*** EXIT ROUTINE IF ADEQUATE NUMBER OF POINTS DO NOT EXIST
10100 !
10110 IF End_pt>No_points THEN Bomb_out
10120 Beg_intg=Max_index
10130 End_intg=Min_index
10140 !
10150 !*** USE TRAPEZOIDAL RULE TO COMPUTE THE AREA ABOVE AND BELOW THE CURVE
10160 !
10170 New_sum: Asum=0
10180 Bsum=0
10190 !
10200 !*** FIRST, ABOVE THE CURVE, 1/2 OF FIRST AND LAST POINTS
10210 !
10220 Asum=.5*(Co2max-(Line1(Beg_pt)-C))+.5*(Co2max-(Line1(Beg_intg)-C))
10230 FOR Wye=Beg_pt+1 TO Beg_intg-1
10240 Asum=Asum+Co2max-(Line1(Wye)-C)
10250 NEXT Wye
10260 !
10270 !*** NEXT, BELOW THE CURVE, 1/2 OF FIRST AND LAST POINTS
10280 !
10290 Bsum=.5*(Line1(Beg_intg)-Co2_dc_offset)+.5*(Line1(End_pt)-Co2_dc_offset)
10300 FOR Wye=Beg_intg+1 TO End_pt-1
10310 Bsum=Bsum+Line1(Wye)-Co2_dc_offset
10320 NEXT Wye
10330 !
10340 !*** COMPUTE DIFFERENCE IN THE TWO AREAS
10350 !
10360 Adiff=ABS(Asum-Bsum)
10370 !
10380 !*** IF AREA DIFFERENCE IS A MINIMUM, REMEMBER THE PROPER INDEX
10390 !
10400 IF Adiff<Best_match THEN Best_index=Beg_intg
10410 IF Adiff<Best_match THEN Best_match=Adiff
10420 !
10430 !*** BUMP THE CENTER INTEGRATION POINT AND GO TRY ANOTHER IF STILL
10440 !*** WITHIN OUTER LIMITS OF INTEGRATION
10450 !
10460 Beg_intg=Beg_intg+1 ! IF NOT TO ENDPOINT SHIFT THE CENTER INTGR PONT
10470 IF Beg_intg<=End_intg THEN New_sum ! GO COMPUTE NEW AREAS FOR LIMITS
10480 !
10490 !*** COMPUTE MASS SPECTROMETER TIME DELAY FOR THIS BREATH AND RETURN
10500 !*** BACK TO ANALYSIS ROUTINE WITH VARIABLES UNALTERED.
10510 !
10520 Bomb_out: ! DATA STREAM EXHAUSTED
10530 Time_delay=(Best_index-Z)/S*1000
10540 A=Temp_a
10550 Z=Temp_z
10560 RETURN
10570 END
```

A COMPUTER-BASED INSTRUMENTATION SYSTEM FOR MEASUREMENT OF BREATH-BY-BREATH OXYGEN CONSUMPTION AND CARBON DIOXIDE PRODUCTION IN EXERCISING HUMANS

by

LOREN EUGENE RIBLETT, JR.

B.S., Kansas State University, 1983

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

1984

ABSTRACT

A complete computer-controlled instrumentation system has been developed to monitor O_2 consumption and CO_2 production on a breath-by-breath basis in exercising humans. Using a custom built data acquisition module, four physiological signals can be monitored, namely fractional concentrations of CO_2 and O_2, respiratory flow, and respiratory flow temperature. In addition to the various transducers necessary to measure the fore mentioned signals, equipment for calibrating these transducers have also been integrated into the complete system.

Calibration, instrument control, and data analysis software has been developed and documented. Using combinations of BASIC, Pascal, and 68000 assembly language routines, the breath-by-breath measurement system can be calibrated, data can be taken and stored in various forms, and ultimately the data can be analyzed and displayed in both tabular and graphical form.

For system verification, comparisons have been made between the breath-by-breath system and an end-expired bag collection technique. Well defined exercise programs were developed and the test subject was carefully selected so comparison of results from the two techniques could be made. Results from these experiments indicate that the breath-by-breath system is as precise and accurate as the bag collection technique in steady-state conditions. The breath-by-breath system has the added advantage of being able to analyze transient respiratory phenomena.